Efficient Online Data Mixing For Language Model Pre-Training

Alon Albalak[1] **Liangming Pan**[1] **Colin Raffel**[2,3] **William Yang Wang**[1]
[1]University of California, Santa Barbara
[2]University of Toronto
[3]Vector Institute

Abstract

The data used to pretrain large language models has a decisive impact on a model's downstream performance, which has led to a large body of work on data selection methods that aim to automatically determine the most suitable data to use for pretraining. Existing data selection methods suffer from slow and computationally expensive processes, a problem amplified by the increasing size of models and of pretraining datasets. Data mixing, on the other hand, reduces the complexity of data selection by grouping data points together and determining sampling probabilities across entire groups. However, data mixing proportions are typically fixed before training and therefore cannot adapt to changing training dynamics. To address these limitations, we develop an efficient algorithm for Online Data Mixing (**ODM**) that combines elements from both data selection and data mixing. Based on multi-armed bandit algorithms, our online approach optimizes the data mixing proportions during training. Remarkably, our method trains a model that reaches the final perplexity of the next best method with 19% fewer training iterations, and improves performance on the 5-shot MMLU benchmark by 1.9% relative accuracy, while adding negligible wall-clock time during pretraining.

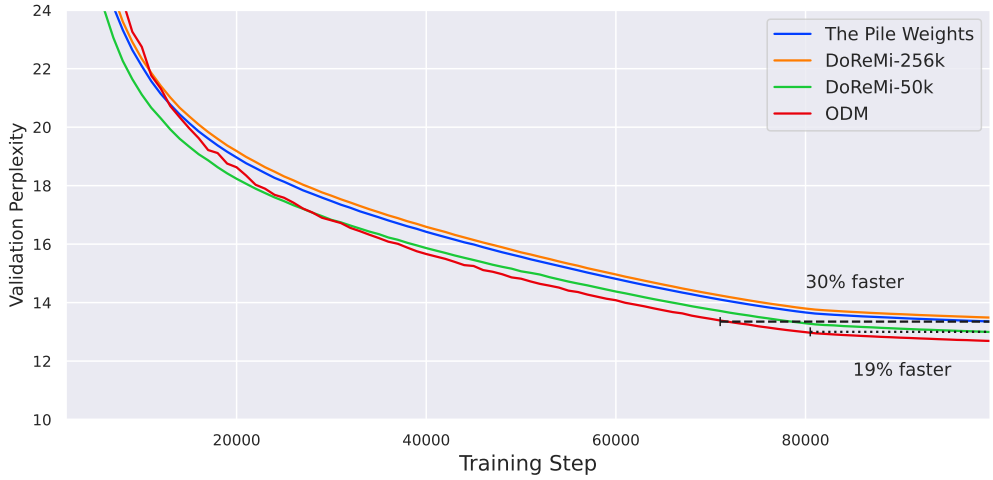

Figure 1: **Validation perplexity**, unweighted average over 22 domains from The Pile [1].

1 Introduction

It is well-known that the training data for machine learning models has a significant influence on their performance. In particular, the data used to pretrain large language models (LLMs) can be a major factor in the performance of a given LLM. For example, the 28 different 7-billion parameter models on the Open LLM Leaderboard[1] have scores varying from 34.92 to 56.26 even though they all use nearly the same model architecture and training process [2]. It is a widely accepted view that *pretraining is performed so that models can absorb large quantities of information* [3], and later training stages such as target task fine-tuning [4], instruction fine-tuning [5], and RLHF [6] primarily refine the model for a specific purpose. This perspective raises the important question of how best to choose pretraining data for training LLMs.

Language models are generally trained on data collected from a variety of domains in hopes that data diversity will lead to a higher-quality model, but the data mixing strategy to use (i.e. how frequently to sample data from each domain) during training is an open question. For example, when introducing The Pile [1] dataset (consisting of data from 22 domains), the authors suggest higher sampling weights on academic texts and those domains that they felt would provide high-quality data, but these weights are determined using intuition and heuristics, raising the question as to whether a more performant set of weights could be found. The recently proposed DoReMi algorithm [7] was specifically designed to automatically determine a data mixing strategy for LLM training. DoReMi optimizes domain weights that maximize the information gained of a "proxy" model over a "reference" model, but requires training multiple models, reducing the method's efficiency. Additionally, we show in this work that their sampling weights don't transfer well across models and thus requires training new "reference" and "proxy" models in order to determine the best weights for each new model architecture or tokenizer. These additional steps and considerations reduce the effective efficiency of DoReMi and further increase the already expensive cost of training large language models. Furthermore, both DoReMi and The Pile fix weights throughout training and therefore cannot adapt to changing dynamics over the course of pretraining.

