Branch-Train-Merge: Embarrassingly Parallel Training of Expert Language Models

Margaret Li[*†◇] **Suchin Gururangan**[*†◇] **Tim Dettmers**[†]

Mike Lewis[◇] **Tim Althoff**[†] **Noah A. Smith**[†♠] **Luke Zettlemoyer**[†◇]

[†]Paul G. Allen School of Computer Science & Engineering, University of Washington
[♠]Allen Institute for AI
[◇]Meta AI

Abstract

We present Branch-Train-Merge (BTM), a communication-efficient algorithm for training of language models (LMs). BTM learns a set of independent EXPERT LMs (ELMs), each specialized to a different domain, such as scientific or legal text. New ELMs are learned by *branching* from (mixtures of) ELMs in the current set, further *training* on new domains, and then *merging* the resulting models back into the set for future use. These ELMs can be ensembled or averaged at inference time. Experiments show that BTM improves in- and out-of-domain perplexities as compared to compute-matched GPT-style transformer LMs. Our results suggest that extreme parallelism could be used to efficiently scale LMs in future work.

1 Introduction

Training and inference in language models (LMs) typically require access to supercomputers that can achieve the massive multi-node synchronization required to compute model activations and gradients (Brown et al., 2020; Fedus et al., 2022; Zhang et al., 2022). We develop a new class of LMs that is instead embarrassingly parallel: different parts of the model are independently trained on different subsets of the data, with no need for multi-node training or inference (Figure 2).

Our new ELMFOREST[1] model consists of a set of **EXPERT LMs** (ELMs), independently functional LMs specialized to a domain in the training corpus, e.g., scientific or legal text, with *no shared parameters*, which can be ensembled or parameter averaged to collapse back to a single LM at inference time. Our Branch-Train-Merge (BTM) algorithm learns an ELMFOREST by repeatedly adding new ELMs. Each new ELM is first *branched* by initializing a new LM with an average of the parameters of relevant LMs in the current set, then further *trained* on new domains, and finally *merged* into the ELMFOREST (Figure 3). The ELMFOREST is initalized with a single LM, trained on heterogeneous data to establish strong shared representations for future domain specialization.

ELMFORESTs trained with BTM outperform GPT-style transformer LMs and a domain-specialized mixture-of-experts baseline (Gururangan et al. 2022) across a range of computational budgets. We release code and models.[2]

[1]**E**xpert **L**anguage **M**odels **For E**fficient **S**parse **T**raining
[2]URL anonymized for review.

2 Methods

We define an **ELMFOREST** to be a set of EXPERT LMs (ELMs), each independently trained to specialize to a different subset of a corpus. ELMs are inspired by the experts in earlier MoE models (Jacobs et al., 1991), but we define ours to be *domain* specialists and specialize the full LM. Following Gururangan et al. 2022, we define domains by *provenance* (the source of the document, e.g., legal document, medical research paper).[3] ELMs remain independent throughout training and inference.

ELMFORESTs support two inference modes. We perform *output ensembling* over the output probabilities of multiple ELMs. Alternatively, we use *parameter averaging* (Izmailov et al., 2018; Wortsman et al., 2022; Matena & Raffel, 2021) to collapse the ELMFOREST into a single LM, keeping inference cost constant as ELMs are added to the set. We weight both inference operations with a *domain posterior*, which estimates the relevance of each expert to the evaluation domain.[4]

2.1 BRANCH-TRAIN-MERGE (BTM)

BRANCH-TRAIN-MERGE training of ELMFOREST models is incremental and embarrassingly parallel; EXPERT LMs are trained fully independently, starting from a seed LM (Appendix Figures 2 and 3). Each BTM iteration begins with an existing ELMFOREST $E = \{\theta_i\}_{i=1}^k$. Each ELM θ_i represents a corresponding domain d_i in the dataset of k domains $D_E = \{d_i\}_{i=1}^k$ modeled by E. We first describe the inductive case of $k > 0$, then describe how to train the initial model θ_0.

Step 1 (Branch): Given some vector of weights $w = \{w_1, w_2, ..., w_k\}$ over the existing experts $\theta_1, \theta_2, ..., \theta_k$, we initialize the new expert with the weighted parameter average $\theta_{k+1} \leftarrow \sum_{i=0}^k w_i \theta_i$.

Step 2 (Train): We train the new ELM θ_{k+1} on data domain d_{k+1} with the log likelihood objective. None of the existing ELMs in E are involved in the training of the new ELM. We also refer to this step as *branched training* to distinguish it from other training regimens.

Step 3 (Merge): We merge the new ELM θ_{k+1} into E to create an updated set: $E' = E \cup \{\theta_{k+1}\}$.

Step 0 (Initialization): In the first iteration of BTM, $E = \emptyset$; we have no ELMs in the set to branch from. Instead of initializing the first ELMs randomly, we perform a *seed phase*, training a *seed LM* θ_{seed} on some data corpus d_{seed} to initialize the first batch of ELMs.

3 Experiments and Results

3.1 Experimental Setup

Data We use data from Gururangan et al. (2022), which consists of 8 diverse training and 8 evaluation (primarily English-language) domains. Details are in Appendix Table 7.

Model hyperparameters The model architecture is a randomly-initialized LM with the GPT-3 (Brown et al., 2020) architecture implemented in Fairseq (Ott et al., 2019). We use 125M (small), 350M (medium), 750M (large), 1.3B (xl) parameter models. Following Brown et al. 2020, we use the GPT-2 (Radford et al., 2019) vocabulary of 50,264 BPE types, and train with 1,024-token sequences, across document boundaries. We prepend a beginning-of-document token to each document.

Compared Models For our ELMFORESTs, we first conduct a seed phase to initialize the ensemble with LM parameters, then conduct branched training on the ELMs (§2.1), all initialized with the seed LM. Our baselines are (1) a *transformer-LM*, implemented with distributed data parallelism (Li, 2021) [5]; (2) DEMIX (Gururangan et al., 2022), where feedforward layers in the transformer are trained to specialize as domain experts. These models are compute-matched.

