

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 embarrassingly parallel training of large language models (LLMs). We show it is possible to independently train subparts of a new class of LLMs on different subsets of the data, eliminating the massive multi-node synchronization currently required to train LLMs. BTM learns a set of independent EXPERT LMs (ELMs), each specialized to a different textual domain, such as scientific or legal text. These ELMs can be added and removed to update data coverage, ensembled to generalize to new domains, or averaged to collapse back to a single LM for efficient inference. New ELMs are learned by *branching* from (mixtures of) ELMs in the current set, further *training* the parameters on data for the new domain, and then *merging* the resulting model back into the set for future use. Experiments show that BTM improves in- and out-of-domain perplexities as compared to GPT-style Transformer LMs, when controlling for training cost. Through extensive analysis, we show that these results are robust to different ELM initialization schemes, but require expert domain specialization; LM ensembles with random data splits do not perform well. We also present a study of scaling BTM into a new corpus of 64 domains (192B whitespace-separated tokens in total); the resulting LM (22.4B total parameters) performs as well as a Transformer LM trained with $2.5\times$ more compute. These gains grow with the number of domains, suggesting more aggressive parallelism could be used to efficiently train larger models in future work.

1 Introduction

Training and inference in large language models (LMs) typically require access to supercomputers that can achieve the massive multi-node (e.g., GPU or TPU) synchronization required to compute model activations and gradients (Brown et al., 2020; Zhang et al., 2022; Fedus et al., 2022; Lepikhin et al., 2021). In this paper, we develop a new class of large 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 ELMFOREST¹ models consist of a set of **EXPERT LMs** (ELMs), each specialized to a distinct domain in the training corpus, e.g., scientific or legal text. The ELMs are each independently functional LMs with *no shared parameters*, unlike previous domain mixture-of-experts models that

^{*}These authors contributed equally to this work. Correspondence to {margsli, sg01}@cs.washington.edu.

¹Expert Language Models For Efficient Sparse Training

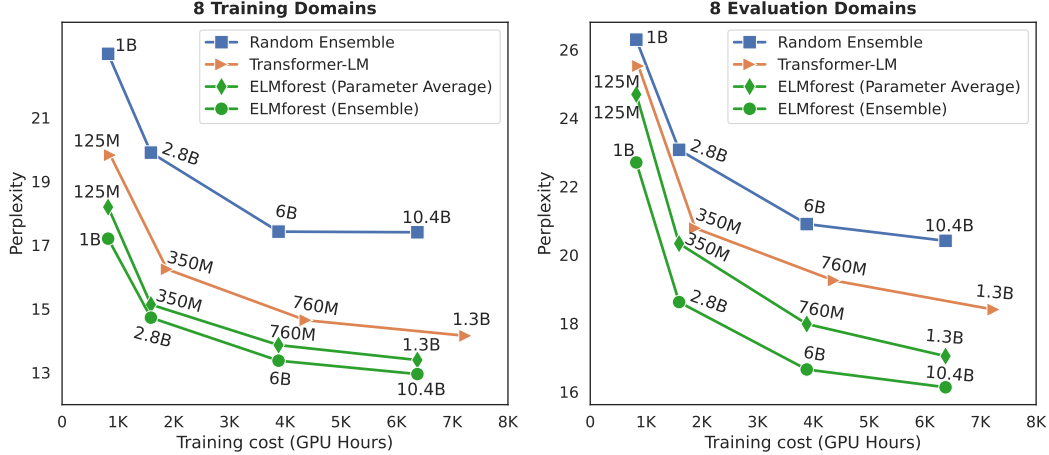


Figure 1: **ELMFORESTS outperform compute-matched TRANSFORMER-LM and random-ensemble baselines across parameter scales (§4.2).** ELMFOREST parameter averaging closely approaches the performance of ELMFOREST ensembling at no additional cost to inference compared to the TRANSFORMER-LM. ELMFOREST efficiency also grows as the model size increases due to substantially reduced GPU communication overhead relative to TRANSFORMER-LM baselines.

only specialize the Transformer feedforward layers (Gururangan et al., 2022). ELMs can be added and removed to the model at any time to update data coverage, ensembled to generalize to new domains, or parameter averaged to collapse back to a single LM for more efficient inference.

We present the Branch-Train-Merge (BTM) algorithm for learning this set of specialized LMs. Our procedure repeatedly expands the ELMFOREST by adding one or more new ELMs completely in parallel. Each new ELM in the ELMFOREST is first *branched* by initializing a new LM with an average of the parameters of the most relevant LMs in the current set, then further *trained* on new domain data with a standard cross-entropy loss, and finally *merged* into the model by simply adding it to the current ELMFOREST (Figure 3 provides a schematic of this process). BTM is initialized with a single LM that is trained on heterogeneous data to establish strong shared representations for future domain specialization, a process that we explore extensively in our ablation analysis.

When evaluated in- and out-of-domain, ELMFORESTS trained with BTM outperform monolithic GPT-style transformer LMs (GPT-LMs) and a previous domain-specialized mixture-of-experts baseline (DEMIX; Gururangan et al. 2022) across a range of computational budgets – up to 1.3B parameters per ELM trained for 7000 GPU-hours in aggregate (Figure 1; §4.2). These gains are biggest for ELMFOREST ensembles, which use all of the model parameters, but also hold when we collapse the models by averaging parameters.

We also perform detailed analysis to understand which aspects of BTM are most important for these gains. Ensembled ELMFORESTS outperform ensembling across random data splits, suggesting that domain specialization is a critical component to our approach (§5.1). We also show that performance is robust to a range of initializations, including the choice of the compute budget allocation (§5.2) and data (§5.3) for training the initial LM. Our ELMFORESTS are also able to forget domains by removing the relevant ELM, as long as they were not included in the initialization phase (§5.3).

Finally, we perform a preliminary scaling study on a training corpus with 192B whitespace-separated tokens (§6.3). Building on our findings, we use BTM to incrementally train a total of 64 experts which form a ELMFOREST. Our scaled ELMFOREST performs comparably with a 1.3B parameter TRANSFORMER-LM trained with 2.5 times the total GPU hours. We find that benefits of BTM increase with the number of domains in the training corpus.

These results provide compelling evidence for the promise of scaling large language models with many smaller, independently trained ELMs. We envision that this work lays the foundation for democratized model development at inclusive compute budgets — that groups with different resource constraints and research interests may combine efforts to build open-sourced, community-authored large language models, comprised of continually-evolving repositories of EXPERT LMs.

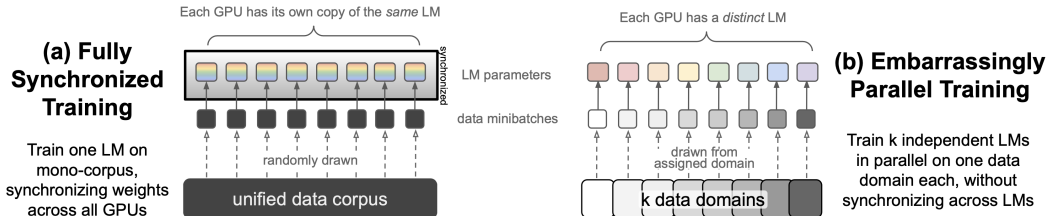


Figure 2: **Fully Synchronized vs. Embarrassingly Parallel Training (§3)**. (a) In the fully synchronized data-parallel training of a TRANSFORMER-LM, all parameters are synchronized across all GPUs. In large LMs, this synchronization incurs hefty cross-node communication costs. (b) In embarrassingly parallel training (our work), individual models are trained on each domain, with no parameter synchronization between those models, which eliminates cross-node communication costs between models.

We release our code publicly.²

Our contributions in this work are, concisely:

- **Branch-Train-Merge (BTM; §3)**, consisting of an initial shared training phase, followed by parallelized training of many ELMs each specializing to one data domain. Inference with an ELMFOREST, either through ensembling or parameter averaging to produce a single LM, outperforms compute-matched baselines at a variety of parameter scales (§4).
- **Analysis (§5)**. We demonstrate that the improved performance of ELMFORESTs is not merely due to ensembling more parameters, and then study the performance tradeoffs of various training and inference decisions. ELMFORESTs trained with BTM robustly outperform compute- and parameter-matched approaches across many settings.
- **Incremental BTM training on 64 textual domains (§6)**. We demonstrate increasing benefits of BTM training over a large baseline as we scale number of domains, up to at least 64 domains.

2 ELMFORESTS

ELMFORESTS are designed to be embarrassingly parallel (Figure 2) and rapidly customizable by completely isolating the influence of distinct data splits to different LMs, as defined in this section.

2.1 Model definition

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 specifically define ours to be *domain* specialists and specialize the full LM instead of just model components. We follow Gururangan et al. 2022 and define domains by *provenance*, or the source of the document (e.g., whether it is a legal document or computer science research paper), which yields simple and interpretable corpus segmentations, and is useful for identifying ELMs in our experiments.³ These can potentially be extended to multi-lingual, -modal, -task, or other types of data splits, but we leave these directions for future work. ELMs remain independent at all stages of training and inference, enabling the functions described below.

2.2 Adding and removing ELMs

We can modify the domain coverage of an ELMFOREST at any time by incorporating new ELMs specialized to different domains or removing existing ELMs in the set. These mechanisms stand in contrast to existing control techniques that have been introduced to steer LMs towards (Keskar et al., 2019; Gururangan et al., 2020; Dathathri et al., 2020) or away (Welleck et al., 2019) from certain behaviors. Existing techniques tend to be expensive, require retraining the model with different

²<https://www.github.com/hadasah/btm>

³See §8 for a discussion on the possible limitations of this domain definition.

objectives, or do not provide strong guarantees on how the LM may behave at test-time (Gehman et al., 2020). In contrast, ELMFORESTS allow for explicit inference-time application of constraints on the provenance of training data; they store all information from a domain in the associated ELM, which is fully independent of all others. Removing any expert, even after fully training the model, guarantees the associated data will be fully ablated and never influence future model predictions.