In this work, we follow the principle that the best data to train on is the data that maximizes information gained and that a data selection method should introduce negligible computational overhead. Motivated by recent uses of multi-armed bandits (MAB) for auxiliary data selection in few-shot LLM fine-tuning [4], we view each data domain as the arm of an MAB and design an algorithm that optimizes the data mixing distribution in an online fashion, thereby adapting to changing training dynamics. Recalling from information theory that perplexity can be thought of as a measure of model uncertainty and the expected information gain from learning the next token, we aim to increase the mixing ratio for domains with the most information to be learned. We therefore utilize the training loss per domain as a reward for our multi-armed bandit algorithm, which fortuitously requires minimal overhead to compute.

To empirically validate the effectiveness and efficiency our approach, we perform language model pretraining using a 1-billion parameter model trained on 50 billion tokens from the 22 domains found in The Pile [1]. We compare our method with three baseline data mixing methods, finding that our online data mixing algorithm is the most effective, reaching the final validation perplexity of the next best method with 19% fewer iterations (Figure 1) and improving on 5-shot MMLU [8] performance by 3% relative accuracy over the baseline (Table 1). Additionally, we find that our method is computationally efficient, introducing a minuscule 0.000007% overhead.

2 Online Data Mixing (ODM)

In this section, we first define the setting under which online data mixing for language model pretraining takes place (outlined in Figure 2). Then, we formulate the online data mixing problem under the multi-armed bandit (**MAB**) setting, and describe our reward function which measures information gain and is very efficient to compute. Finally, we describe our algorithm for ODM and present pseudo-code in Algorithm 1.

Problem setup. Consider the setting where we are given K groups of data for language model pretraining, where each group $\mathcal{D}_i$ will be sampled according to the probability defined by $\pi(\mathcal{D}_i)$.

[1] Open LLM Leaderboard accessed on 10/02/2023, 28 models includes only pretrained models without fine-tuning, instruction-tuning, or RL-tuning.

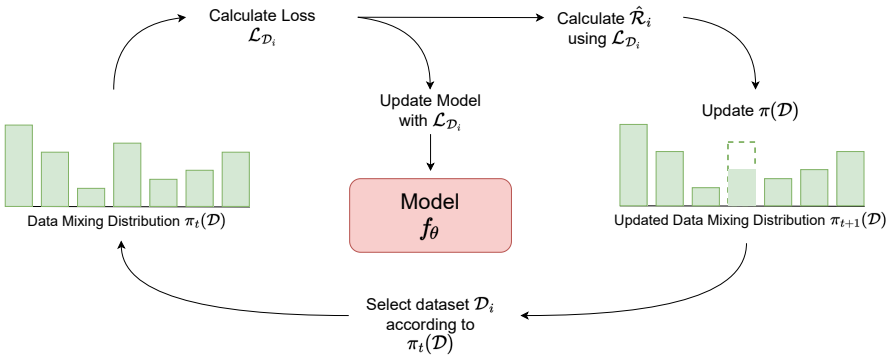

Figure 2: **Overview of Online Data Mixing (ODM) as a multi-armed bandit**. At each iteration of training, t, a dataset $\mathcal{D}_i$ is sampled according to the data mixing distribution π. The loss $\mathcal{L}_{\mathcal{D}_i}$ is calculated w.r.t the model f_θ and subsequently used to update the model. Simultaneously, a reward $\hat{\mathcal{R}}_i$ is calculated and used to update π for the next iteration, $i+1$.

Each group $\mathcal{D}_i$ could be assigned explicitly according to different domains as in The Pile [1], or they could determined via some automatic method (as e.g. in [9]). In traditional data mixing, each $\pi(\mathcal{D}_i)$ is fixed prior to training, but in online data mixing, we let each $\pi(\mathcal{D}_i)$ be redefined at every training iteration. Given that we want to update $\pi(\mathcal{D}_i)$ at every training iteration, the problem this work attempts to solve is how to update $\pi(\mathcal{D}_i)$ so that the information content of the data being trained on is maximized, and how to do so efficiently.

Adapting multi-armed bandits to data mixing. We adopt the multi-armed bandit (**MAB**) framework to attack the online data mixing problem by formulating it as a Markov decision process [10] that is played over N turns. We design our approach based on Exp3 (*Exp*onential-weight algorithm for *Exp*loration and *Exp*loitation) [11]. Exp3 defines the policy as a Gibbs distribution based on the empirically determined importance-weighted reward of dataset proportions [12] and allows for exploration by mixing the Gibbs distribution with a uniform distribution [11]. Let $\mathcal{E}_t$ represent the exploration rate at time step t, then the probability of selecting dataset $\mathcal{D}_i \in \mathcal{D}$ is defined by π as the linear combination of Gibbs and uniform distributions