[3]See §C.4 for a discussion on the possible limitations of this domain definition.

[4]Details of the method are in Appendix §A.1.

[5]This is identical to the DENSE model from Gururangan et al. (2022) – data from each domain is balanced, which achieves better performance than without data balancing (Appendix Table 9).

125M			
	T-LM 125M	DEMIX 512M	ELMFOREST 1B
Train	$19.9_{0.23}$	$18.2_{0.82}$	$\mathbf{17.2}_{0.02}$
Eval	$25.2_{0.18}$	$23.4_{0.54}$	$\mathbf{22.4}_{0.12}$
All	$22.5_{0.14}$	$20.8_{0.63}$	$\mathbf{19.8}_{0.05}$

350M			
	T-LM 350M	DEMIX 1.8B	ELMFOREST 2.8B
Train	16.3	15.0	**14.7**
Eval	20.8	19.9	**18.6**
All	18.5	17.5	**16.7**

750M			
	T-LM 750M	DEMIX 3.8B	ELMFOREST 6B
Train	14.7	13.5	**13.4**
Eval	19.3	17.7	**16.7**
All	17.0	15.6	**15.0**

1.3B			
	T-LM 1.3B	DEMIX 7B	ELMFOREST 10.4B
Train	14.2	13.7	**13.0**
Eval	18.4	17.6	**16.3**
All	16.3	15.6	**14.6**

Table 1: **ELMFORESTs trained with BTM outperform baselines.** Average test-set perplexity (↓) for each model scale across the 8 training, 8 evaluation, and 16 data domains. Total parameters are shown for each model. At 125M parameters per GPU, we show the mean and standard deviation over 8 random seeds. For BTM, we show results with 50% of compute dedicated to the seed phase.

	Eval Domains PPL (↓)			
	125M	350M	760M	1.3B
TRANSFORMER-LM	**25.2**	20.8	19.3	18.4
ELMFOREST parameter average (uniform weights)	31.0	22.4	20.8	19.5
Argmax ELM (one-hot weights)	28.3	22.3	22.3	20.3
ELMFOREST parameter average (posterior weights)	28.5	**20.3**	**18.0**	**17.0**
ELMFOREST ensemble	22.4	18.6	16.7	16.3

Table 2: **ELMs can be combined through parameter averaging**. Average test-set perplexity across the 8 evaluation domains, comparing techniques to collapse ELMFOREST into a single LM.

Training settings To disentangle variations in GPU speed, we use *number of updates* as our computational budget: 80k, 32k, 24k, and 12k updates on 16, 32, 64, and 128 GPUs in parallel for the 125M, 350M, 750M, 1.3B parameter TRANSFORMER-LM and DEMIX baselines, respectively. We use the same GPU counts for the seed phase in the ELMFOREST. For branched training, we divide these GPUs equally among the ELMs. For all models, we fix the learning rate at 0.0005 with a polynomial (linear) decay learning rate schedule and 8% warmup, which we found to be optimal for most settings after a large grid search. We use a batch size of 16 for each GPU, with gradient accumulation of 32 steps, and train with `fp16`. We train on NVIDIA V100 32GB GPUs.

3.2 Results

Ensembling results are shown in Table 1. At these model scales, ELMFOREST ensembles outperform both the sparsely trained DEMIX LM and the densely trained TRANSFORMER-LM baselines.

While ELMFOREST substantially improves performance at lower training cost relative to the TRANSFORMER-LM, it comes at the price of a larger model size and higher associated inference costs when ensembling. Thus, we explore parameter averaging to combine experts for improved generalization with no additional inference costs relative to the TRANSFORMER-LM baseline. Given some weight vector w over k ELMs $\{\theta_i, ..., \theta_k\}$, we define a single model such that its parameters are a weighted average of the ELM parameters, according to w: $\theta = \sum_{i=0}^{k} w_i \theta_i$. For w, we consider:

Uniform: We set w to be a uniform; i.e., $\frac{1}{k}$. This setting disregards the relevance of each ELM to the target domain, assuming all ELMs should contribute equally to the average.

Argmax: We set w to be an indicator vector corresponding to the maximum probability in the domain posterior, thus activating only the estimated best-performing ELM.

Posterior: We set w to be the domain posterior, computed on the validation set.

Results on the evaluation domains are in Table 2.[6] Using uniform weights underperforms all baselines, even TRANSFORMER-LMs, highlighting the importance of domain relevance in output ensembling and parameter averaging ELMs. Using the argmax ELM outperforms uniform averaging for small models, but not larger models. Weighting the average with the domain posterior outperforms all other techniques, and consistently improves performance over TRANSFORMER-LMs at no additional inference cost. Though parameter averaging does not reach the performance of output ensembling, the lower inference costs and simplicity of deployment may make averaging the preferred inference technique for resource-constrained applications.

In Figure 1, we fix the parameter average to use the domain posterior weights and vary the portion of the compute budget dedicated to the seed phase and observe the effects on performance. We observe that parameter averaging performance on training domains is relatively robust to seed training. On evaluation domains, however, the smallest scale ELMFOREST does not achieve optimal performance until about 60% or more updates are dedicated to seed training. This explains the poor performance of the 125M parameter scale ELMFOREST average on evaluation domains in Table 2. Overall, results suggest a strong effect of the seed phase on the viability of ELMFOREST averaging. ELMFOREST averaging does not work with ELMs trained from random initialization (i.e., with no seed phase).

