BAM! Just Like That: Simple and Efficient Parameter Upcycling for Mixture of Experts

Anonymous Authors[1]

Abstract

Training Mixture of Experts (MoEs) from scratch in a large-scale regime is expensive. Previous work addresses this challenge by independently training multiple dense expert models and using them to initialize an MoE. In particular, initializing MoE layers using experts' feed-forward parameters while merging all other parameters. This limits the advantages of the specialized dense models when "upcycling" them as MoEs. We propose BAM (Branch-Attend-Mix), a simple yet effective improvement to MoE training. BAM makes full use of specialized dense models by not only using their feed-forward network (FFN) to initialize the MoE layers but also leveraging experts' attention weights fully by initializing them as Mixture of Attention (MoA) layers. Our experiments using seed models ranging from 590 million to 2 billion parameters show that our approach outperforms state-of-the-art approaches under the same data and compute budget in both perplexity and downstream tasks evaluations.

1. Introduction

Mixture of Experts (MoE) (Fedus et al., 2022; Zoph et al., 2022; Shazeer et al., 2017; DeepSeek-AI, 2024; Jiang et al., 2024; Gale et al., 2023) have emerged as a popular alternative architecture to dense models. During training and inference, each input activates only a subset of the MoE model parameters (often referred to as experts), thereby decoupling computation cost from the total parameter count, allowing the total number of parameters to grow without additional computation. Consequently, MoEs empirically outperform dense models with equivalent computational requirements (Fedus et al., 2022; Krajewski et al., 2024).

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

However, training MoEs (i.e. starting from randomly initialized weights) is expensive (Gale et al., 2023; Fedus et al., 2022) and hard (Zoph et al., 2022). To address this, recent works have explored more efficient alternatives by initializing MoEs using pre-trained dense models (Komatsuzaki et al., 2022; Sukhbaatar et al., 2024), followed by a small number of MoE fine-tuning steps. In particular, Branch-Train-MiX (BTX) (Sukhbaatar et al., 2024) initializes an MoE with N FFN experts through a three-step upcycling process. Initially, N copies of a pre-trained seed model are created. Each copy is then further pre-trained on domain-specific data for specialization. The MoE's FFN expert parameters are subsequently initialized from these specialized dense models by directly copying their FFN parameters. It's important to note that the MoE's non-FFN parameters, such as attention layers, are not converted to MoE layers. Since there are N dense models, each with its own set of attention parameters, BTX uses the uniform average of the N attention parameters to initialize the MoE's attention module while the router parameters are initialized randomly.

However, this approach fails to take full advantage of the specialized dense models: 1) If we are already training N specialized dense models with N specialized attention modules, we should also leverage the expert knowledge from these attention modules of specialized dense models; 2) uniformly averaging model parameters from different models, even when branched from the same seed model, is known to degrade performance (Li et al., 2022).

Motivated by this, we propose Branch-Attend-Mix (BAM), a simple yet effective improvement to BTX that upcycles both feed-forward and attention parameters into expert layers. See 1 for an illustration of BAM. In order to fully leverage the specialized attention experts and improve training stability, BAM uses a variant of Mixture of Attention (MoA; Zhang et al., 2022) with soft routing where each token is assigned to every attention expert. To account for the increased computations from soft routing MoA, BAM employs a parallel-attention transformer architecture (Wang and Komatsuzaki, 2021; Chowdhery et al., 2023) that allows the attention experts and FFN experts to be computed in parallel for better throughput.