2.3 Ensembling the ELMFOREST

ELMFORESTS support two inference modes, which trade off model efficiency and size for performance. In this first method, we ensemble the output probabilities of multiple ELMs. This allows us to generalize to new text of unknown domain provenance. We use the *cached prior* method proposed in Gururangan et al. (2022), summarized below.

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

$$p(X_t | \mathbf{x}_{<t}) = \sum_{j=1}^n p(X_t | \mathbf{x}_{<t}, D = j) \cdot p(D = j | \mathbf{x}_{<t}) \quad (1)$$

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

$$p(D = j | \mathbf{x}_{<t}) = \frac{p(\mathbf{x}_{<t} | D = j) \cdot p(D = j)}{p(\mathbf{x}_{<t})} = \frac{p(\mathbf{x}_{<t} | D = j) \cdot p(D = j)}{\sum_{j'=1}^k p(\mathbf{x}_{<t} | D = j') \cdot p(D = j')} \quad (2)$$

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: $p(D = j) = \sum_{i=1}^N \lambda^i \cdot p(D = j | \mathbf{x}_{<T}^{(i)})$. 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.

This inference procedure naively requires a forward pass through all ELMs in the ELMFOREST, but we observe in practice that the domain posterior is sparse, even as the number of ELMs increases, which suggests that top- k selection of EXPERT LMs can reduce inference time costs with negligible effects on performance. We quantify this sparsity and the effectiveness of using only the top- k experts in our scaled-up experiments (§6.5).

2.4 Averaging ELM parameters

As an alternative to ensembling, we can also use parameter averaging (Izmailov et al., 2018; Wortsman et al., 2022a; Matena and Raffel, 2021) to collapse the ELMFOREST into a single LM. This operation keeps inference cost constant regardless of how many ELMs are added to the set. When scaling the ELMFOREST to many domains, we also use ELMFOREST parameter averaging to initialize new experts, as we see in the next section. We experiment with several techniques to construct the average for a target domain in §4.4, including a uniform average and using the best performing expert. We find that using the cached prior from §2.3 to define a weighted average LM is the strongest method. ELMFOREST weighted parameter averages outperform TRANSFORMER-LM baselines at all model sizes studied, and continue to outperform TRANSFORMER-LMs and approach ELMFOREST ensembling performance as we increase the number of domains (§6.5).

3 BRANCH-TRAIN-MERGE (BTM)

BRANCH-TRAIN-MERGE training of ELMFOREST models is incremental and embarrassingly parallel (Figure 2). EXPERT LMs are trained completely independently in batches, starting from an LM trained on a shared, heterogeneous corpus. Figure 3 summarizes the process, which we detail in this section.

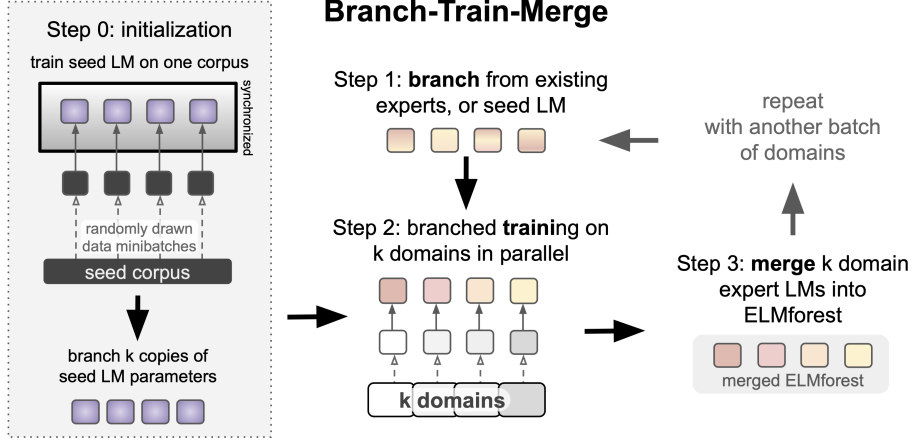


Figure 3: **BTM training process overview (§3)**. Our BTM method is initialized with a seed training phase (Step 0), in which a single LM is trained on a corpus. The resulting parameters are branched, or copied, k times (Step 1) and we continue training each LM on a unique assigned data domain, resulting in k ELMs (Step 2). Trained ELMs are merged into the ELMFOREST (Step 3), and the process can be repeated from Step 1, initializing new ELMs with parameter averages of existing ELMs. After the seed phase (Step 0), ELMs are fully disconnected, with no communication between them. This process is shown at $k = 4$ for simplicity, but our experiments in §4-5 use $k = 8$ domains and train for one iteration through BTM. Our experimental setup for §6, illustrated in Figure 6, trains on $k = 64$ domains over 4 iterations of BTM.

3.1 The Branch-Train-Merge Iteration

Each BRANCH-TRAIN-MERGE 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 $D_E = \{d_1, \dots, d_k\}$ currently modeled by E . In this section, we first describe the inductive case of $k > 0$ and then describe how to train the initial model θ_0 .

Step 1 (Branch): Sprouting a new ELM The new ELM parameters are a function of the current expert set E . Given some vector of weights $w = \{w_1, w_2, \dots, w_k\}$ over the existing experts $\theta_1, \theta_2, \dots, \theta_k$, we can initialize the new expert with the weighted parameter average $\theta_{k+1} \leftarrow \sum_{i=1}^k w_i \theta_i$. We experiment with a few ways to set w , including initializing from only the nearest ELM or – our best performing approach – a parameter average of ELMs according to their domain posterior on the new data d_{k+1} (§4.4).

Step 2 (Train): Growing the ELM 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* later, to distinguish it from other training regimens we compare to.

Step 3 (Merge): Transplanting the ELM We merge the new ELM θ_{k+1} , which now represents the domain d_{k+1} , into E to create an updated set: $E' = E \cup \{\theta_{k+1}\}$ and $D_{E'} = D_E \cup \{d_{k+1}\}$.

These operations are described for a single ELM, but can be parallelized to add multiple ELMs in a batch or iterated to add them asynchronously.

3.2 Step 0 (Initialization): Seeding the ELMFOREST

On the first iteration of BTM, $E = \emptyset$; we have no ELMs in the set to branch from. Instead of initializing the first ELMs of the set randomly, we hypothesize that ELM performance is boosted by branching from pretrained LM parameters, since multi-phase adaptive pretraining is an effective way to develop domain-specific language models (Gururangan et al., 2020), and parameter interpolation techniques work best with models that have a shared initialization (Izmailov et al., 2018; Frankle

et al., 2020; Wortsman et al., 2022b; Matena and Raffel, 2021; Wortsman et al., 2022a). Specifically, we perform a *seed phase*, training a *seed LM* θ_{seed} on some data corpus d_{seed} , which can be used to initialize the first batch of ELMs in the set. In §4.4, we find that the seed phase is important for enabling ELMFOREST parameter averaging (§2.3). Ensembling ELMFORESTS trained with BTM results in comparable performance across a wide range of seed LM training settings (§5).

3.3 Scale: Expanding the ELMFOREST

ELMFORESTS can be scaled incrementally by repeating the steps in §3.1 on batches of new domains. Incremental training allows new ELMs to be initialized with parameter averages of existing LMs when branching. We hypothesize that this allows us to amortize the compute necessary to train new ELMs as BTM proceeds; subsequent ELMs can be trained for shorter periods of time by leveraging the pretraining of existing ELMs in the set.

4 Core Experiments and Results

We first compare BTM training to compute-matched baselines, to carefully measure the efficiency gains. We use the simplest form of BTM training on a set of $k = 8$ domains, consisting of one iteration through the Branch-Train-Merge cycle. We compare inference through ensembling and different parameter averaging methods.

4.1 Experimental Setup

Data Following prior work, we use the data introduced by Gururangan et al. (2022), which consists of 8 training and 8 evaluation (primarily English-language) domains.⁴ These 16 domains cover a diverse span, from Web text and U.S court opinions for training to GitHub and COVID-19 research papers for evaluation. Details are listed in Appendix Table 9.

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) at the following model sizes, specified by parameter count: 125M (small), 350M (medium), 750M (large), 1.3B (x1). 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

- **TRANSFORMER-LM:** The first baseline is a standard Transformer LM, implemented with distributed data parallelism (Li, 2021). This is identical to the DENSE model from Gururangan et al. (2022), in which data from each domain is balanced.⁵
- **DEMIX:** We follow the training procedure outlined in Gururangan et al. (2022), where feedforward layers in the Transformer are trained to specialize as domain experts, but all other parameters are synchronized, as in the TRANSFORMER-LM. Gururangan et al. (2022) demonstrated that DEMIX LMs exhibit better domain specialization and generalization than other sparsely activated (e.g., MoE) models.
- **ELMFOREST:** We first conduct a seed phase to initialize the ensemble with LM parameters (§3.2), then conduct branched training on the ELMs (§3.1), all of which are initialized with the seed LM. Between the seed and branched phases, we continue training from the saved optimizer state.⁶ We ensemble the outputs of experts for all results with this model, using the procedure outlined in §2.3.

These models are compute-matched, since computation is typically the limiting factor in model training. Like other sparse models (Fedus et al., 2022; Lepikhin et al., 2021; Gururangan et al., 2022),

⁴We refer to the novel domains from Gururangan et al. 2022 as evaluation domains in this paper.

⁵We find, in line with Gururangan et al. (2022), that balancing data domains achieves better performance than without data balancing. Results comparing these two settings are in Appendix Table 13.

⁶We found in earlier experiments that resetting or reusing the optimizer state has little effect on performance; we reuse the optimizer state to fix the learning rate schedule across all experiments.