$\pi_t(\mathcal{D}_i) = (1-K\mathcal{E}_t)\frac{\exp(\mathcal{E}_{t-1}\hat{R}_i)}{\sum_j \exp(\mathcal{E}_{t-1}\hat{R}_j)} + \mathcal{E}_t$ where $\hat{R}_{i,t}$ is the moving average of the importance weighted reward $\hat{R}_{i,t} = \alpha\hat{R}_{i,t-1} + (1-\alpha)\frac{R_{i,t}}{\pi_{t-1}(\mathcal{D}_i)}$. We adopt the decaying exploration rate from Seldin et al. [12], defined at turn t as $\mathcal{E}_t = \min\left\{\frac{1}{K}, \sqrt{\frac{\ln K}{K \cdot t}}\right\}$. The main deviation of our method from Exp3 is the use of a moving average estimated reward instead of a cumulative estimated reward. Under normal MAB settings, rewards at each turn are weighted equally, but in our setting we care most about recent rewards. Thus, we still account for past rewards through the use of a moving average, but rewards from the past are weighted less and less moving further into the past.

Designing the reward function. When designing our reward function we have 2 main goals: (1) ensure that the policy favors data with the highest information content, and (2) minimize the computation required. To achieve these goals, we define the reward to be the current loss for a given dataset group. Formally, at turn t, suppose that dataset $\mathcal{D}_i$ is sampled from $\pi(\mathcal{D})$, and a batch is sampled as $\{\mathbf{x}, \mathbf{y}\} \sim \mathcal{D}_i$. Then, the reward is simply $\mathcal{R}_{i,t} = \mathcal{L}(f, \mathbf{x}, \mathbf{y})$. By formulating the reward as the training loss on a dataset, we add no additional forward or backward passes through the model beyond standard training procedures, minimizing the computation required. Additionally, as discussed in section 1, perplexity (the exponentiated loss) is a measure of expected information gain from each token in a sequence. Thus, by assigning a high reward to datasets with high perplexity, we favor data with the highest information content.

Online data mixing algorithm. Our algorithm is shown in pseudocode in Algorithm 1 and runs as follows: At each turn, the exploration rate $\mathcal{E}_t$ is calculated and the policy π defines a sampling strategy over all K datasets $\mathcal{D}_i \in \mathcal{D}$. Since we are dealing with LLM pretraining which typically uses a large batch size, we assume that we will have G gradient accumulation steps. For each accumulation

Algorithm 1 Online Data Mixing (ODM)

Require: $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$: Grouped dataset
Require: f_θ: Parameterized model
Require: $\mathcal{L}$: Loss function
Require: G: Gradient accumulation steps

Initialize: $K = |\mathcal{D}|$; $\mathcal{E}_0 = \frac{1}{K}$; $\forall i \in \{1, \ldots, K\} : \hat{R}_i = 0$
for $t = 1, 2, \ldots, N$ **do**
  $\mathcal{E}_t = \min\left\{\frac{1}{K}, \sqrt{\frac{\ln K}{K \cdot t}}\right\}$ ▷ Update the exploration rate
  $\pi(\mathcal{D}) : \pi(\mathcal{D}_i) \leftarrow (1 - K\mathcal{E}_t)\frac{\exp(\mathcal{E}_{t-1}\hat{R}_i)}{\sum_j \exp(\mathcal{E}_{t-1}\hat{R}_j)} + \mathcal{E}_t$ ▷ Calculate the mixing distribution
  $\forall i = 1, 2, \ldots, K : \mathcal{L}_{\mathcal{D}_i} = 0$ ▷ Reset group losses
  for $g = 1, 2, \ldots, G$ **do**
    Sample $\mathcal{D}_i \sim \pi(\mathcal{D})$ and sample a batch $\{\mathbf{x}, \mathbf{y}\}$ from $\mathcal{D}_i$
    $\mathcal{L}_{\mathcal{D}_i} \leftarrow \mathcal{L}_{\mathcal{D}_i} + \mathcal{L}(f_\theta, \mathbf{x}, \mathbf{y})$ ▷ Record group losses for reward updates
  end for
  Update model parameters w.r.t $\sum_i \nabla_\theta \mathcal{L}_{\mathcal{D}_i}$
  for $i \in \{1, \ldots, K\}$ where $\mathcal{L}_{\mathcal{D}_i} \neq 0$ **do**
    $\hat{R}_i \leftarrow \alpha \hat{R}_i + (1 - \alpha)\frac{\mathcal{L}_{\mathcal{D}_i}}{\pi(\mathcal{D}_i)}$ ▷ Update estimated rewards
  end for
end for

step we sample one of the datasets $\mathcal{D}_i$, then sample a batch $\{\mathbf{x}, \mathbf{y}\} \sim \mathcal{D}_i$ and calculate the loss $\mathcal{L}_{\mathcal{D}_i}$. After accumulating losses, we calculate the gradient w.r.t. θ and update the model. Finally, for each sampled dataset $\mathcal{D}_i$, we calculate a reward $\mathcal{R}_i$ that is used to update the policy π for the next turn. As a practical method to reduce the very high variance of losses at the beginning of language model training, we include a warmup period during which the model trains, but the policy remains stationary. In practice, we find a warmup period of 1% of total steps to be sufficient.