We conduct experiments using seed models ranging from

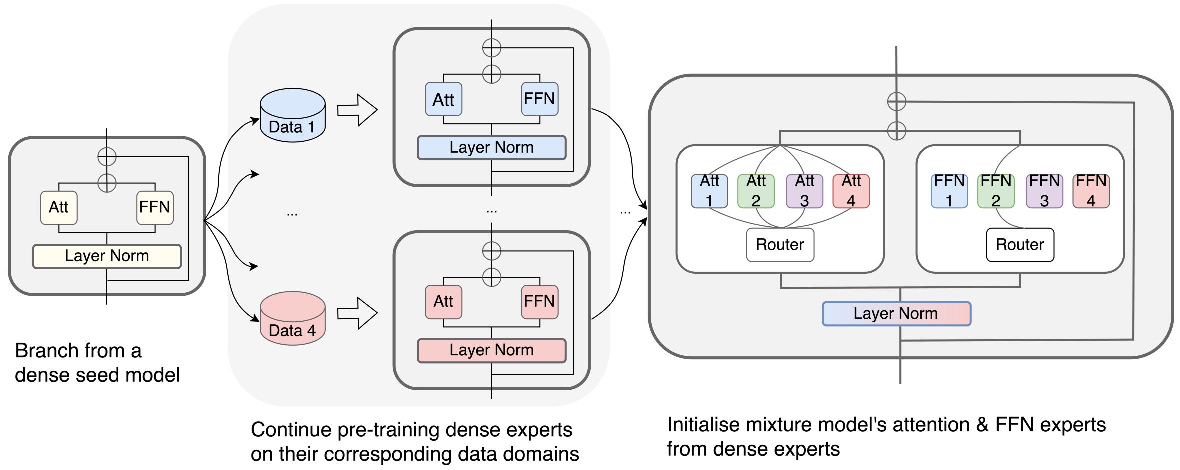
Att
FFN
Layer Norm
Branch from a dense seed model
Data 1
Data 4
Continue pre-training dense experts on their corresponding data domains
Att 1
Att 2
Att 3
Att 4
FFN 1
FFN 2
FFN 3
FFN 4
Router
Router
Layer Norm
Initialise mixture model's attention & FFN experts from dense experts

Figure 1. BAM operates in three phases: branching continued pre-training, mixture model training phase [†].

590 million to 2 billion parameters and show that our approach outperforms BTX under the same data and compute budget in both perplexity and downstream evaluations on various domains, confirming the effectiveness of our proposed method.

2. Background

2.1. Parallel-Attention Transformers

The default architecture for LLMs is based on stacking multiple blocks of the transformers (Vaswani et al., 2017). The original transformer architecture consists of a multi-headed attention (MHA), often referred to as the attention layer, a residual connection, followed by a feed-forward neural network (FFN). Recent works (Wang and Komatsuzaki, 2021; Chowdhery et al., 2023) introduced the parallel-attention transformer, a variant of the original architecture, which computes the output of the attention and the FFN layers in parallel using the same input. The final transformer output is the projected concatenation of the attention output and the FFN output with its residual connection (See Figure 1). Processing the two layers in parallel increases the computational throughput without degrading the performance.

2.2. Multi-Headed Attention

In an attention layer with h heads, the input is first linearly projected to generate the query, key, and value matrices through learned projections. For each head, the outputs are calculated using the projected matrices. The resulting outputs from different heads are concatenated and then projected back to the original model dimension size:

$$\text{Head}_i = \text{softmax}\left(\frac{QW_i^q(KW_i^k)^T}{\sqrt{d_k}}\right)VW_i^v$$
$$\text{MHA}(Q,K,V) = \text{concatenate}\left(\text{head}_1,\ldots,\text{head}_\text{h}\right)W^o, \quad (1)$$

where $W_i^q, W_i^k \in \mathbb{R}^{d_\text{model}\times d_k}$, $W_i^v \in \mathbb{R}^{d_\text{model}\times d_v}$ and $W^o \in \mathbb{R}^{hd_v\times d_\text{model}}$, d_k is the key dimension.

2.3. Mixture of Experts

The Mixture of Experts (MoE; Shazeer et al., 2017) model differs from the vanilla transformer by replacing the FFN with an MoE Layer. An MoE Layer consists of a linear router parameterized by $W_\text{FFN router} \in \mathbb{R}^{d_\text{model}\times N}$ and a collection of N FFN experts $\{\text{FFN}_i(x)\}_{i=1}^N$. The normalized router logits are defined as $p(x) = \text{softmax}\left(W_\text{FFN router}\cdot x\right)$, where each element $p_i(x)$ represents the gate value for the i-th expert, FFN_i. The router assigns each input token representation x to the k experts with the highest gate values. Denoting the set of selected top-k indices as $\mathcal{T}$, the final output of the MoE layer is the gate-value weighted combination of each selected expert's computation, given by:

$$\text{FFN}_\text{MoE} = \sum_{i\in\mathcal{T}} p_i(x)\text{FFN}_i(x).$$

2.4. Mixture of Attention

In addition to replacing FFNs in the dense architecture with MoE layers, Mixture of Attention (MoA; Zhang et al., 2022; Shen et al., 2024) also replaces the attention layer in the dense architecture with MoA layers. An MoA layer consists of a linear router parameterized by the router weight $W_\text{attn router} \in \mathbb{R}^{d_\text{model}\times N}$ and a set of N attention experts, $\{\text{MHA}_j(x)\}_{j=1}^N$. The normalized router logits are defined as $g(x) = \text{softmax}\left(W_\text{attn router}\cdot x\right)$, where each element $g_j(x)$ represents the gate value for the j-th expert, MHA_j. Each attention expert is made up of Q-projection experts $\left\{W_j^q\right\}_{j=1}^N$ and the attention output projection experts $\left\{W_j^o\right\}_{j=1}^N$. The KV-projections W^k and W^v are shared among all experts for computation efficiency. Denoting the set of selected top-k indices as $\mathcal{M}$, the final output of the

MoA layer is given by:

$$\mathrm{MHA}_{\mathrm{MoA}} = \sum_{i \in \mathcal{M}} g_i(x)\mathrm{MHA}_i(x).$$

3. BAM: Branch-Attend-Mix

BAM operates in three phases. We describe them in more detail below:

1) Branching: we create N copies of the model from a pre-trained dense model. Specifically for BAM, we propose to use a pre-trained seed model with a parallel-attention transformer architecture. This choice is motivated by two reasons: 1) Compared to vanilla transformers, parallel-attention transformers offer better throughput with minor degradation in performance at small scales but no degradation at larger scales. This has been demonstrated empirically in works like PALM (Chowdhery et al., 2023) and GPT-J (Wang and Komatsuzaki, 2021). 2) A seed model with parallel attention blocks allows us to have a parallel-attention mixture model. This enables parallel computation of attention and FFN experts, which improves throughput of BAM during training and inference.

2) Continued Pre-training: we continue pre-training each copy of the seed model independently on its own data mixture covering different specialized domains, namely, law, math, and coding. This phase yields specialized dense expert models.

3) Mixture Model Training: we initialize the experts in the mixture model using the specialized dense models from above. In addition, we also keep a copy of the original pre-trained seed model to the mixture (similar to BTX) so that the seed model's general knowledge learned in the pre-training stage is transferred to the mixture model. What sets BAM apart is the initialization of the mixture model, which takes full advantage of the dense specialized models by leveraging both the FFN and attention experts. We directly use the expert parameters from each dense model to initialize the corresponding mixture parameters. Layer normalization, input embedding layer, and output embedding layer parameters of the mixture model are initialized by uniformly averaging the corresponding parameters in each of the dense models. The router layers are initialized randomly. Upcycling attention experts is beneficial for two reasons: 1) It allows us to take full advantage of the specialized dense models; and 2) it avoids uniformly averaging attention parameters, which has shown to degrade model performance (Li et al., 2022). In the following, we discuss in more detail how attention experts and FFN experts are trained.

3.1. Mixture Model Training: Attention Experts

The MoA architecture (Zhang et al., 2022) uses only the query projection (W^q) and the attention output as MoA experts, while sharing the KV projections among all experts for efficiency. In this work, we employ a soft version of MoA where the attention router assigns each token to all attention experts. We also propose a variant that also uses the key-value projections (W^k, W^v) as MoA experts. These decisions are based on three reasons: 1) to fully leverage specialized attention experts, 2) since FFNs already dominate transformer computations (Zhang et al., 2023; Karpathy, 2023), we can afford a more expensive attention operation using parallel transformers, 3) soft-routing allows for more stable training since it is no longer a discrete optimization problem and is not subjected to common MoE training problems such as imbalanced router load (Fedus et al., 2022; Zoph et al., 2022) .

These choices involve a trade-off between efficiency and performance, which we discuss in Section B.4 on how to evaluate fairly. From here on, we refer to upcycling into soft-MoA with all attention parameters as *BAM with KV experts*, and soft-MoA with KV projections shared among experts as simply *BAM with KV sharing*.

4. Experiment Details

We validate BAM empirically through two sets of experiments, varying in scale. The first set uses seed model with 590 million parameters and the second set with 2 billion parameters. For both set of experiments, we train a mixture model with 4 experts: a general pre-training expert (seed model), a code expert, a math expert, and a law expert. See the Appendix for model architecture and experiment details.

4.1. Evaluation

We compare BAM to BTX as well as the dense models which they are upcycled from (see the Appendix for more evaluation details). We evaluate BAM in two settings. 1) **Data-Matching (DM)**: We use exactly the same training data as our BTX baseline. 2) **Compute-Matching (CM)**: We train BAM and BTX with the same amount of TPU-core-days.