125M				350M			
	T-LM 125M	DEMIX 512M	ELMFOREST 1B		T-LM 350M	DEMIX 1.8B	ELMFOREST 2.8B
Train	19.9 _{0.23}	18.2 _{0.82}	17.2 _{0.02}	Train	16.3	15.0	14.7
Eval	25.2 _{0.18}	23.4 _{0.54}	22.4 _{0.12}	Eval	20.8	19.9	18.6
All	22.5 _{0.14}	20.8 _{0.63}	19.8 _{0.05}	All	18.5	17.5	16.7

750M				1.3B			
	T-LM 750M	DEMIX 3.8B	ELMFOREST 6B		T-LM 1.3B	DEMIX 7B	ELMFOREST 10.4B
Train	14.7	13.5	13.4	Train	14.2	13.7	13.0
Eval	19.3	17.7	16.7	Eval	18.4	17.6	16.3
All	17.0	15.6	15.0	All	16.3	15.6	14.6

Table 1: **ELMFORESTS trained with BTM outperform all baselines across multiple model scales (§4.2).** Average test-set perplexity (\downarrow) for each model scale (125M, 350M, 750M, 1.3B parameters) across the 8 training, 8 evaluation, and all 16 domains described in §4.1. Total parameters are shown for each model type at each scale. At 125M parameter per GPU scale, we show the mean and standard deviation of results over 8 random seeds. For BTM, we show results with 50% of compute dedicated to the seed phase. DEMIX outperforms TRANSFORMER-LM, abbreviated as T-LM. ELMFORESTS trained with BTM consistently achieve the lowest average test perplexity.

ELMFORESTS decouple compute and parameters; we can train many more parameters at the same computational cost as the equivalent TRANSFORMER-LM. Total parameter counts are in Table 1.

Training settings To disentangle variations in GPU speed, we use *number of updates* as our computational budget in these experiments. We choose the number of updates in our budget so that training completes in roughly 48 wall-clock hours: 80k, 32k, 24k, and 12k updates for the 125M, 350M, 750, 1.3B parameter per GPU models, respectively. We additionally report the average updates per second of each model, and present performance as a function of GPU hours, to illustrate efficiency improvements. We use 16, 32, 64, and 128 GPUs in parallel for the 125M, 350M, 750M, 1.3B parameter TRANSFORMER-LM and DEMIX baselines, respectively. We also use these GPU budgets for the seed phase in the ELMFOREST. For branched training, we divide these GPU budgets equally among the ELMs; for example, the 1.3B parameter per GPU ELMFOREST uses 16 GPUs for each of the 8 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. For all experiments, 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.

4.2 Performance Comparisons

Our results are shown in Table 1. Atbvfvhufckrjihhubfggkt these model scales, ELMFORESTS trained with the BTM procedure outperform both the sparsely trained DEMIX LM and the densely trained TRANSFORMER-LM baselines. The improvements in performance we observe over DEMIX layers suggest that complete isolation of all LM parameters results in better specialization of domain experts. At the 125M parameter scale, we report the mean over 8 random seeds, as well as the standard deviation.

4.3 Efficiency Comparisons

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, as shown in Table 2. Additionally, the embarrassingly parallel, fully disconnected nature of 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 2 nodes of 8 GPUs simultaneously, so that we were able to conduct some BTM experiments on

	Average updates per second, normalized (\uparrow)		
	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 2: **BTM training is more efficient (§4.3)**. Average updates per second (\uparrow) for each setup and model size, normalized by the average updates per second during fully synchronized training of the TRANSFORMER-LM. The embarrassingly parallel training used during the branched phase of BTM achieves higher updates per second than fully or partially synchronized training. The efficiency gains from embarrassingly parallel training become more substantial with larger model size – and more nodes used in parallel. At 1.3B parameters per GPU, BTM’s branched training provides a 33% speedup over fully synchronized training. To better leverage this speedup, we experiment with dedicating a larger proportion of the total compute budget to branched ELM training in §5.2. These speed estimates may vary considerably with hardware and environment factors.

limited (e.g., academic-scale) resources. Our TRANSFORMER-LM training experiments, on the other hand, consumed 16 nodes of 8 GPUs simultaneously. We observe this phenomenon throughout all experiments; 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.

4.4 ELMFOREST Parameter Average

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. Here, we explore an alternative way to combine experts to improve generalization with no additional inference costs relative to the TRANSFORMER-LM baseline: parameter averaging (§2.4). Given some weight vector w over k ELMs $\{\theta_i, \dots, \theta_k\}$, we define a single model such that all of 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 a number of options:

- **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 that corresponds to the maximum probability in the domain posterior. (§2.3). This collapses the ELMFOREST into the estimated best-performing ELM for the target dataset.
- **Posterior** We set w to be the domain posterior (§2.3), computed on the validation set.

We show our results on ELMFOREST parameter averaging in Table 3. Using uniform weights underperforms all baselines, even TRANSFORMER-LMs, highlighting the importance of domain relevance in 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 provides consistent performance improvements over TRANSFORMER-LMs at no additional inference cost.

The best parameter averaging performance does not reach the performance of ensembling. However, the lower inference costs and simplicity of deployment may make averaging the preferred inference technique for certain resource-constrained applications. We explore these computational tradeoffs further in §6.5. Due to its superior performance, we report results for ELMFOREST with ensembling, unless otherwise noted.

Surprisingly, we observe poor performance of model averaging at the 125M scale regardless of the type of weight vector we use. We see later that the amount of compute allocated to the seed phase has a critical effect on the viability of ELMFOREST parameter averaging (§5.2). With sufficient seed training, parameter averaging outperforms TRANSFORMER-LM at all scales.

	Train Domains PPL (\downarrow)			
	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
	Eval Domains PPL (\downarrow)			
	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 posterior)	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 3: **ELMs can be combined through parameter averaging (§4.4)**. Average test-set perplexity across the 8 training domains (top) and 8 evaluation domains (bottom), from the models in Table 1, comparing techniques to collapse ELMFOREST into a single LM. 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. The relatively poor performance of ELMFOREST parameter averaging for the 125M parameter model is investigated further (and improved) in §5.2.

4.5 Summary

ELMFORESTS trained with BTM demonstrate performance gains over update-matched DEMIX and TRANSFORMER-LMs. Additionally, BTM is more efficient (higher updates per second) than TRANSFORMER-LM or DEMIX training, due to a substantial reduction in cross-GPU communications for parameter synchronization. Collapsing the ELMFOREST into a single LM via parameter averaging results in consistent benefits over the TRANSFORMER-LM at no additional inference costs, approaching the performance of ELMFOREST ensembling.

5 Analysis

In the core results of §4, we fixed the training setup to conduct a controlled comparison of BTM to baseline methods. Given the evidence supporting the improved performance of ELMFOREST trained with BTM, we now analyze the importance of various training and inference decisions. We study the effects of the increased parameter count in an ELMFOREST, of seed LM training data, as well as the compute budget dedicated to the seed phase, and also discuss the observed tradeoffs between performance, efficiency, and functionality.

5.1 ELMFOREST performance is not simply the result of ensembling more parameters

To determine whether ensembling extra parameters is sufficient for the improvement our method provides, we compare multiple variations of LM ensembles:

Random Ensemble (seed init) A set of LMs trained on random data splits, to assess the effect of removing the domain specialization of ELMs. We first pool the training and development sets of our 8 train domains and divide into 8 random data splits, then execute the BTM procedure on those random splits, dedicating 50% of training to the seed phase.⁷

⁷In Figure 1, this setting is labeled *Random Ensemble*.

125M				350M			
	Random Ensemble (seed init)	ELM FOREST (random init)	ELM FOREST (seed init)		Random Ensemble (seed init)	ELM FOREST (random init)	ELM FOREST (seed init)
Train	23.0	18.2	17.2	Train	19.9	15.3	14.7
Eval	26.0	23.4	22.4	Eval	23.1	21.3	18.6
All	24.7	20.8	19.8	All	21.5	18.3	16.7

750M				1.3B			
	Random Ensemble (seed init)	ELM FOREST (random init)	ELM FOREST (seed init)		Random Ensemble (seed init)	ELM FOREST (random init)	ELM FOREST (seed init)
Train	17.4	14.4	13.4	Train	17.4	13.3	13.0
Eval	20.9	19.3	16.7	Eval	20.4	17.8	16.3
All	19.2	16.9	15.0	All	18.9	15.6	14.6

Table 4: **Domain expert ensemble outperforms random split ensemble (§5.1)**. Average test-set perplexity (\downarrow) for each model scale across the 8 training, 8 evaluation, and all 16 domains described in §4.1. Training ELMFOREST experts with random data splits performs much worse than the proposed method of training ELMFOREST experts on domains of data defined by provenance. This suggests that ensembling with more parameters is not sufficient, but that domain specialization is a crucial component for the performance improvements we observe.

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 §4, which follows the BTM training procedure on the 8 train domains, and splits the compute budget such that 50% of the updates are dedicated to seed training and 50% to branched ELM training.

These results are in Table 4. 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. We speculate that the random ensemble is poor because its constituent models make correlated errors during evaluation (Gontijo-Lopes et al., 2022).

5.2 ELMFOREST performance robust to wide range of seed LM training compute allocations

The ELMFOREST (random init), which has no seed training, underperforms ELMFOREST (LM init) in §5.2, indicating that the seed training phase is essential. On the other hand, TRANSFORMER-LM, which is equivalent to 100% seed training, also underperforms ELMFOREST (LM init) in §4, 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 ELM training at dramatically reduced computational cost relative to the prevailing training of TRANSFORMER-LMs, due to reduced GPU communication (§4.3). 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.