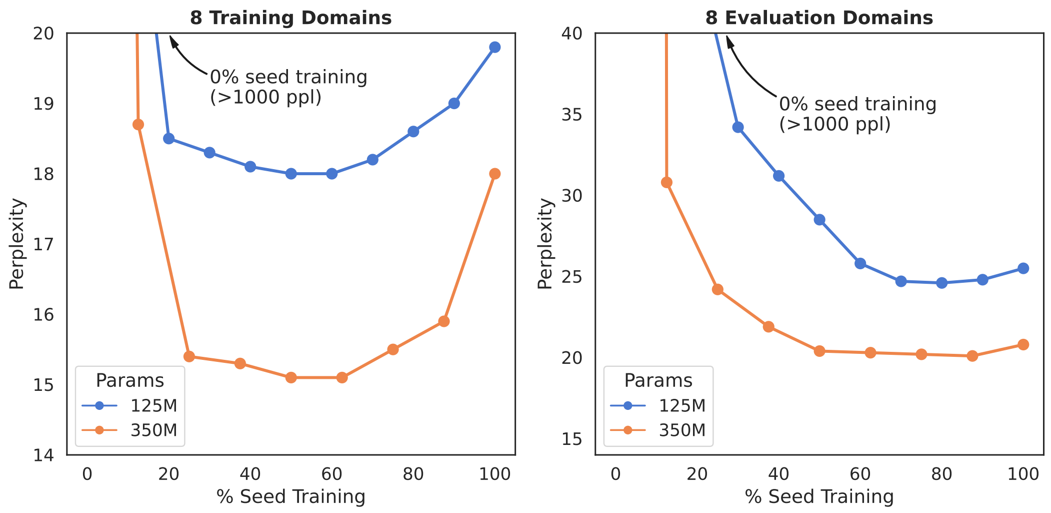

Figure 1: **The seed phase is vital to our ability to parameter average ELMs.** Test perplexity averaged across the 8 training (left) and 8 evaluation (right) domains when averaging ELMFOREST with different seed training compute allocations for the 125M and 350M parameter LMs.

4 Related Work

Sparsely activated language models have been considered in a few forms (Evci et al., 2020; Mostafa & Wang, 2019; Dettmers & Zettlemoyer, 2019), but most related to this work is the Mixture-of-Experts (MoE) model (Jacobs et al., 1991; Lepikhin et al., 2021; Fedus et al., 2022; Lewis et al., 2021; Roller et al., 2021). Of this line of work, ours is most closely related to DEMix layers Gururangan et al. (2022), which replace transformer feedforward layers as domain experts.

Ensemble methods are classic techniques in machine learning (Breiman, 1996; Freund, 1995; Wolpert, 1992). Similar to our work, recent work has considered growing ensembles, in which new models are trained sequentially on streaming data Caccia et al. (2021).

Our averaging mechanism is inspired by the model merging techniques in the vision and NLP literature (Wortsman et al., 2022; Izmailov et al., 2018; Matena & Raffel, 2021). Our posterior weighted average is highly related to Bayesian model averaging techniques used in classic ensembling methods (Fragoso et al., 2018). Model averaging has also been explored for federated learning (McMahan et al., 2017).

[6]We display similar findings with training domains in Appendix Table 8.

5 Conclusion

We introduce BTM, a new algorithm to train an ELMFOREST, which contains many EXPERT LMs. Our experiments show that ELMFORESTs trained with BTM outperform compute-matched baselines, when conducting inference through output ensembling or parameter averaging. These results provide compelling evidence for the promise of scaling LMs with many smaller, independently trained ELMs.

References

Roee Aharoni and Yoav Goldberg. Unsupervised domain clusters in pretrained language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 7747–7763, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.692. URL `https://www.aclweb.org/anthology/2020.acl-main.692`.

Mikel Artetxe, Shruti Bhosale, Naman Goyal, Todor Mihaylov, Myle Ott, Sam Shleifer, Xi Victoria Lin, Jingfei Du, Srinivasan Iyer, Ramakanth Pasunuru, Giri Anantharaman, Xian Li, Shuohui Chen, Halil Akin, Mandeep Baines, Louis Martin, Xing Zhou, Punit Singh Koura, Brian O'Horo, Jeff Wang, Luke Zettlemoyer, Mona Diab, Zornitsa Kozareva, and Ves Stoyanov. Efficient large scale language modeling with mixtures of experts, 2021. URL `https://arxiv.org/abs/2112.10684`.

Jason Baumgartner, Savvas Zannettou, Brian Keegan, Megan Squire, and Jeremy Blackburn. The pushshift reddit dataset, 2020.

Douglas Biber. *Variation across Speech and Writing*. Cambridge University Press, 1988. doi: 10.1017/CBO9780511621024.

John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pp. 120–128, Sydney, Australia, July 2006. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/W06-1615`.

Su Lin Blodgett, Lisa Green, and Brendan O'Connor. Demographic dialectal variation in social media: A case study of African-American English. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1119–1130, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1120. URL `https://www.aclweb.org/anthology/D16-1120`.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL `https://arxiv.org/abs/2005.14165`.

Lucas Caccia, Jing Xu, Myle Ott, Marc'Aurelio Ranzato, and Ludovic Denoyer. On anytime learning at macroscale. *CoRR*, abs/2106.09563, 2021. URL `https://arxiv.org/abs/2106.09563`.

Caselaw Access Project. Caselaw access project. URL `https://case.law/`.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling, 2014.

Alexandra Chronopoulou, Matthew Peters, and Jesse Dodge. Efficient hierarchical domain adaptation for pretrained language models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1336–1351, Seattle, United States, July 2022. Association for Computational Linguistics. URL `https://aclanthology.org/2022.naacl-main.96`.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4599–4610, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.365. URL `https://aclanthology.org/2021.naacl-main.365`.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *CoRR*, abs/1907.04840, 2019. URL `http://arxiv.org/abs/1907.04840`.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 2943–2952. PMLR, 13–18 Jul 2020. URL `https://proceedings.mlr.press/v119/evci20a.html`.

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. URL `http://jmlr.org/papers/v23/21-0998.html`.