5. Results

We evaluate BAM against baselines on both perplexity and benchmark tasks. For the case of sharing KV projections among MoA experts, we refer to it as *BAM (shared KV)* in the results tables. For the case of using all attention parameters as MoA experts, we refer to it as *BAM (Expert KV)*.

We show perplexity evaluation on small-scale experiments

	Pretrain	Code	Law	Math	Average
Base Model	**23.98**	8.62	8.92	14.12	13.91
Specialized Models					
Code Dense Expert	39.50	3.71	10.83	8.77	15.70
Law Dense Expert	87.82	23.45	7.53	30.79	37.40
Math Dense Expert	57.61	6.78	10.43	5.67	20.12
Generalist Models					
BTX (Baseline)	26.72	3.78	6.63	5.77	10.72
BAM DM (Expert KV), ours	24.02	**3.45**	**6.04**	**5.31**	**9.70**
BAM CM (Expert KV), our	24.84	3.53	6.21	5.43	10.00
BAM DM (Shared KV), ours	25.67	3.64	6.39	5.57	10.32
BAM CM (Shared KV), ours	26.00	3.72	6.50	5.63	10.46

Table 1. Perplexity evaluation (↓) of BAM versus BTX for small scale experiments.

	Pretrain	Code	Law	Math	Average
Base Model	**9.83**	2.41	3.86	3.71	3.33
Specialized Models					
Code Dense Expert	15.39	**2.18**	5.22	4.34	3.91
Law Dense Expert	32.69	6.84	**3.09**	8.61	6.18
Math Dense Expert	20.32	3.20	5.11	**3.20**	3.84
Generalist Models					
BTX (Baseline)	10.35	2.40	3.76	3.64	3.27
BAM DM (Expert KV), ours	10.11	2.36	3.66	3.55	**3.19**
BAM CM (Expert KV), our	10.19	2.37	3.69	3.57	3.21
BAM DM (Shared KV), ours	10.20	2.37	3.69	3.59	3.22
BAM CM (Shared KV), ours	10.28	2.38	3.72	3.61	3.24

Table 2. Perplexity evaluation (↓) of BAM versus BTX for large-scale experiments.

in Table 1. BAM outperforms BTX under both the compute and token-matching regimes. Even though the dense seed model achieves the best perplexity on the general pretraining dataset, it lags far behind all generalist models in all other data domains. All BAM models achieve the best average perplexity out of all models.

See Table 2 for perplexity results on large-scale experiments. BAM again outperforms the BTX baseline under both the compute-matching and token-matching regimes. The dense models are better than generalist models only when evaluating the corresponding expert domain. However, the dense models still lag far behind all generalist models in all other data domains. All BAM models achieve a lower average perplexity than the BTX baseline.

Finally, we also show BAM outperforms BTX in most downstream tasks evaluation domains, and results are shown in Table 3 in the Appendix.

6. Conclusion

In this work, we propose BAM (Branch-Attend-Mix), a simple yet effective improvement for upcycling dense models into mixture models. In addition to upcycling just the FFN parameters, we also upcycle the dense model's attention parameters into Mixture of Attention (MoA). In order to fully leverage the specialized attention experts and improve training stability, BAM uses a variant of MoA with soft-routing, where each token is assigned to every attention expert. To account for the increased computations from soft-routing MoA, BAM employs a parallel-attention transformer architecture that allows the attention experts and FFN experts to be computed in parallel for better throughput. We evaluate BAM empirically against baselines on seed models ranging from 590 million to 2 billion parameters. We show that our approach outperforms the baseline BTX under the same data and compute budget in both perplexity and downstream task evaluations, confirming the effectiveness of our proposed method.

References

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. St-moe: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*, 2022.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. Megablocks: Efficient sparse training with mixture-of-experts. *Proceedings of Machine Learning and Systems*, 5, 2023.

Jakub Krajewski, Jan Ludziejewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, et al. Scaling laws for fine-grained mixture of experts. *arXiv preprint arXiv:2402.07871*, 2024.

Sainbayar Sukhbaatar, Olga Golovneva, Vasu Sharma, Hu Xu, Xi Victoria Lin, Baptiste Rozière, Jacob Kahn, Daniel Li, Wen-tau Yih, Jason Weston, et al. Branch-train-mix: Mixing expert llms into a mixture-of-experts llm. *arXiv preprint arXiv:2403.07816*, 2024.

Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. Sparse upcycling: Training mixture-of-experts from dense checkpoints. *arXiv preprint arXiv:2212.05055*, 2022.

Margaret Li, Suchin Gururangan, Tim Dettmers, Mike Lewis, Tim Althoff, Noah A Smith, and Luke Zettlemoyer. Branch-train-merge: Embarrassingly parallel training of expert language models. *arXiv preprint arXiv:2208.03306*, 2022.

Xiaofeng Zhang, Yikang Shen, Zeyu Huang, Jie Zhou, Wenge Rong, and Zhang Xiong. Mixture of attention heads: Selecting attention heads per token. *arXiv preprint arXiv:2210.05144*, 2022.

Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. `https://github.com/kingoflolz/mesh-transformer-jax`, May 2021.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24 (240):1–113, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Yikang Shen, Zhen Guo, Tianle Cai, and Zengyi Qin. Jetmoe: Reaching llama2 performance with 0.1 m dollars. *arXiv preprint arXiv:2404.07413*, 2024.

Longteng Zhang, Xiang Liu, Zeyu Li, Xinglin Pan, Peijie Dong, Ruibo Fan, Rui Guo, Xin Wang, Qiong Luo, Shaohuai Shi, et al. Dissecting the runtime performance of the training, fine-tuning, and inference of large language models. *arXiv preprint arXiv:2311.03687*, 2023.

Andrej Karpathy. I fixed the transformer diagram, 2023. URL `https://x.com/karpathy/status/1658161721251602432`.

Hao Peng, Roy Schwartz, Dianqi Li, and Noah A Smith. A mixture of $h - 1$ heads is better than h heads. *arXiv preprint arXiv:2005.06537*, 2020.

Róbert Csordás, Piotr Piekos, and Kazuki Irie. Switchhead: Accelerating transformers with mixture-of-experts attention. *arXiv preprint arXiv:2312.07987*, 2023.

Takuya Akiba, Makoto Shing, Yujin Tang, Qi Sun, and David Ha. Evolutionary optimization of model merging recipes, 2024.

Yikang Shen, Zheyu Zhang, Tianyou Cao, Shawn Tan, Zhenfang Chen, and Chuang Gan. Moduleformer: Modularity emerges from mixture-of-experts, 2023.

Zhen Yang, Ming Ding, Qingsong Lv, Zhihuan Jiang, Zehai He, Yuyi Guo, Jinfeng Bai, and Jie Tang. Gpt can solve mathematical problems without a calculator. *arXiv preprint arXiv:2309.03241*, 2023.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics, 2023.

Zengzhi Wang, Rui Xia, and Pengfei Liu. Generative ai for math: Part i–mathpile: A billion-token-scale pretraining corpus for math. *arXiv preprint arXiv:2312.17120*, 2023.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

Peter Henderson, Mark Krass, Lucia Zheng, Neel Guha, Christopher D Manning, Dan Jurafsky, and Daniel Ho. Pile of law: Learning responsible data filtering from the law and a 256gb open-source legal dataset. *Advances in Neural Information Processing Systems*, 35:29217–29234, 2022.

Mirac Suzgun, Luke Melas-Kyriazi, Suproteem Sarkar, Scott D Kominers, and Stuart Shieber. The harvard uspto patent dataset: A large-scale, well-structured, and multi-purpose corpus of patent applications. *Advances in Neural Information Processing Systems*, 36, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021a.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. 2021.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021.

Neel Guha, Julian Nyarko, Daniel Ho, Christopher Ré, Adam Chilton, Alex Chohlas-Wood, Austin Peters, Brandon Waldon, Daniel Rockmore, Diego Zambrano, et al. Legalbench: A collaboratively built benchmark for measuring legal reasoning in large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: a benchmark for question answering research. *Transactions of the Association of Computational Linguistics*, 2019.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension, 2017.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv*, abs/1803.05457, 2018. URL https://api.semanticscholar.org/CorpusID:3922816.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. Socialiqa: Commonsense reasoning about social interactions, 2019.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale, 2019.

Eunsol Choi, He He, Mohit Iyyer, Mark Yatskar, Wen tau Yih, Yejin Choi, Percy Liang, and Luke Zettlemoyer. Quac : Question answering in context, 2018.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021b.