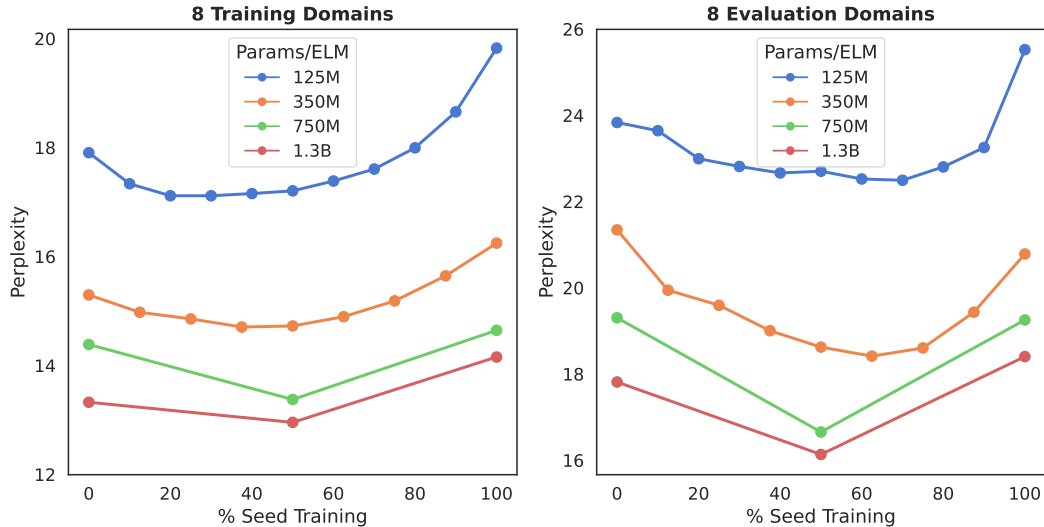


Figure 4: **ELMFOREST ensembling performance is robust to most seed training compute allocations (§5.2).** Test perplexity averaged across the 8 training (left) or 8 evaluation (right) domains (from §4.1) when varying the proportion of compute allocated to seed training, but keeping total compute budget fixed. All models use the same random initialization. The best training domain performance occurs when roughly 20–50% of updates are seed training, whereas best evaluation domain performance occurs when roughly 50–80% of updates are seed training. This supports the hypothesis that seed training is useful for generalization. However, performance varies little between 20–80%, and almost all BTM settings perform better than both TRANSFORMER-LM (100% seed) and ELMFOREST-random init (0% seed) training across model sizes.

ELMFOREST averaging performance strongest with about 60-70% seed LM training Separate experiments, shown in Figure 5, suggest a strong effect of the seed phase on the viability of ELMFOREST averaging. We find that ELMFOREST averaging does not work with ELMs trained from random initialization (i.e., with no seed phase), resulting in perplexities in the thousands. Since these ELMs still share the same random initialization, we conclude that there is importance to seeding the ELMFOREST with a shared set of partially trained LM parameters.

The averaging procedure is more robust to seed phase compute budget when evaluated on train domains. 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 3. However, this optimal range shifts much lower, to about 40%, for the next largest scale. Results for ELMFOREST parameter averaging at 50% seed training for 350M, 750M, and 1.3B parameter scales do outperform TRANSFORMER-LM baselines on both train and evaluation domains (Table 3).

5.3 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 §5.2 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 overall, but even seed training on JavaScript code alone yielded better results than the compute-matched TRANSFORMER-LM baseline. This lack of sensitivity to seed phase training corpus suggests that initializing ELMs with parameters of a model checkpoint is key – perhaps regardless of the performance of that model on any target domain. However, the ELMFOREST (random init) models in Table 1, which use identical random initialization, achieve worse performance, indicating the benefit of pretrained model parameters.

Domain forgetting through ELM removal is mostly robust to seed training corpus We introduced in §2.2 the possibility of removing ELMs to reduce the influence of specific training domains

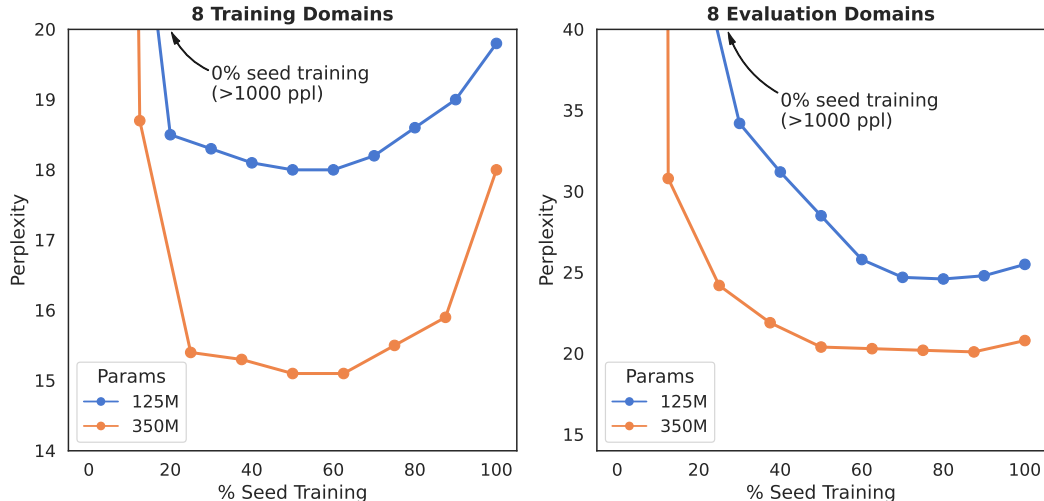


Figure 5: **The seed phase is vital to our ability to parameter average ELMs (§5.2).** Test perplexity averaged across the 8 training (left) and 8 evaluation (right) domains when averaging ELMFOREST with different seed training compute allocations for the 125M and 350M parameter LMs. All models use the same random initialization. Seed training is critical to reasonable averaging performance. Lowest perplexity is achieved at slightly higher seed phase training proportions than when using output ensembling; however, as with output ensembling, the variation in performance is small throughout a wide range of seed training phase lengths. For training domains, effective parameter averaging performance can be achieved with even less seed training.

		Average Test PPL (↓)		
		Train	Evaluation	Overall
seed corpus	TRANSFORMER-LM	19.8	25.5	22.7
	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 (§5.3).** Test perplexity averages on the 8 training and the 8 novel test sets (from §4.1), as well as averaged across all 16, with different training corpora used in seed LM training. All models are of the 125M parameters per GPU scale. All ELMFORESTs outperform the TRANSFORMER-LM baseline. Notably, even training the seed LM on JavaScript results in lower perplexity than the TRANSFORMER-LM, suggesting that BTM training is not overly sensitive to the selection of the seed training corpus.

at inference time (e.g., those that contain stale or harmful text). Now, we evaluate the effectiveness of removing ELMs in practice. In Table 6, we show the performance of ELMFOREST ensembles on the training domains when using all ELMs (from Table 5), and compare to performance when removing the relevant ELM. For example, to evaluate on REDDIT, we keep active all ELMs *except* the REDDIT ELM. Removal of an ELM from an ELMFOREST *guarantees* that that domain will be forgotten, if the seed training corpus did not include that domain’s data (§2.2)⁸. We find that performance does indeed degrade appreciably on ELMFORESTs when removing the ELM, indicating that ELMFORESTs are capable of effectively forgetting a data domain without any gradient updates to the parameters. Even in the model with seed training on the 8 training domains, removal of a ELM greatly degrades test perplexity in the corresponding domain. Taken together with the previous results,

⁸The actual effect of ELM removal on model performance may depend on the overlap between the removed domain and other training domains. Future work may also investigate the effects of overlap and any required guarantees of domain disjunction.

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

Table 6: **The ability to reduce the influence of domains through ELM removal is (mostly) robust to seed training corpus (§5.3).** 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 perplexity increases are desired, as they suggest the ease of removing data from the ELMFOREST’s distribution after training has completed. This is a useful operation to reduce the influence of training domains that contain harmful text or become stale.

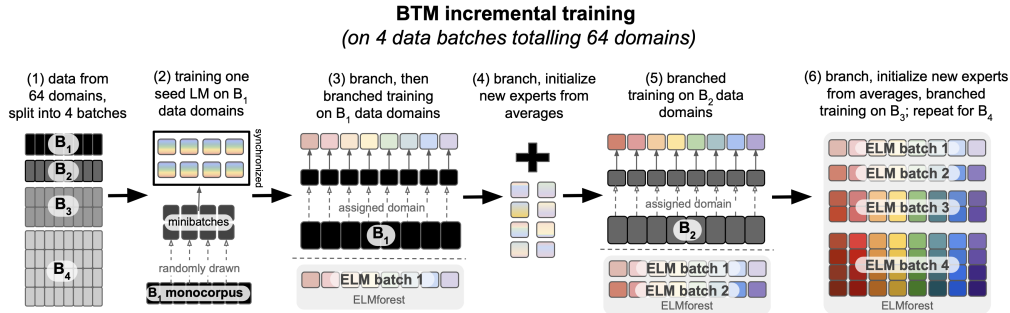


Figure 6: **BTM incremental training on 64 domains (§6).** We now iterate over the BTM steps (§3; Figure 3) to train incrementally on 4 data batches B_1 – B_4 , totalling 64 domains. After training the seed LM on the pooled B_1 domains, we train ELMs on each B_1 domain independently, then initialize new ELMs from weighted averages of existing expert parameters. These new ELMs are each trained to specialize on a B_2 domain. We repeat this process for data domains in B_3 , then B_4 , initializing new ELMs from the set of existing experts. After this full process, all 64 total ELMs over 4 batches have been merged into the same ELMFOREST and may be ensembled or averaged together for inference.

it may be possible to carefully curate the seed corpus (e.g., to avoid including harmful language) and domain data, to provide even stronger guarantees on the ability to forget unwanted domains via expert removal with minimal performance degradation.

6 Incrementally Training an ELMFOREST on 64 Domains

Next, using the best settings discovered in §4 and §5, we repeat the BTM procedure to scale into an 80-domain corpus, which contains 64 training domains and 16 evaluation domains. Our training procedure is summarized in Figure 6. In §6.3, we demonstrate how the performance of the ELMFOREST scales relative to a large TRANSFORMER-LM trained on the same set of domains from random initialization. We then examine the sparsity of the resulting ELMFOREST at inference time (§6.4), and compare techniques to reduce the inference cost of larger ELMFORESTS (§6.5).

6.1 Experimental Setup

Data: 80-domain Corpus Using provenance as our domain label, we curate a corpus of 64 training domains and 16 evaluation domains (Table 7). Corpora were selected for their research-friendly data licensing, substantial size, and content diversity, and are drawn from a collection of publicly available data repositories (e.g., Gao et al., 2021; Lo et al., 2020). Full details on the data sources that make up these corpora can be found in Appendix Table 10 and 11.