Tiago Fragoso, Wesley Bertoli, and Francisco Louzada. Bayesian model averaging: A systematic review and conceptual classification. *International Statistical Review*, 86:1–28, 04 2018. doi: 10.1111/insr.12243.

Yoav Freund. Boosting a weak learning algorithm by majority. *Information and computation*, 121(2): 256–285, 1995.

Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. RealToxicityPrompts: Evaluating neural toxic degeneration in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 3356–3369, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.301. URL `https://aclanthology.org/2020.findings-emnlp.301`.

Github Archive Project. Github archive project. URL `https://cloud.google.com/blog/topics/public-datasets/github-on-bigquery-analyze-all-the-open-source-code`.

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus, 2019. URL `http://Skylion007.github.io/OpenWebTextCorpus`.

Raphael Gontijo-Lopes, Yann Dauphin, and Ekin Dogus Cubuk. No one representation to rule them all: Overlapping features of training methods. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=BK-4qbGgIE3`.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah 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*, pp. 8342–8360, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.740. URL `https://www.aclweb.org/anthology/2020.acl-main.740`.

Suchin Gururangan, Mike Lewis, Ari Holtzman, Noah A. Smith, and Luke Zettlemoyer. DEMix layers: Disentangling domains for modular language modeling. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5557–5576, Seattle, United States, July 2022. Association for Computational Linguistics. URL `https://aclanthology.org/2022.naacl-main.407`.

Dan Hendrycks, Collin Burns, Anya Chen, and Spencer Ball. Cuad: An expert-annotated nlp dataset for legal contract review, 2021.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization, 2018. URL `https://arxiv.org/abs/1803.05407`.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Angeliki Lazaridou, Adhiguna Kuncoro, Elena Gribovskaya, Devang Agrawal, Adam Liska, Tayfun Terzi, Mai Gimenez, Cyprien de Masson d'Autume, Tomáš Kočiský, Sebastian Ruder, Dani Yogatama, Kris Cao, Susannah Young, and Phil Blunsom. Mind the gap: Assessing temporal generalization in neural language models. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=73OmmrCfSyy`.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. {GS}hard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=qrwe7XHTmYb`.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. Base layers: Simplifying training of large, sparse models. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 6265–6274. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/lewis21a.html`.

Shen Li. Getting started with distributed data parallel, 2021. URL `https://pytorch.org/tutorials/intermediate/ddp_tutorial.html`.

Kyle Lo, Lucy Lu Wang, Mark Neumann, Rodney Kinney, and Daniel Weld. S2ORC: The semantic scholar open research corpus. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4969–4983, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.447. URL `https://www.aclweb.org/anthology/2020.acl-main.447`.

Li Lucy and David Bamman. Characterizing English variation across social media communities with BERT. *Transactions of the Association for Computational Linguistics*, 9:538–556, 2021. doi: 10.1162/tacl_a_00383. URL `https://aclanthology.org/2021.tacl-1.33`.

Kelvin Luu, Daniel Khashabi, Suchin Gururangan, Karishma Mandyam, and Noah A. Smith. Time waits for no one! analysis and challenges of temporal misalignment. 2021. doi: 10.48550/ARXIV.2111.07408. URL `https://arxiv.org/abs/2111.07408`.

Michael Matena and Colin Raffel. Merging models with fisher-weighted averaging. *arXiv preprint arXiv:2111.09832*, 2021.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR, 2017.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 4646–4655. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/mostafa19a.html`.

Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 188–197, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1018. URL `https://www.aclweb.org/anthology/D19-1018`.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pp. 48–53, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-4009. URL `https://www.aclweb.org/anthology/N19-4009`.

Ou-Yang, Lucas. Newspaper3k. URL `https://newspaper.readthedocs.io/en/latest/`.

Project Gutenberg. Project gutenberg. URL `www.gutenberg.org`.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. URL `https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf`.

John R. Rickford. Ethnicity as a sociolinguistic boundary. *American Speech*, 60:99, 1985.

Stephen Roller, Sainbayar Sukhbaatar, Arthur Szlam, and Jason E Weston. Hash layers for large sparse models. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=lMgDDWb1ULW`.

Lucy Lu Wang, Kyle Lo, Yoganand Chandrasekhar, Russell Reas, Jiangjiang Yang, Doug Burdick, Darrin Eide, Kathryn Funk, Yannis Katsis, Rodney Kinney, Yunyao Li, Ziyang Liu, William Merrill, Paul Mooney, Dewey Murdick, Devvret Rishi, Jerry Sheehan, Zhihong Shen, Brandon Stilson, Alex Wade, Kuansan Wang, Nancy Xin Ru Wang, Chris Wilhelm, Boya Xie, Douglas Raymond, Daniel S. Weld, Oren Etzioni, and Sebastian Kohlmeier. Cord-19: The covid-19 open research dataset, 2020.

David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time, 2022. URL `https://arxiv.org/abs/2203.05482`.

Jing Xu, Da Ju, Margaret Li, Y-Lan Boureau, Jason Weston, and Emily Dinan. Bot-adversarial dialogue for safe conversational agents. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2950–2968, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.235. URL `https://aclanthology.org/2021.naacl-main.235`.

Yelp Reviews. Yelp reviews. URL `https://www.yelp.com/dataset`.

Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. Defending against neural fake news. In *NeurIPS*, 2019. URL `http://papers.nips.cc/paper/9106-defending-against-neural-fake-news.pdf`.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022. URL `https://arxiv.org/abs/2205.01068`.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Anonymized for review
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]

   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] When possible given our compute budget
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] In the appendices and the README of our code repo (currently anonymized)
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Anonymized
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] Briefly – we describe our deidentification schema in the appendices

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

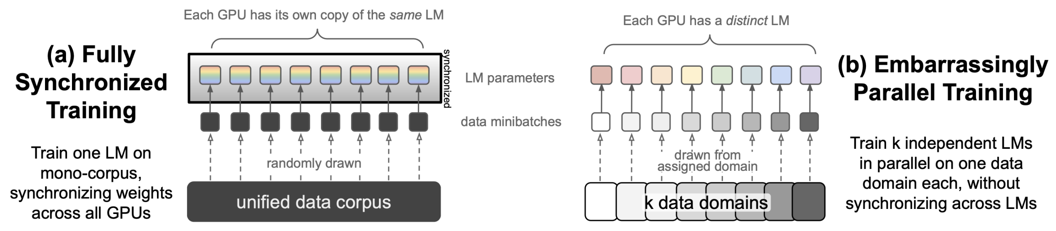
(a) Fully Synchronized Training
Train one LM on mono-corpus, synchronizing weights across all GPUs
Each GPU has its own copy of the *same* LM
synchronized
LM parameters
data minibatches
randomly drawn
unified data corpus
Each GPU has a *distinct* LM
drawn from assigned domain
k data domains
(b) Embarrassingly Parallel Training
Train k independent LMs in parallel on one data domain each, without synchronizing across LMs

Figure 2: **Fully Synchronized vs. Embarrassingly Parallel Training (§2.1).** (a) In fully synchronized data-parallel training of a TRANSFORMER-LM, all parameters are synchronized across all GPUs. This synchronization incurs hefty cross-node communication costs. (b) In embarrassingly parallel training (our work), individual models are trained on each domain, eliminating expensive cross-node parameter synchronization between those models.

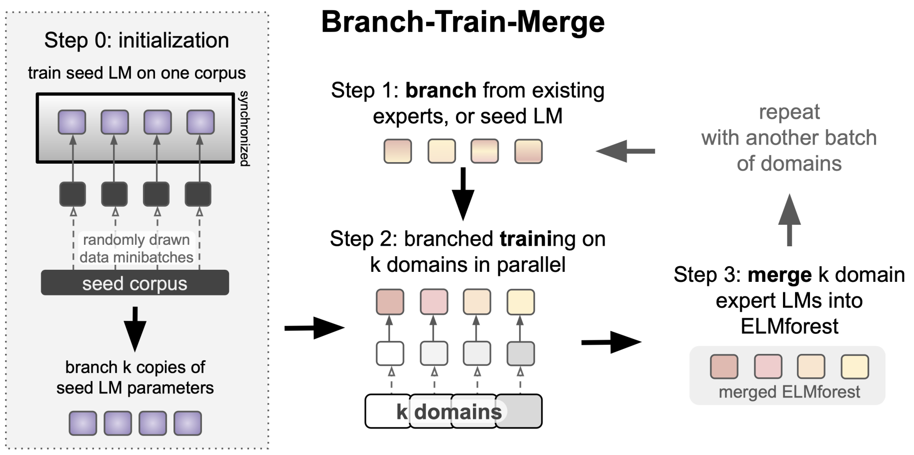
Branch-Train-Merge
Step 0: initialization
train seed LM on one corpus
synchronized
randomly drawn data minibatches
seed corpus
branch k copies of seed LM parameters
Step 1: **branch** from existing experts, or seed LM
Step 2: branched **training** on k domains in parallel
k domains
repeat with another batch of domains
Step 3: **merge** k domain expert LMs into ELMforest
merged ELMforest

Figure 3: **BTM training process overview (§2.1).** In the seed phase (Step 0), an LM is trained on one data corpus. We branch, or copy, the parameters k times (Step 1), and continue to train each copy on a unique data domain, resulting in k ELMs (Step 2), which are merged into the ELMFOREST (Step 3). After the seed phase, ELMs are fully disconnected, with no communication between them.

A Additional method details

A.1 Finding the *domain posterior* for inference

Consider the probabilistic view of language modeling, where we estimate $p(X_t \mid \boldsymbol{x}_{<t})$. We introduce a domain variable D, alongside each sequence. Then the next-step conditional distribution on the history $\boldsymbol{x}_{<t}$ is:

$$p(X_t \mid \boldsymbol{x}_{<t}) = \sum_{j=1}^{n} p(X_t \mid \boldsymbol{x}_{<t}, D = j) \cdot p(D = j \mid \boldsymbol{x}_{<t}) \tag{1}$$

We estimate a **domain posterior**, or a probability of a sequence belonging to each of the k domains using Bayes' rule:

$$p(D = j \mid \boldsymbol{x}_{<t}) = \frac{p(\boldsymbol{x}_{<t} \mid D = j) \cdot p(D = j)}{p(\boldsymbol{x}_{<t})} = \frac{p(\boldsymbol{x}_{<t} \mid D = j) \cdot p(D = j)}{\sum_{j'=1}^{k} p(\boldsymbol{x}_{<t} \mid D = j') \cdot p(D = j')} \tag{2}$$

ELMs are used to compute the likelihood over contexts given a domain label. To compute the cached prior, we maintain an exponential moving average of posterior probabilities over domains, updated only at the end of each sequence block: $p(D = j) = \sum_{i=1}^{N} \lambda^i \cdot p(D = j \mid x_{<T}^{(i)})$. Following Gururangan et al. 2022, we use $N = 100$ sequences (of length $T = 1024$ each) of development data, and set EMA decay $\lambda = 0.3$. We fix this prior at test time for each domain.

	Average updates per second, normalized (↑)		
	fully synchronized (TRANSFORMER-LM)	partially synchronized (DEMIX)	BTM: embarrassingly parallel (branched ELMs)
125M	1.00	1.01	1.05
350M	1.00	1.11	1.23
750M	1.00	1.01	1.27
1.3B	1.00	0.97	1.33

Table 3: **BTM is more efficient (§B).** Average updates per second (↑) for each setup and model size, normalized by the average updates per second during fully synchronized training of the TRANSFORMER-LM. The efficiency gains from embarrassingly parallel training (the branched phase of BTM) become more substantial with larger model size – and more nodes used in parallel.