A. Related Work

Efficient initialization for MoEs Two recent works have studied how to use pre-trained dense models to initialize MoEs (Komatsuzaki et al., 2022; Sukhbaatar et al., 2024). Sparse Upcycling (Komatsuzaki et al., 2022) upcycles a single dense model by simply copying over dense model parameters as MoE parameters. It replicates FFN parameters N times into each FFN expert in sparse MoE layers. BTX (Sukhbaatar et al., 2024) improves on this approach by upcycling not just a single dense seed model but also multiple specialized dense models branched from the seed model. This is advantageous because not only can the dense models be trained in parallel, but also they are specialized in particular domains. However, unlike our method, both approaches only upcycle the FFN part of the dense (seed or specialized) models.

Alternative architecture for MoEs Alternative to the standard MoE architectures where only feed-forward blocks (FFNs) are used as MoE layers, recent work (Zhang et al., 2022; Peng et al., 2020; Csordás et al., 2023; Shen et al., 2024) investigates the mixture-of-attention framework, which uses attention experts in addition to FFN experts. This approach has not gained widespread popularity because it only achieves modest performance improvements while requiring additional engineering tricks for optimization (Csordás et al., 2023). However, in the setting of upcycling specialized dense experts into an MoE, the usage of attention experts is much more appropriate as the attention parameters contain specialized domain knowledge that would otherwise not be accessible in the final MoE.

Alternatives Methods for Model Merging Recent work has tried to combine many openly available, dense models to create an improved model, via layerwise merging or stitching of pre-trained model weights (Akiba et al., 2024). However, this approach creates a new dense model, not a sparse one. Other work has looked into dynamically adding new blocks to MoA whenever the model is trained on a new domain, but uses random initialization for the new experts instead of utilizing existing models (Shen et al., 2023).

B. Experiment Details

B.1. Mixture Model Training Phase: FFN Experts

We use top-1 routing of FFN experts for all experiments unless otherwise stated. For training stability, we use a load balancing loss $\mathcal{L}_{\text{LB}}$ (Fedus et al., 2022) and a router z-loss (Zoph et al., 2022), where the latter is noted to be important in (Zhang et al., 2022; Shen et al., 2024). Let B be the number of tokens in the batch, f_i be the fraction of tokens assigned to FFN expert i, and P_i be the fraction of the expert i's gate value, then the two auxiliary losses are as follows:

$$\begin{aligned} \mathcal{L}_{\text{LB}} &= N \sum_{i=1}^{N} f_i P_i \\ \mathcal{L}_z &= \frac{1}{B} \sum_{i=1}^{B} \left(\text{LogSumExp}_{j=1:N} \left(x_j^i \right) \right)^2 \end{aligned} \tag{2}$$

The model's final loss function $\mathcal{L}$ is a weighted sum, where $\mathcal{L}_{\text{NLL}}$ is the negative log-likelihood loss, and α and β are hyperparameters.

$$\mathcal{L} = \mathcal{L}_{\text{NLL}} + \sum_{\forall \text{ MoE Layers}} \left(\alpha \mathcal{L}_{\text{LB}} + \beta \mathcal{L}_z \right) \tag{3}$$

B.2. Experiments

Small-Scale Experiments Our first set of experiments involves a dense seed model with 590 million parameters, pre-trained on 400 billion tokens. We create three copies of this seed model, each of which is then further trained on a specialized data domain for 100 billion tokens. For all experiments, we use the AdamW optimizer with a weight decay of 0.1. The batch size is set to 1 million tokens. During the pre-training and mixture model training phases, we employ a learning rate schedule that warms up from 0 to $1.2e-4$ over 1500 steps, then decays to a tenth of the peak value using a cosine schedule. For the continued training phase, we use a learning rate of half the pre-training peak to ensure training stability.