80-DOMAIN CORPUS: 192.3B WHITESPACE-SEPARATED TOKENS

Category	Domains
SEMANTIC SCHOLAR (26.6%)	Medicine (5.2%), Biology (4.7%), CS (3.4%), Physics (2.7%), Math (2.3%), Unlabeled (1.3%), Psychology (1.2%), Chemistry (1.0%), Economics (0.8%), Engineering (0.7%), CORD19 (0.6%), Material Science (0.5%), Geology (0.5%), Sociology (0.5%), Business (0.3%), Political Science (0.2%), Geography (0.2%), Environmental Science (0.1%), History (0.1%), Philosophy (0.1%), ACL (0.1%), Art (0.05%)
GITHUB CODE (22.4%)	JavaScript (3.7%), Java (3.5%), HTML (2.7%), C (2.5%), C++ (1.9%), Python (1.5%), C# (1.2%), PHP (1.1%), Markdown (1.1%), Code Contests (1.0%), GO (1.0%), CSS (0.7%), Ruby (0.4%)
WEB FORUMS (17.5%)	Reddit Dialogues (13.0%), StackOverflow (1.7%), Twitter (0.9%), StackExchange (0.8%), HackerNews (0.4%), Gaming Subreddits (0.1%), Sports Subreddits (0.1%)
WEB CRAWL (16.0%)	C4 (5.2%), RealNews (5.2%), OpenWebText (3.4%), Wikipedia (en) (1.3%), WMT News Crawl 2021 (0.5%), 1B Words Corpus (0.4%)
BOOKS (5.8%)	Stories (3.8%), Gutenberg Books (1.6%), BookCorpus (0.4%)
LEGAL TEXT (5.5%)	Legal Case Law (5.5%), Supreme Court Opinions (HTML) (0.1%)
REVIEWS (5.0%)	Books Reviews (2.1%), Amazon Reviews (1.1%), Electronics Reviews (0.5%), Clothing, Shoes and Jewelry Reviews (0.5%), Home and Kitchen Reviews (0.4%), Yelp Reviews (0.3%), Sports and Outdoors Reviews (0.3%), Movies and TV Reviews (0.3%)
OTHER (1.3%)	DM Mathematics (0.8%), OpenSubtitles (0.4%), USPTO (0.1%)
EVALUATION DOMAINS (TEST ONLY)	Enron, #COVID-19 Tweets, IMDB, TOEFL exams, Congressional bills, Legal Contracts, /r/careerquestions, /r/india, /r/hiphopheads, Irish Parliamentary Speeches, SQL, Rust, Perl, TeX, FORTRAN, Breaking News

Table 7: **Overview of the 80-domain corpus (§6.1).** The 80 domains that make up the multi-domain corpus we train and evaluate on, presented here in 8 descriptive categories for ease of inspection. For each of the 64 training domains, we include the percentage of the total number of tokens (in the entire corpus) comprising that domain. At the bottom, we include the 16 evaluation domains. All domains additionally include 1M tokens for validation and test data each. We include full details of each corpus in Appendix Table 10 and 11.

Domain Batches We first organize the 64 training domains into four ordered batches of increasing size, which form a curriculum that we denote B_1 to B_4 . We maintain the training domains of Gururangan et al. (2022) as the first batch (B_1), which allows us to leverage the best models from §4 and §5, but organize the batches B_2 - B_4 randomly. See Appendix Table 10 for our batch assignments.

Model hyperparameters and training settings We follow the same settings from §4.1, but only compare models at the 350M (medium) and 1.3B (xl) scales.

6.2 Compared Models

The models in these experiments are intentionally **not** compute-matched, to demonstrate the efficiency gains of our approach over an expensive baseline.

TRANSFORMER-LM Our baseline is a large, randomly-initialized 1.3B parameter transformer LM, trained for 6144 GPU hours (with 128 GPUs) on all 64 domains. We use the same training settings for this baseline as the TRANSFORMER-LM in §4.

ELMFOREST We train on the 64 domains incrementally using 4 GPUs per ELM (see Figure 6 for a diagram of the overall process). For each batch of domains, we follow the basic procedure of BTM (§3): branch a new set of experts, train them on the batch, and merge trained ELMs back into the ELMFOREST. We start with ELMs from §5 trained with 75% seed training and 25% branched training on the B_1 domains, which achieves the best evaluation-domain performance in our initial experiments (Figure 4). To train on B_2 , we branch new ELMs with weighted averages of B_1 experts, then train on B_2 for 40 GPU hours. For B_3 , we branch new ELMs again, with weighted averages of B_1 and B_2 experts, and train on B_3 for 40 GPU hours. Finally, we scale into B_4 by training for 20 GPU hours on each domain, initializing those ELMs with the weighted average of ELMs in B_1 , B_2 , and B_3 .⁹ Our final ELMFOREST contains 64 ELMs and is trained for a total of 2565 GPU hours.

⁹The choice of compute budget at each training stage is an open question for future work. This may be informed by how much data existing ELMs have already been exposed to and how much data exists in the group.

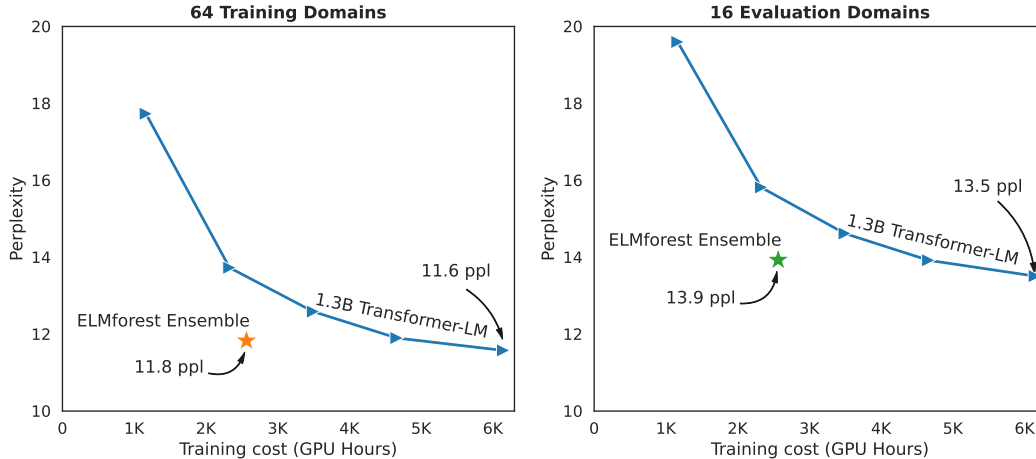


Figure 7: **ELMFOREST achieves performance comparable to a large TRANSFORMER-LM baseline at a much smaller computational budget (§6.3).** Efficiency-performance trend for training (left) and evaluation (right) domains when ELMFORESTS are incrementally trained on a 64 domain corpus, versus a large TRANSFORMER-LM that is trained on 64 domains from scratch. ELMFORESTS achieve better perplexity at smaller compute budgets than TRANSFORMER-LMs. ELMFORESTS perform comparably to the 1.3B TRANSFORMER-LM, using only about 40% of the TRANSFORMER-LM total budget.

6.3 Results

Figure 7 presents the performance of ELMFOREST and TRANSFORMER-LM as a function of the total compute budget used for training on the 64 domains. We observe that ELMFOREST with ensembling achieves perplexities on training and evaluation domains that are comparable to a large TRANSFORMER-LM, despite using only 40% of the total compute.¹⁰ ELMFOREST effectively matches training domain performance of the large TRANSFORMER-LM because the TRANSFORMER-LM exhibits worse average performance on each individual training domain as it is exposed to more domains during training (see Appendix Figure 10 for more details).¹¹ On the other hand, we always observe monotonically non-worsening performance on each training domain with BTM, since previously trained ELMs are unaffected when branching into new domains.

The order and composition of the domain batches likely affects the performance-efficiency tradeoff curve for the ELMFOREST. We leave a careful analysis of these factors, and other optimal settings for scaling ELMFORESTS, to future work.

6.4 Sparsity of the 64-expert domain posterior

A visualization of our resulting 64-expert ELMFOREST domain posteriors on held out validation data from the 80-domain corpus is displayed in Figure 8. We observe sparsity in the domain posterior in both the training and evaluation domains. This sparsity suggests that only a few ELMs need to be active at test time, a hypothesis we test next. We also display the top 3 ELMs for a sample of evaluation domains in Table 8. The most likely ELMs are relevant to the evaluation domain.

6.5 Reducing inference costs

The sparsity of the domain posterior in the 64-expert ELMFOREST suggests that fewer ELMs can be used at test time with minimal performance degradation. We see in Figure 9 that this is indeed true; as few as the top-8 ELMs can be ensembled together with negligible loss in performance relative

¹⁰ELMFOREST parameter averaging approaches the ensemble in performance; see §6.5 for details.

¹¹Previous work has shown that phenomenon afflicts multilingual TRANSFORMER-LMs as well, as more languages are added to the training corpus (Pfeiffer et al., 2022).

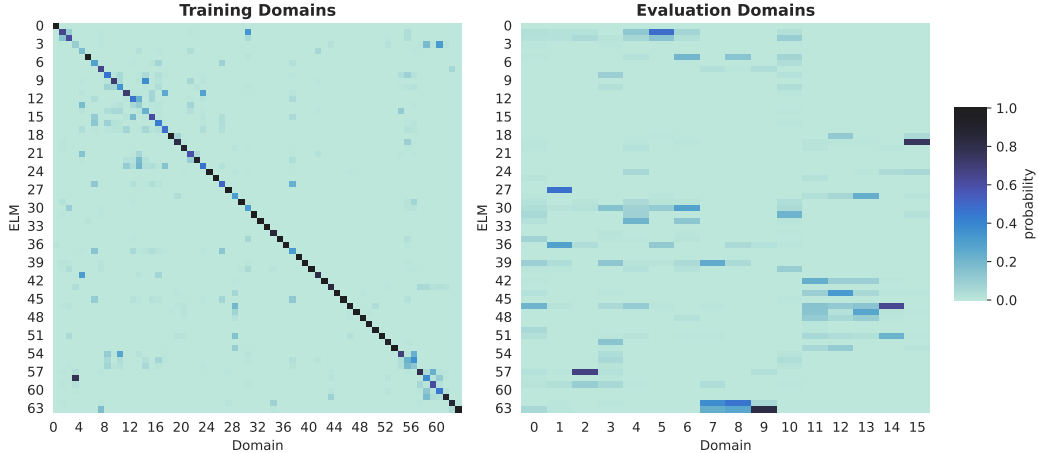


Figure 8: **Training and evaluation domain inference both use ELMs sparsely (§6.4).** Domain posterior visualization for 22.4B parameter ELMFOREST trained on 64 domains. ELM activation is extremely sparse for both training and evaluation domains.