750M			
	Random Ensemble (seed init)	ELM FOREST (random init)	ELM FOREST (seed init)
Train	17.4	14.4	**13.4**
Eval	20.9	19.3	**16.7**
All	19.2	16.9	**15.0**

1.3B			
	Random Ensemble (seed init)	ELM FOREST (random init)	ELM FOREST (seed init)
Train	17.4	13.3	**13.0**
Eval	20.4	17.8	**16.3**
All	18.9	15.6	**14.6**

Table 4: **Domain expert ensemble outperforms random split ensemble (§C.1).** Average test-set perplexity (↓) for our largest model scales across the 8 training, 8 evaluation, and all 16 domains. We show similar results for the 125M and 350M parameter scale models in Appendix Figure 10.

Output ensembling naively requires a forward pass through all ELMs in the ELMFOREST, but we observe in practice that the domain posterior is sparse, which suggests that top-k selection of EXPERT LMs can reduce inference time costs with negligible effects on performance.

B Efficiency Comparison Results

Training ELMFORESTs requires less inter-GPU communication than TRANSFORMER-LM or DEMIX models, since no synchronization occurs between GPUs assigned to different ELMs. This results in higher updates per second and therefore shorter train times (Table 3). Additionally, the embarrassingly parallel branched training provides flexibility in resource consumption; GPUs dedicated to different ELMs may be online at different times, and ELMs may even be trained serially on the same GPUs. Specifically, none of our branched training required more than 16 GPUs simultaneously, while our TRANSFORMER-LM training experiments consumed 128 GPUs simultaneously. Empirically, ELMFOREST training jobs were scheduled and run more quickly, and with less preemption, than the TRANSFORMER-LM and DEMIX training jobs at the same overall budget.

C Analysis

In §3, we largely fix the training setup to conduct a controlled comparison of BTM to baseline methods. We now analyze the importance of various training and inference decisions on language modeling performance.

C.1 ELMFOREST outperforms parameter-matched ensembles

We compare our method to other LM ensembles to study the effect of increased parameter counts:

Random Ensemble (seed init) A set of LMs trained on random data splits, to assess the importance of ELM domain specialization. We pool the training and development sets of the 8 train domains, divide into 8 random splits, then conduct BTM on those splits, with 50% seed training.

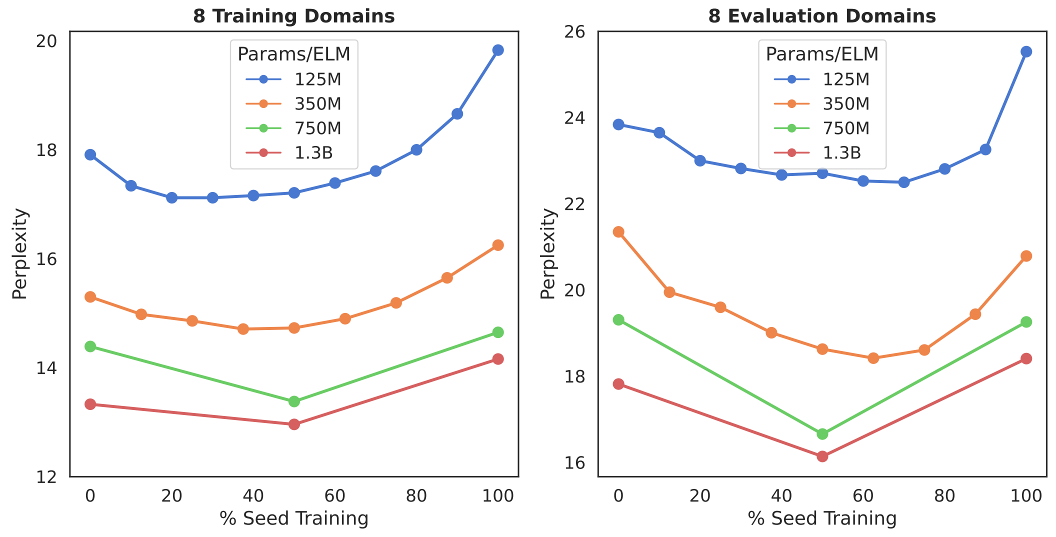

Figure 4: **ELMFOREST ensembling performance is robust to most seed training compute allocations (§C.2).** Test perplexity averaged across the 8 training (left) or 8 evaluation (right) domains (from §3.1) when fixing total compute budget but varying the portion allocated to seed training.

ELMFOREST (random init) An ELMFOREST trained with BTM where all ELMs are randomly initialized, to assess the effect of seed training. This is equivalent to setting the seed training compute budget to zero updates. We fix the random initialization across models.

ELMFOREST (seed init) The ELMFOREST setting of §3. We conduct BTM on the 8 train domains, and dedicate 50% of the updates in the budget to seed and to branched ELM training.

Results with the largest models are in Table 4.[7] ELMFOREST (random init) nearly matches ELMFOREST on training domains but performs poorly on evaluation domains. The random ensemble is consistently worse than both variants of ELMFOREST, showing that the performance improvement is not only due to ensembling or increased total model size.[8]

C.2 ELMFOREST performance is robust to seed LM training compute allocation

The ELMFOREST (random init), which has no seed training, underperforms ELMFOREST (LM init) in §C.1, indicating that seed training is essential. On the other hand, TRANSFORMER-LM, equivalent to 100% seed training, also underperforms ELMFOREST (LM init) in §3, which suggests the importance of branched ELM training. We now study the changes to performance when we vary the portion of the compute budget dedicated to seed training. We control for the total compute budget (across seed and branched training).

Our results, in Figure 4, show that the optimal amount of seed training is about 40–60% of the total budget. At both ends of the full range, performance deteriorates, approaching the ELMFOREST (random init) and TRANSFORMER-LM performance (at 0% and 100% seed training, respectively).