	Math	Code	Law	Know.	Reason.	MMLU	Average
Seed Model	3.68%	9.41%	73.34%	**21.33%**	**47.73%**	34.13%	31.60%
Specialized Models							
Math Dense Expert	**4.92%**	12.39%	68.21%	13.32%	46.11%	34.29%	29.87%
Code Dense Expert	3.19%	**18.80%**	21.49%	12.18%	44.29%	31.50%	21.91%
Law Dense Expert	3.05%	0.20%	**88.80%**	10.41%	44.08%	32.18%	29.79%
Generalist Models							
BTX (Baseline)	3.86%	10.05%	81.85%	19.07%	47.36%	34.07%	32.71%
BAM DM (Expert KV), ours	4.44%	12.83%	85.47%	19.89%	47.11%	34.42%	**34.02%**
BAM CM (Expert KV), ours	4.34%	12.48%	82.79%	19.51%	47.43%	34.43%	33.50%
BAM DM (Shared KV), ours	4.10%	11.76%	86.73%	19.48%	47.27%	**34.55%**	33.98%
BAM DM (Shared KV), ours	3.65%	11.77%	80.98%	19.22%	47.56%	34.16%	32.89%

Table 3. Benchmark evaluations (↑) for BAM versus BTX. Table shows large-scale experiments (using a seed model of **2B parameters**). Highlighted entries indicate models outperform the BTX baseline. All BAM variants outperform BTX on average across all domains.

Large-Scale Experiments The second set of experiments utilizes a dense seed model with 2 billion parameters, pre-trained on 750 billion tokens. Similarly, we create three copies of this seed model, each continuing pre-training on its own data domain for an additional 100 billion tokens. We use the AdamW optimizer with a weight decay of 0.1. The batch size is set to 4 million tokens. During the pre-training phase, we employ a learning rate schedule that warms up from 0 to $1e-2$ over 2000 steps, then cosine-decays to $3e-4$. For the continued training phase, we use a learning rate that is half of the pre-training peak to ensure training stability. For the mixture model training phase, we encountered gradient spikes when using the same learning rate schedule as in the pre-training phase. To address this, we adopted a lower peak learning rate of $1e-4$, which decays to $1e-5$ with 1000 warmup steps, following the same schedule as BTX.

B.3. Data Details

In the **continued pre-training phase**, each dense expert is trained on its data domain for 100 billion tokens. The data domains contain the following data:

- **Math**: the MathGLM (Yang et al., 2023), GSM8K (Cobbe et al., 2021), proof-pile-2 (Azerbayev et al., 2023), and MathPile (Wang et al., 2023) datasets. Note that there is no overlap between the GSM8K train and evaluation split.
- **Code**: the Starcoder dataset (Li et al., 2023).
- **Law**: the pile-of-law (Henderson et al., 2022) and HUPD (Suzgun et al., 2024) datasets.

Additionally, we incorporate 10% text data from Common Crawl into each domain .

In the **mixture model training phase**, we use a data mixture composed of 25% of each expert domain, plus 25% of pre-training data used for training the seed model. The data mixture weights for the validation set are the same as for the training set.

B.4. Evaluation

We compare BAM to BTX trained in the same setting as well as the original dense seed model and the dense expert models. We evaluate BAM in two settings:

- **Data-Matching (DM)**: We use exactly the same training data as our BTX baseline.

- **Compute-Matching (CM)**: We train BAM and BTX with the same amount of TPU-core-days. In the small-scale experiments, all MoE training phases take 25 TPU-core-days. In the large-scale experiments, all MoE training phases take 305 TPU-core-days.

In addition to perplexity, we also evaluate large-scale experiments' model performances on a variety of domains with zero-shot and few-shot downstream tasks. For each domain, we report the average of scores of tasks in that domain.[‡] We evaluate on the following general and domain-specific downstream tasks:

- **Math**: we report the average performance on GSM8K (8-shot) (Cobbe et al., 2021) and MATH (4-shot) (Hendrycks et al., 2021a) for mathematical reasoning.
- **Code**: we report the average performance on HumanEval (0-shot) (Chen et al., 2021) and MBPP (3-shot) (Austin et al., 2021) for code generation.
- **Law**: we report the performance on the International Citizenship Questions sub-task in LegalBench (Guha et al., 2024).
- **World Knowledge**: we report the average performance on Natural Questions (0-shot) (Kwiatkowski et al., 2019) and TriviaQA (0-shot) (Joshi et al., 2017) for fact knowledge.
- **Reasoning**: we report the average performance on ARC-Easy, ARC-Challenge (Clark et al., 2018), SIQA (Sap et al., 2019), PIQA (Bisk et al., 2019), WinoGrande (Sakaguchi et al., 2019), and QUAC (Choi et al., 2018) (all 0-shot) for reasoning abilities.
- **General Knowledge**: we report the performance on MMLU (5-shot) (Hendrycks et al., 2021b) for general language understanding.