Evaluation Domain	Top-1	Top-2	Top-3
<i>Covid Tweets</i>	Twitter (0.48)	2021 Newsrawl (0.29)	HackerNews (0.05)
<i>TeX</i>	stackexchange (0.64)	Markdown (0.22)	HTML (0.05)
<i>Congressional Bills</i>	SCOTUS Opinions (0.21)	OpenWebText (0.13)	Stackexchange (0.11)
<i>IMDB</i>	Movie & TV Reviews (0.66)	Book Reviews (0.11)	RealNews (0.05)
<i>Legal Contracts</i>	C4 (0.30)	Legal (0.21)	SCOTUS Opinions (0.14)

Table 8: **Top-3 ELMs (with probabilities in parentheses) under the domain posterior for a sample of evaluation domains (§6.4).** The most likely ELMs under the domain posterior are usually highly relevant to the evaluation domain by inspection.

to the 64-expert ELMFOREST. Even with just the top-1 ELM, we see better performance for the training cost than a TRANSFORMER-LM, at no additional inference cost.

On the other hand, weighted parameter averaging of all 64 ELMs (with weights defined by the domain posterior) provides the best performance at the same cost of the original TRANSFORMER-LM, but ensembling at least two ELMs tends to outperform the ELMFOREST average¹². The runtime efficiency of averaged models may be desirable for resource-constrained applications, where using large LMs (i.e., those that cannot fit on a single GPU) is difficult. With averaging, inference costs stay constant regardless of the number of ELMs in the ELMFOREST.

6.6 Summary

Our results demonstrate that a large collection of smaller ELMs, each of which can be branched independently and on small compute budgets (e.g., at most four GPUs) with BTM, can perform comparably with a large TRANSFORMER-LM trained with at least $2.5\times$ the compute. While the costs of using all ELMs in an ensemble at inference time is high, these costs can be reduced (with minimal performance degradation) to that of the equivalent TRANSFORMER-LM through parameter averaging. Future work may perform closer investigations of best practices for scaling and coordinating ELMFORESTS to improve on the results we report.

¹²This suggests that it may be possible to improve the algorithm to estimate weights for our parameter average. We leave this for future work.

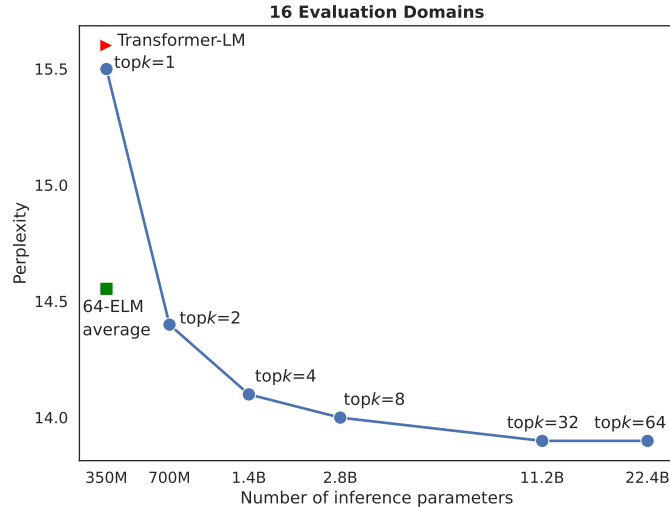


Figure 9: **Even using only the top-1 ELM outperforms the TRANSFORMER-LM baseline.** We vary the number of ELMs active at inference time, and observe performance changes in the evaluation domains. In general, using just the top 4 or 8 ELMs results in minimal performance loss relative to using 64 ELMs at test time, owing to sparsity in the domain posterior. ELM averaging results in a substantial performance improvement over TRANSFORMER-LMs at the same inference cost; but larger gains are observed with larger top- k values with output ensembling.

7 Related Work

Sparse Language Models Sparsely activated language models have been considered in a few forms (Evci et al., 2020; Mostafa and Wang, 2019; Dettmers and Zettlemoyer, 2019), but the Mixture-of-Experts (MoE) model is of particular note. Early versions (Jacobs et al., 1991) had independent feed-forward networks serving as experts. Recent MoE models (Shazeer et al., 2017) have been studied with token-based routing through backpropagation – notably, by Lepikhin et al. (2021), which applies this concept to machine translation, and Fedus et al. (2022), which simplifies the architecture to activation of only one expert per layer. Lewis et al. (2021), find an alternative approach to routing by formulating it as a linear assignment problem, and Roller et al. (2021) use a fixed hash as the gating function.

Of this line of work, ours is most closely related to Gururangan et al. (2022). In that work, DEMix layers – placed in the feedforward layers of the Transformer – contain experts which specialize on specific domains. Routing at train time is determined only by the domain label, but all experts are activated at inference time and mixed according to weights estimated from a validation set. Similarly, Pfeiffer et al. (2022) develop a multilingual expert model with language-specific routing, and Kudugunta et al. (2021) develop a multi-task expert model with task-specific routing.

Adapters Previous work has also explored extending the capacity of a model with additional specialized parameters (e.g., adapters; Houlsby et al., 2019; Pfeiffer et al., 2020; Ben Zaken et al., 2022). However, unlike these existing approaches, our approach is significantly simplified, as our ELMs each consist of an entire model which requires no additional parameters and no shared parameters. Future work may explore combining ELMs with adapters to scale into smaller domains.

Ensembles Ensemble methods are widely used in machine learning, for example in bagging, boosting, and stacking (Breiman, 1996; Freund, 1995; Wolpert, 1992). In a setting where training data is streamed, Caccia et al. (2021) define a *growing ensemble*, in which new base models are trained sequentially on incoming batches. However, their growing ensemble, incrementally trained on the randomly created batches of their setting, underperforms non-incremental methods.

Parameter Averaging Our averaging mechanism is inspired by the discovery that averaging many fine-tuned vision models improves out-of-domain generalization (Wortsman et al., 2022a; Izmailov

et al., 2018). In Wortsman et al. 2022a, the authors propose a greedy mechanism for averaging experts with uniform weights. Here, we find that uniform weighted averaging does not work for combining domain-specific models; instead we use a posterior weighted average, where the averaging weights are estimated based on the relevance of the model to the target domain. 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), where different models are trained locally to fit privacy-sensitive data on different devices and merged. However, these works have found success averaging models trained from the same random initialization, which we do not find to hold in our setting. Matena and Raffel (2021) compute a parameter average of models, estimating the optimal weights via an approximation of the Fisher information. Future work may explore these (and other) variations of weighted averages with ELMs.

Seed training Our discovery of the importance of the seed training as a critical warm-up phase for BTM is in line with findings that parameter averaging only works when models share part of their optimization trajectory (Frankle et al., 2020; Entezari et al., 2022). Future work may investigate what is learned in the seed phase that makes it so useful for ELM specialization, regardless of the corpus used for seeding. Similar to seed training, Nie et al. (2021) propose *dense-to-sparse* gating, where mixture-of-experts routing mechanisms are gradually sparsified during the course of training.

8 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 and 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 and 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.

9 Conclusion

We introduce BTM training, a new algorithm to train an ELMFOREST, which contains many EXPERT LMs that can be added and removed, ensembled, or parameter averaged at any time for efficient scaling and rapid customization. Our extensive experiments show that ELMFOREST ensembles trained with BTM outperform baselines at no additional training cost. Additionally, parameter

averaged ELMFORESTS closely approach ELMFOREST ensemble performance while enabling substantially cheaper inference. These results provide compelling evidence for the promise of scaling large language models with many smaller, independently trained ELMs. We envision that this work lays the foundation for democratized model development at inclusive compute budgets — that groups with different resource constraints and research interests may combine efforts to build open-sourced, community-authored large language models, comprised of continually-evolving repositories of EXPERT LMs.

Acknowledgments and Disclosure of Funding

This paper benefited from thoughtful feedback from a number of people: Ari Holtzman, Candace Ross, Colin Raffel, Gabriel Ilharco, Ishaan Gulrajani, Julian Michael, Mitchell Wortsman, Stephen Roller, Swabha Swayamdipta and William Fedus.

At UW, this work was partially supported by the Office of Naval Research under MURI grant N00014-18-1-2670.

References

- Roei Aharoni and Yoav Goldberg. 2020. [Unsupervised domain clusters in pretrained language models](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7747–7763, Online. Association for Computational Linguistics.
- 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. 2021. [Efficient large scale language modeling with mixtures of experts](#).
- Loïc Barrault, Ondřej Bojar, Marta R. Costa-jussà, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Philipp Koehn, Shervin Malmasi, Christof Monz, Mathias Müller, Santanu Pal, Matt Post, and Marcos Zampieri. 2019. [Findings of the 2019 conference on machine translation \(WMT19\)](#). In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 1–61, Florence, Italy. Association for Computational Linguistics.
- Jason Baumgartner, Savvas Zannettou, Brian Keegan, Megan Squire, and Jeremy Blackburn. 2020. [The pushshift reddit dataset](#).
- Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. [BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, Dublin, Ireland. Association for Computational Linguistics.
- Douglas Biber. 1988. *Variation across Speech and Writing*. Cambridge University Press.
- Daniel Blanchard, Joel R. Tetreault, Derrick Higgins, A. Cahill, and Martin Chodorow. 2013. [TOEFL11: A corpus of non-native English](#). *ETS Research Report Series*, 2013:15.
- John Blitzer, Ryan McDonald, and Fernando Pereira. 2006. [Domain adaptation with structural correspondence learning](#). In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 120–128, Sydney, Australia. Association for Computational Linguistics.
- Su Lin Blodgett, Lisa Green, and Brendan O’Connor. 2016. [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*, pages 1119–1130, Austin, Texas. Association for Computational Linguistics.
- Leo Breiman. 1996. Bagging predictors. *Machine learning*, 24(2):123–140.