However, as little as 10% of seed training can be performed to result in strong gains over the ELMFOREST (random init) and TRANSFORMER-LM. This suggests that the majority of BTM training may focus on branched training to dramatically reduced computational costs (§B). The optimal share of compute to use towards each training phase likely depends on many factors, including the total compute budget. We leave more thorough study of this trend to future work.

[7]We display similar findings with smaller models in Appendix Table 10.

[8]We speculate that the random ensemble is poor because its constituent models make correlated errors during evaluation (Gontijo-Lopes et al., 2022).

		Average Test PPL (↓)		
		Train	Evaluation	Overall
	TRANSFORMER-LM	19.8	25.5	22.7
seed corpus	**8 train domains**	**17.2**	**22.7**	**20.0**
	Wikipedia	17.7	23.2	20.5
	C4	17.9	23.5	20.7
	StackOverflow	18.4	24.6	21.5
	JavaScript	19.2	24.9	22.0

Table 5: **ELMFOREST ensembling performance is robust to seed training corpus (§C.3).** Test set perplexity averages on the 8 training, 8 evaluation, and all 16 data domains, using different training corpora used in seed LM training. All models are of the 125M parameters per GPU scale.

C.3 ELMFOREST performance is robust to the choice of seed training corpus

We compare the effects of using different training corpora for seed training in BTM. Here, we fix the compute budget allocations studied in §C.2 so that 50% of updates are allocated to seed training and 50% to branched training. As seen in Table 5, our experiments using the most diverse corpora for seed training resulted in the best performance, but even seed training on only JavaScript code yielded better results than the compute-matched TRANSFORMER-LM baseline, and far better than the ELMFOREST (random init) models in Table 1, which use identical random initialization. This suggests that initializing ELMs with parameters of any model checkpoint is critical.

C.4 Limitations

The definition of a domain The nature of domains in NLP is a matter of active research. Textual domains reflect language variation that stems from factors such as vocabulary differences (Blitzer et al., 2006), sociolinguistic (Biber, 1988) or demographic (Rickford, 1985; Blodgett et al., 2016) variables, community membership (Lucy & Bamman, 2021), end-tasks (Gururangan et al., 2020), or temporal shifts (Lazaridou et al., 2021; Luu et al., 2021). In this work, we follow Gururangan et al. (2022) and define domains by *provenance*, or the source of the document. Provenance labels yield simple and interpretable segmentations of a corpus, which are useful for identifying ELMs in our experiments. However, other methods for discovering domains, including unsupervised techniques (Aharoni & Goldberg, 2020; Chronopoulou et al., 2022), may yield better expert assignments. We leave experimentation with other definitions of domain for future work.

Domain posterior data requirement To calculate the domain posteriors used for our ensembling and parameter averaging weights, we assume access to a small additional sample of data to train the vector w. While it is easy to imagine that extra data may be available for most applications to estimate the posterior, future work may explore the possibility of eliminating this requirement.

Other distributed training baselines Our TRANSFORMER-LM baseline is implemented with distributed data-parallel. Model-parallel, fully sharded data-parallel, and other distributed training strategies (Artetxe et al., 2021) confer different scaling patterns that may change the conclusions that we report in this work. However, we expect that BTM will provide strong efficiency gains against these alternatives.

Harms of language models BTM results in an LM whose test time behaviors can be controlled with much stronger guarantees after training due to the isolation of domains in ELMs. However, ELMFORESTs exposed to large datasets scraped from the Internet may contain toxic language (e.g., hatespeech) that are difficult to identify with coarse provenance domain labels, and nevertheless result in harmful output from the ELMs (Gehman et al., 2020). Future work may explore recipes for training and deploying ELMFORESTs to better support user safety.

Category	**Link to Regex**	Dummy Token
Email	`https://regex101.com/r/ZqsF9x/1`	`<EMAIL>`
DART	`https://regex101.com/r/0tQ6EN/1`	`<DART>`
FB User ID	`https://regex101.com/r/GZl5EZ/1`	`<FB_USERID>`
Phone Number	`https://regex101.com/r/YrDpPD/1`	`<PHONE_NUMBER>`
Credit Card Number	`https://regex101.com/r/9NTO6W/1`	`<CREDIT_CARD_NUMBER>`
Social Security Number	`https://regex101.com/r/V5GPNL/1`	`<SSN>`
User handles	`https://regex101.com/r/vpey04/1`	`<USER>`

Table 6: De-identification schema. We de-identify text using the regexes provided in the above links for the categories listed.

	Domain	**Corpus**	**# Train (Eval.) Tokens**
TRAINING	1B	30M NewsWire sentences (Chelba et al., 2014)	700M (10M)
	CS	1.89M full-text CS papers from S2ORC (Lo et al., 2020)	4.5B (10M)
	LEGAL	2.22M U.S. court opinions (Caselaw Access Project)	10.5B (10M)
	MED	3.2M full-text medical papers from S2ORC (Lo et al., 2020)	9.5B (10M)
	WEBTEXT†	8M Web documents (Gokaslan & Cohen, 2019)	6.5B (10M)
	REALNEWS†	35M articles from REALNEWS Zellers et al. (2019)	15B (10M)
	REDDIT	Reddit comments from pushshift.io (Baumgartner et al., 2020)	25B (10M)
	REVIEWS†	30M Amazon product reviews (Ni et al., 2019)	2.1B (10M)
		Total	**73.8B (80M)**
	Domain	**Corpus**	**# Train (Eval.) Tokens**
EVALUATION	ACL PAPERS	1.5K NLP papers from ACL (Dasigi et al., 2021)	1M (1M)
	BREAKING NEWS†	20K English news articles, scraped using (Ou-Yang, Lucas)	11M (1M)
	CONTRACTS†	500 commercial legal contracts (Hendrycks et al., 2021)	1.5M (1M)
	CORD-19	400K COVID-19 research papers (Wang et al., 2020)	60M (10M)
	GITHUB	230K public Github code (Github Archive Project)	200M (10M)
	GUTENBERG	3.2M copyright-expired books (Project Gutenberg)	3B (10M)
	TWEETS†	1M English tweets from 2013-2018	8M (1M)
	YELP REVIEWS†	6M Yelp restaurant reviews (Yelp Reviews)	600M (10M)