C. Model Architecture Details

	Large Scale	**Small Scale**
Embedding dimension	2304	1024
FFN dimension	18432	4096
# Heads	18	8
# KV heads	6	8
Vocabulary Size	256000	256000
Activation Function	swiglu	swiglu
# layers	18	6
Positional Embedding	rope	rope
Share Input & Output Embedding	Yes	No
Seed Model Parameters	2 Billion	590 Million

Table 4. Model architecture details for large and small scale experiments

.

C.1. Benchmark Evaluations

See Table 3 for benchmark evaluations on large scale experiments, BAM models alaway outperforms BTX on the average benchmark scores.

D. Ablations

We ablate the importance of upcylcling attention experts in BAM by comparing BTX with matching total parameters and / or active parameters. All ablation experiment use the same number of 100B tokens in the mixture model training phase. See Table 5 in the Appendix for a detailed walk-through on calculating the number of total and active parameters of each model.

[‡]For each experiment setting, we only include a benchmark if the maximum score achieved by any model is above random guessing. For LegalBench, we chose International Citizenship QA because models scored above random among the few sub-task implemented in our code-base.

	Dense	**BAM**	**BAM (KV sharing)**	**BTX top-1**	**BTX top-3**
layernorm / Block	1024	1024	1024	1024	1024
attn_out / Block	1,048,576	4,194,304	4,194,304	1,048,576	1,048,576
qkv_proj / Block	3,145,728	1,258,2912	4,194,304	3,145,728	3,145,728
ffn_exp / Block	4,194,304	4,194,304	4,194,304	4,194,304	12,582,912
ffn_red / Block	2,097,152	2,097,152	2,097,152	2,097,152	6,291,456
router / Block	0	4096	4096	4096	4096
input embedding	262,144,000	262,144,000	262,144,000	262,144,000	262,144,000
output embedding	262,144,000	262,144,000	262,144,000	262,144,000	262,144,000
layernorm	1024	1024	1024	1024	1024
Active Params	587,209,728	662,731,776	612,400,128	587,234,304	662731776
Total Params	587,209,728	776,002,560	738,253,824	700,480,512	700,480,512

Table 5. Parameter counting for small scale ablation experiments. Transformer parameters are reported as per parameters per transformer block.

Matching the Number of Total Parameters with BTX To match the number of total parameters in BTX with BAM, we simply increase the number of FFN experts. All newly added FFN experts are initialized from the seed model's FFN. In Table 6, we see that BTX matches BAM's total parameters when BTX uses 6 FFN experts. Even when increasing BTX's total parameter count to 851M, BAM still outperforms BTX.

	Total Params	Active Params	Code	Law	Math	Pre-train
BAM, *ours*	776M	663M	**3.44**	**6.04**	**5.31**	**24.06**
BTX 4 Experts	700M	587M	3.78	6.63	5.77	26.72
BTX 5 Experts	738M	587M	4.11	7.19	6.25	29.08
BTX 6 Experts	776M	587M	3.99	7.03	6.10	28.50
BTX 7 Experts	813M	587M	3.90	6.84	6.00	27.36
BTX 8 Experts	851M	587M	3.94	6.92	6.05	27.79

Table 6. Perplexity ablation (↓) of BAM total parameter matching with BTX on small scale experiments. To match BTX's total parameters to BAM, we increase BTX from having 4 to a maximum of 8 experts. The additional added experts are upcycled from the same copy of the FNN in the dense seed model.

Matching the Number of Total & Active Parameters with BTX To match the number of active parameters with BAM, we simply increase the number of top-k from the usual top-1 to top-3 in every MoE layer of BTX. We see that BTX matches total parameters and active parameters with BAM when using top-3 routing with a total of 6 experts in Table 7. Despite matching parameters, BTX does not outperform BAM.

	Total Params	Active Params	Code	Law	Math	Pre-train
BAM, *ours*	776M	662M	**3.44**	**6.04**	**5.31**	**24.06**
BTX top 3/6 Experts	776M	662M	3.55	6.23	5.48	24.13

Table 7. Perplexity ablation (↓) of BAM total parameter and active parameter matching with BTX on small scale experiments. To match BTX's parameters to BAM, we increase BTX from top-1 to top-3 gating. This means that each token representation activates 3 FFN experts. Same as the previous ablation, 3 of 6 experts are upcycled from the 3 expert dense models while the other 3 are upcycled from the same copy of the dense seed model).