- 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. 2020. [Language models are few-shot learners](#).
- Lucas Caccia, Jing Xu, Myle Ott, Marc’ Aurelio Ranzato, and Ludovic Denoyer. 2021. [On anytime learning at macroscale](#). *CoRR*, abs/2106.09563.
- Caselaw Access Project. [Caselaw access project](#).
- Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2014. [One billion word benchmark for measuring progress in statistical language modeling](#).
- Alexandra Chronopoulou, Matthew Peters, and Jesse Dodge. 2022. [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*, pages 1336–1351, Seattle, United States. Association for Computational Linguistics.
- Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. 2021. [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*, pages 4599–4610, Online. Association for Computational Linguistics.
- Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. 2020. [Plug and play language models: A simple approach to controlled text generation](#). In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.
- Tim Dettmers and Luke Zettlemoyer. 2019. [Sparse networks from scratch: Faster training without losing performance](#). *CoRR*, abs/1907.04840.
- Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. 2022. [The role of permutation invariance in linear mode connectivity of neural networks](#). In *International Conference on Learning Representations*.
- Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. 2020. [Rigging the lottery: Making all tickets winners](#). In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2943–2952. PMLR.
- William Fedus, Barret Zoph, and Noam Shazeer. 2022. [Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity](#). *Journal of Machine Learning Research*, 23(120):1–39.
- Tiago Fragoso, Wesley Bertoli, and Francisco Louzada. 2018. [Bayesian model averaging: A systematic review and conceptual classification](#). *International Statistical Review*, 86:1–28.
- Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. 2020. [Linear mode connectivity and the lottery ticket hypothesis](#). In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3259–3269. PMLR.
- Yoav Freund. 1995. Boosting a weak learning algorithm by majority. *Information and computation*, 121(2):256–285.
- Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2021. [The pile: An 800gb dataset of diverse text for language modeling](#).

- Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. 2020. **Real-ToxicityPrompts: Evaluating neural toxic degeneration in language models**. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3356–3369, Online. Association for Computational Linguistics.
- GitHub Archive Project. **Github archive project**.
- Aaron Gokaslan and Vanya Cohen. 2019. **Openwebtext corpus**.
- Raphael Gontijo-Lopes, Yann Dauphin, and Ekin Dogus Cubuk. 2022. **No one representation to rule them all: Overlapping features of training methods**. In *International Conference on Learning Representations*.
- Suchin Gururangan, Mike Lewis, Ari Holtzman, Noah A. Smith, and Luke Zettlemoyer. 2022. **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*, pages 5557–5576, Seattle, United States. Association for Computational Linguistics.
- Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. 2020. **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. Association for Computational Linguistics.
- Dan Hendrycks, Collin Burns, Anya Chen, and Spencer Ball. 2021. **Cuad: An expert-annotated nlp dataset for legal contract review**.
- Alexander Herzog and Slava Mikhaylov. 2017. **Database of Parliamentary Speeches in Ireland, 1919-2013**.
- Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. **Parameter-efficient transfer learning for NLP**. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.
- Huggingface. **Datasets**.
- Pavel Izmailov, Dmitrii Podoprikin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. 2018. **Averaging weights leads to wider optima and better generalization**.
- Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87.
- Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. 2019. **Ctrl: A conditional transformer language model for controllable generation**.
- Anastassia Kornilova and Vladimir Eidelman. 2019. **BillSum: A corpus for automatic summarization of US legislation**. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pages 48–56, Hong Kong, China. Association for Computational Linguistics.
- Sneha Kudugunta, Yanping Huang, Ankur Bapna, Maxim Krikun, Dmitry Lepikhin, Minh-Thang Luong, and Orhan Firat. 2021. **Beyond distillation: Task-level mixture-of-experts for efficient inference**. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3577–3599, Punta Cana, Dominican Republic. Association for Computational Linguistics.
- 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. 2021. **Mind the gap: Assessing temporal generalization in neural language models**. In *Advances in Neural Information Processing Systems*.
- Dmitry Lepikhin, Hyoungho Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2021. **{GS}hard: Scaling giant models with conditional computation and automatic sharding**. In *International Conference on Learning Representations*.

- Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. 2021. **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.
- Shen Li. 2021. **Getting started with distributed data parallel**.
- Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d’Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-level code generation with alphacode. *arXiv preprint arXiv:2203.07814*.
- Pierre Lison and Jörg Tiedemann. 2016. **OpenSubtitles2016: Extracting large parallel corpora from movie and TV subtitles**. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC’16)*, pages 923–929, Portorož, Slovenia. European Language Resources Association (ELRA).
- Kyle Lo, Lucy Lu Wang, Mark Neumann, Rodney Kinney, and Daniel Weld. 2020. **S2ORC: The semantic scholar open research corpus**. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4969–4983, Online. Association for Computational Linguistics.
- Li Lucy and David Bamman. 2021. **Characterizing English variation across social media communities with BERT**. *Transactions of the Association for Computational Linguistics*, 9:538–556.
- Kelvin Luu, Daniel Khoshnab, Suchin Gururangan, Karishma Mandyam, and Noah A. Smith. 2021. **Time waits for no one! analysis and challenges of temporal misalignment**.
- Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. **Learning word vectors for sentiment analysis**. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.
- Michael Matena and Colin Raffel. 2021. Merging models with fisher-weighted averaging. *arXiv preprint arXiv:2111.09832*.
- Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR.
- Hesham Mostafa and Xin Wang. 2019. **Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization**. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4646–4655. PMLR.
- Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019a. **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)*, pages 188–197, Hong Kong, China. Association for Computational Linguistics.
- Jianmo Ni, Chenguang Zhu, Weizhu Chen, and Julian McAuley. 2019b. **Learning to attend on essential terms: An enhanced retriever-reader model for open-domain question answering**. 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 335–344, Minneapolis, Minnesota. Association for Computational Linguistics.
- Xiaonan Nie, Shijie Cao, Xupeng Miao, Lingxiao Ma, Jilong Xue, Youshan Miao, Zichao Yang, Zhi Yang, and Bin Cui. 2021. **Dense-to-sparse gate for mixture-of-experts**.

- Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. [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)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.
- Ou-Yang, Lucas. [Newspaper3k](#).
- Jonas Pfeiffer, Naman Goyal, Xi Victoria Lin, Xian Li, James Cross, Sebastian Riedel, and Mikel Artetxe. 2022. [Lifting the curse of multilinguality by pre-training modular transformers](#).
- Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020. [MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, Online. Association for Computational Linguistics.
- Project Gutenberg. [Project gutenber](#).
- Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. [Language models are unsupervised multitask learners](#).
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. [Exploring the limits of transfer learning with a unified text-to-text transformer](#). *Journal of Machine Learning Research*, 21(140):1–67.
- John R. Rickford. 1985. Ethnicity as a sociolinguistic boundary. *American Speech*, 60:99.
- Stephen Roller, Sainbayar Sukhbaatar, Arthur Szlam, and Jason E Weston. 2021. [Hash layers for large sparse models](#). In *Advances in Neural Information Processing Systems*.
- David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019. [Analysing mathematical reasoning abilities of neural models](#). In *International Conference on Learning Representations*.
- Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarsz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. 2017. [Outrageously large neural networks: The sparsely-gated mixture-of-experts layer](#). In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.
- Trieu H. Trinh and Quoc V. Le. 2018. [A simple method for commonsense reasoning](#).
- Twitter Academic API. [Twitter academic api](#).
- 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. 2020. [Cord-19: The covid-19 open research dataset](#).
- Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. 2019. Neural text generation with unlikelihood training. In *International Conference on Learning Representations*.
- Wikimedia Foundation. [Wikimedia downloads](#).
- David H Wolpert. 1992. Stacked generalization. *Neural networks*, 5(2):241–259.
- 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. 2022a. [Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time](#).

- Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, and Ludwig Schmidt. 2022b. Robust fine-tuning of zero-shot models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7959–7971.
- Jing Xu, Da Ju, Margaret Li, Y-Lan Boureau, Jason Weston, and Emily Dinan. 2021. **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*, pages 2950–2968, Online. Association for Computational Linguistics.
- Yelp Reviews. **Yelp reviews**.
- Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. 2019. **Defending against neural fake news**. In *NeurIPS*.
- 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. 2022. **Opt: Open pre-trained transformer language models**.
- Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*.

A Appendix

	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 and 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., 2019a)	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 9: **Multi-domain data corpus used in §4 and §5.** 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 12. 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.