Table 7: **Multi-domain data corpus used in §3 and §C**. Details of this corpus, both training and evaluation domains, including the size of our training and evaluation (i.e. validation and test) data in whitespace-separated tokens. We borrow these datasets from Gururangan et al. (2022). † indicates datasets we de-identify with regexes in Table 6. REDDIT was de-identified by Xu et al. (2021); we use their version. Meta researchers did not collect any of the Reddit or Twitter data and the data was not collected on behalf of Meta.

	Train Domains PPL (↓)			
	125M	350M	760M	1.3B
TRANSFORMER-LM	19.9	16.3	14.7	14.2
ELMFOREST parameter average (uniform weights)	47.4	19.9	19.0	18.0
Argmax ELM (one-hot posterior)	**18.0**	15.3	14.1	13.8
ELMFOREST parameter average (posterior weights)	**18.0**	**15.1**	**13.9**	**13.4**
ELMFOREST ensemble	17.2	14.7	13.4	13.0

Table 8: **Performance of ELM parameter averaging on training domains (§3.2)**. Average test-set perplexity across the 8 training domains, from the models in Table 1, comparing techniques to collapse ELMFOREST into a single LM. As with evaluation domain results in the main paper, parameter averaging (with posterior weights) generally yields better average perplexities than TRANSFORMER-LM at no additional inference cost, but underperforms ELMFOREST ensembling, which uses more effective parameters and is included for comparison as a lower bound.

125M – 16 GPUs – 80k updates

	T-LM 125M	T-LM UNBALANCED 125M	DEMIX 512M	ELMFOREST (random init) 1B	RANDOM ENSEMBLE 1B	ELMFOREST (seed init) 1B
Train	19.8	20.7	17.7	18.0	23.0	**17.2**
Novel	25.6	26.4	23.1	24.1	26.0	**22.4**
All	22.7	23.5	20.4	21.0	24.7	**19.8**

350M – 32 GPUs – 32k updates

	T-LM 350M	T-LM UNBALANCED 350M	DEMIX 1.8B	ELMFOREST (random init) 2.8B	RANDOM ENSEMBLE 2.8B	ELMFOREST (seed init) 2.8B
Train	16.3	16.7	15.0	15.3	19.9	**14.7**
Novel	20.8	21.2	19.9	21.3	23.1	**18.6**
All	18.5	19.0	17.5	18.3	21.5	**16.7**

750M – 64 GPUs – 24k updates

	T-LM 750M	T-LM UNBALANCED 750M	DEMIX 3.8B	ELMFOREST (random init) 6B	RANDOM ENSEMBLE 6B	ELMFOREST (seed init) 6B
Train	14.7	14.9	13.5	14.4	17.4	**13.4**
Novel	19.3	19.8	17.7	19.3	20.9	**16.7**
All	17.0	17.4	15.6	16.9	19.2	**15.0**

1.3B – 128 GPUs – 12k updates

	T-LM 1.3B	T-LM UNBALANCED 1.3B	DEMIX 7B	ELMFOREST (random init) 10.4B	RANDOM ENSEMBLE 10.4B	ELMFOREST (seed init) 10.4B
Train	14.2	15.0	13.7	13.3	17.4	**13.0**
Novel	18.4	19.5	17.6	17.8	20.4	**16.3**
All	16.3	17.3	15.6	15.6	18.9	**14.6**

Table 9: **ELMFORESTs trained with BTM outperform all baselines and ensemble variations across multiple model scales**. Average test-set perplexity (↓) for each model scale (125M, 350M, 750M, 1.3B parameters) across the 8 training, 8 novel, and all 16 domains described in §3.1. Total compute budget (in update numbers) and GPU usage are shown for each model size, and total parameters are shown for each model type at each size. TRANSFORMER-LMs (here, abbreviated to T-LM) trained without balancing between data domains performs worse than T-LM trained with data balancing; hence, we only compare against the balanced T-LM setting in §3. For ELMFOREST, we show results with 50% dense training.

125M

	Random Ensemble (seed init)	ELM FOREST (random init)	ELM FOREST (seed init)
Train	23.0	18.2	**17.2**
Eval	26.0	23.4	**22.4**
All	24.7	20.8	**19.8**

350M

	Random Ensemble (seed init)	ELM FOREST (random init)	ELM FOREST (seed init)
Train	19.9	15.3	**14.7**
Eval	23.1	21.3	**18.6**
All	21.5	18.3	**16.7**

Table 10: **Domain expert ensemble outperforms random split ensemble (§C.1).** Average test-set perplexity (↓) for our smallest model scales across the 8 training, 8 evaluation, and all 16 domains.

		w/ ELM (Average Test PPL ↓)	**–ELM (△ PPL)**
	8 train domains	17.2	(+9.4)
seed corpus	**Wikipedia**	17.7	(+11.9)
	C4	17.9	(+11.8)
	StackOverflow	18.4	(+12.7)
	JavaScript	19.2	(+13.6)

Table 11: **The ability to reduce the influence of domains through ELM removal is (mostly) robust to seed training corpus (§C.3).** We present the average test perplexity for the 8 train domains in ELMFORESTs where all ELMs are active. We vary the seed training corpora. In parentheses, we show the *increase* in perplexity when the ELM trained to specialize on each domain is removed at inference time. Large increases are desired and suggest the ease of removing (e.g., stale or harmful) data from the ELMFOREST's distribution after training.