Domain	Corpus	Batch
1B	NewsWire sentences (Chelba et al., 2014)	B1
CS	Full-text CS papers from S2ORC (Lo et al., 2020)	B1
LEGAL	U.S. court opinions (Caselaw Access Project)	B1
MED	Full-text medical papers from S2ORC (Lo et al., 2020)	B1
OPENWEBTEXT [†]	OpenWebText Corpus (Gokaslan and Cohen, 2019)	B1
REALNEWS [†]	Realnews Corpus Zellers et al. (2019)	B1
REDDIT	Reddit comments from pushshift.io (Baumgartner et al., 2020)	B1
REVIEWS [†]	Amazon product reviews (Ni et al., 2019a)	B1
PSYCHOLOGY	Full-text Psychology papers from S2ORC (Lo et al., 2020)	B2
CHEMISTRY	Full-text Chemistry papers from S2ORC (Lo et al., 2020)	B2
ECONOMICS	Full-text Economics papers from S2ORC (Lo et al., 2020)	B2
ENGINEERING	Full-text Engineering papers from S2ORC (Lo et al., 2020)	B2
MATERIALS	Full-text Materials papers from S2ORC (Lo et al., 2020)	B2
GEOLOGY	Full-text Geology papers from S2ORC (Lo et al., 2020)	B2
SOCIOLOGY	Full-text Sociology papers from S2ORC (Lo et al., 2020)	B2
BUSINESS	Full-text Business papers from S2ORC (Lo et al., 2020)	B2
C4	Colossal Cleaned Common Crawl snapshot (Raffel et al., 2020)	B3
WIKIPEDIA	2022.03.01 Wikipedia snapshot (Wikimedia Foundation)	B3
STACKOVERFLOW [†]	Stackoverflow posts from The Pile (Gao et al., 2021)	B3
TWITTER [†]	English tweets from 2013-2018 (Twitter Academic API)	B3
BIOLOGY	Full-text Biology papers from S2ORC (Lo et al., 2020)	B3
JAVASCRIPT	JavaScript code from Github [MIT, BSD, Apache 2.0] (Huggingface)	B3
HTML	HTML code from Github [MIT, BSD, Apache 2.0] (Huggingface)	B3
GUTENBERG	Copyright-expired books (Project Gutenberg)	B3
POLITICAL SCIENCE	Full-text Political Science papers from S2ORC (Lo et al., 2020)	B3
ENVIRONMENTAL SCIENCE	Full-text Environmental Science papers from S2ORC (Lo et al., 2020)	B3
PHYSICS	Full-text Physics Papers from S2ORC (Lo et al., 2020)	B3
MATHEMATICS	Full-text Mathematics papers from S2ORC (Lo et al., 2020)	B3
JAVA	Java code from Github [MIT, BSD, Apache 2.0] (Huggingface)	B3
C	C code from Github [MIT, BSD, Apache 2.0] (Huggingface)	B3
C++	C++ code from Github [MIT, BSD, Apache 2.0] (Huggingface)	B3
GEOGRAPHY	Full-text Geography papers from S2ORC (Lo et al., 2020)	B3
STACKEXCHANGE [†]	Stackexchange posts from The Pile (Gao et al., 2021)	B4
PHILOSOPHY	Full-text Philosophy papers from S2ORC (Lo et al., 2020)	B4
CORD19	COVID-19 research papers (Wang et al., 2020)	B4
HISTORY	Full-text History papers from S2ORC (Lo et al., 2020)	B4
BOOKS REVIEWS [†]	Book review subset of Amazon reviews (Ni et al., 2019b)	B4
ART	Full-text Art papers from S2ORC (Lo et al., 2020)	B4
PYTHON	Python code from Github [MIT, BSD, Apache 2.0] (Huggingface)	B4
C#	C# code from Github [MIT, BSD, Apache 2.0] (Huggingface)	B4
PHP	PHP code from Github [MIT, BSD, Apache 2.0] (Huggingface)	B4
MARKDOWN	Markdown code from Github [MIT, BSD, Apache 2.0] (Huggingface)	B4
CODE CONTESTS	Programming challenge questions and answers generated by AlphaCode (Li et al., 2022)	B4
MOVIE AND TV REVIEWS [†]	Movie and TV review subset of Amazon reviews (Ni et al., 2019b)	B4
SUPREME COURT OPINIONS	Supreme Court Opinions from the Pile (Gao et al., 2021)	B4
HACKER NEWS [†]	Hacker news comments from The Pile (Gao et al., 2021)	B4
2021 WMT NEWS CRAWL	2021 newswire sentences (Barrault et al., 2019)	B4
N/A SEMANTIC SCHOLAR	Full-text papers marked N/A from S2ORC (Lo et al., 2020)	B4
OPENSUBTITLES	Movie subtitles (Lison and Tiedemann, 2016)	B4
STORIES	Filtered "story-like" Common Crawl documents (Trinh and Le, 2018)	B4
BOOKCORPUS	Self-published novels (Zhu et al., 2015)	B4
RUBY	Ruby code from Github [MIT, BSD, Apache 2.0] (Huggingface)	B4
SPORTS AND OUTDOOR REVIEWS [†]	Sports and Outdoor Reviews subset of Amazon Reviews (Ni et al., 2019b)	B4
US PATENT OFFICE	US Patents from The Pile (Gao et al., 2021)	B4
ACL PAPERS	Full-text ACL papers from S2ORC (Lo et al., 2020)	B4
YELP REVIEWS [†]	6M Yelp restaurant reviews (Yelp Reviews)	B4
DEEPMIND MATHEMATICS	Synthetically-generated mathematical question answer pairs (Saxton et al., 2019)	B4
CLOTHING REVIEWS [†]	Clothing Reviews subset of Amazon Reviews (Ni et al., 2019b)	B4
HOME AND KITCHEN REVIEWS [†]	Home and Kitchen subset of Amazon reviews (Ni et al., 2019b)	B4
GAMING REDDIT COMMENTS	Reddit comments from pushshift.io, gaming-topic (Baumgartner et al., 2020)	B4
ELECTRONIC REVIEWS	Electronic Reviews subset of Amazon Reviews (Yelp Reviews)	B4
SPORTS COMMENTS	Reddit comments from pushshift.io, sports-topic (Baumgartner et al., 2020)	B4
Go	Go code from Github [MIT, BSD, Apache 2.0] (Huggingface)	B4
CSS	CSS code from Github [MIT, BSD, Apache 2.0] (Huggingface)	B4

Table 10: 64 training domains that make up our multi-domain training corpus, with the batch that they appear in for our scaling study in §6. [†] indicates datasets we de-identify with regexes in Table 12. 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.

	Domain	Corpus
EVALUATION DOMAINS	IMDB	IMDB reviews (Maas et al., 2011)
	LEGAL CONTRACTS	500 commercial legal contracts (Hendrycks et al., 2021)
	CSCAREERQUESTIONS [†]	Reddit comments from pushshift.io, restricted to /r/cscareerquestions (Baumgartner et al., 2020)
	INDIA [†]	Reddit comments from pushshift.io, restricted to /r/india (Baumgartner et al., 2020)
	ENRON [†]	6M Yelp restaurant reviews (Yelp Reviews)
	HIPHOPHEADS [†]	Reddit comments from pushshift.io, restricted to /r/hiphopheads (Baumgartner et al., 2020)
	CONGRESSIONAL BILLS	Congressional bills from BillSum (Kornilova and Eidelman, 2019)
	IRELAND SPEECHES	Irish parliamentary speeches, 1919-2013 (Herzog and Mikhaylov, 2017)
	SQL	SQL code from Github [MIT, BSD, Apache 2.0] (Huggingface)
	RUST	Rust code from Github [MIT, BSD, Apache 2.0] (Huggingface)
	PERL	PERL code from Github [MIT, BSD, Apache 2.0] (Huggingface)
	TEX	TeX code from Github [MIT, BSD, Apache 2.0] (Huggingface)
	FORTTRAN	FORTTRAN code from Github [MIT, BSD, Apache 2.0] (Huggingface)
	COVID19 TWEETS [†]	Tweets with #COVID-19 hashtag (Twitter Academic API)
	TOEFL EXAM RESPONSES	TOEFL exam responses (Blanchard et al., 2013)
	BREAKING NEWS [†]	20K English news articles, scraped using Newspaper3K (Ou-Yang, Lucas)

Table 11: 32 domains that make up our novel domain corpus, 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 12. Meta researchers did not collect any of the Reddit or Twitter data and the data was not collected on behalf of Meta.

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

125M – 16 GPUs – 80k updates						
	T-LM	T-LM UNBALANCED	DEMIX	ELMFOREST (random init)	RANDOM ENSEMBLE	ELMFOREST (seed init)
	125M	125M	512M	1B	1B	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	T-LM UNBALANCED	DEMIX	ELMFOREST (random init)	RANDOM ENSEMBLE	ELMFOREST (seed init)
	350M	350M	1.8B	2.8B	2.8B	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	T-LM UNBALANCED	DEMIX	ELMFOREST (random init)	RANDOM ENSEMBLE	ELMFOREST (seed init)
	750M	750M	3.8B	6B	6B	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	T-LM UNBALANCED	DEMIX	ELMFOREST (random init)	RANDOM ENSEMBLE	ELMFOREST (seed init)
	1.3B	1.3B	7B	10.4B	10.4B	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 13: **ELMFORESTS trained with BTM outperform all baselines and ensemble variations across multiple model scales.** Average test-set perplexity (\downarrow) for each model scale (125M, 350M, 750M, 1.3B parameters) across the 8 training, 8 novel, and all 16 domains described in §4.1. 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 §4. For ELMFOREST, we show results with 50% dense training.

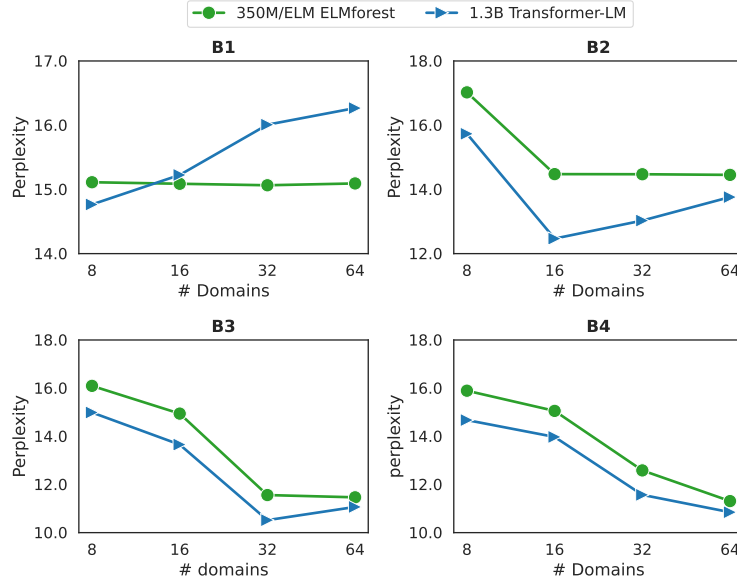


Figure 10: **As the number of training domains increase, ELMFORESTS retain or improve performance, while compute-matched TRANSFORMER-LMs degrade.** Average test set perplexity over each batch of training domains ($B_1 - B_4$). For the TRANSFORMER-LM experiment, we train a new TRANSFORMER-LM from scratch on 8 (B_1), 16 ($B_1 + B_2$), 32 ($B_1 + B_2 + B_3$), and 64 ($B_1 + B_2 + B_3 + B_4$) domains, for 6144 GPUs hours each. For the ELMFOREST experiment, we use BTM to train on the batches incrementally. We observe that TRANSFORMER-LMs suffer as one trains on more domains while ELMFORESTS retain or improve performance as domains are added incrementally. This reflects the "curse of multilinguality" phenomenon discovered for multilingual transformer LMs (Pfeiffer et al., 2022).