3 Experimental Setup

Training. For our experiments we use The Pile [1], an 825Gb open-sourced language modelling dataset comprising 22 smaller datasets from various domains including Wikipedia, Github, and PubMed Central. We train decoder-only style transformers using an adapted version of the GPT-NeoX library [13]. For all experiments, we train a 1 billion parameter model using the model configuration of Pythia [14]. We train using a batch size of 60 sequences per GPU, and accumulate gradients across 8 GPUs in parallel ($G = 8$) to reach a total batch size of 480 samples. We let the sequence length be 1024 and pack sequences together [15]. We train for a total of 100,000 steps, reaching 50 billion tokens. For ODM, we initialize the domain weights using those defined by The Pile. The full model configuration and hyperparameters can be found in Appendix A.

Evaluation. To validate the performance of our approach and the baselines, we compute perplexity on held-out validation and test data from each domain of The Pile. Additionally, we evaluate each model on downstream capabilities by performing multiple choice classification on the 57 tasks from MMLU [8]. For each task in MMLU we use 5 in-context examples.

Baselines. We compare the performance of our method against that of the original domain weights suggested by The Pile [1], and refer to it as The Pile Weights (**TPW**). Additionally, we compare with the domain weights proposed by DoReMi [7], but empirically find that the weights do not perform as published. However, after discussion with the authors, we attained weights that were re-calculated on the same tokenizer as ours[2]. The original DoReMi weights are computed with a 256k vocabulary tokenizer while we use a 50k vocabulary tokenizer, so to specify each DoReMi baseline we name them **DoReMi-256k** and **DoReMi-50k**.

[2]It is hypothesized by the authors of [7] that different tokenizers may lead to different domain weights, but is still an open question why that may be the case.

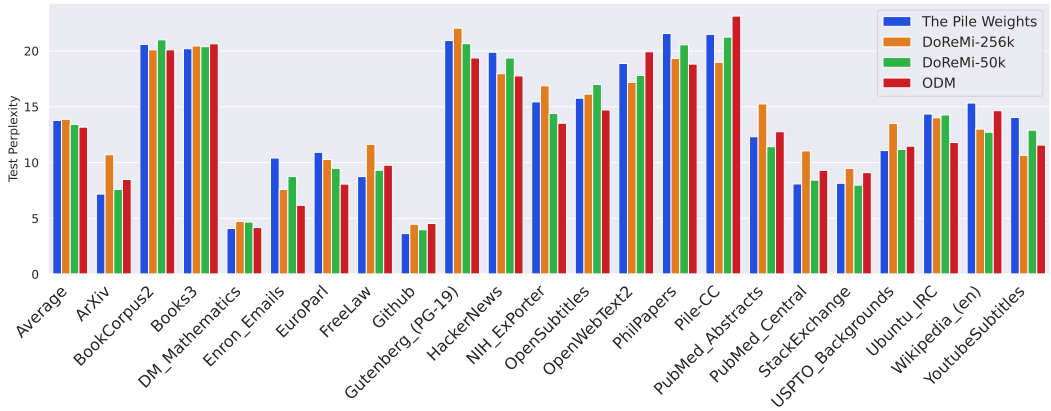

Figure 3: **Test perplexity** on average, and on 22 individual domains.

4 Findings and analysis.

In Figures 1 and 3 we compare the perplexities of training models using ODM with the baseline data mixing methods. Table 1 shows the average 5-shot accuracy on MMLU of ODM and baseline methods.

Main results. Figure 1 shows that ODM achieves the final performance of the originally suggested Pile weights (TPW) with 30% fewer iterations, and 19% fewer than DoReMi-50k. Additionally, Figure 1 shows that ODM's final validation perplexity is 4.8% lower than TPW, 2.4% lower than DoReMi-50k, and 4.9% lower than DoReMi-256k, emphasizing how the DoReMi method is not transferrable across models. These results show that ODM improves the training efficiency compared with static data mixing methods. Additionally, Table 1 shows that ODM leads to better downstream performance in 5-shot classification tasks, improving over TPW by 3%, and DoReMi-50k by 1.9%.

Figure 3 shows the test perplexity of each method on held-out data as well as the average perplexity. Surprisingly, we find that the original domain weights reported for DoReMi [7] (DoReMi-256k) leads to test perplexity that is, on average, 0.7% worse than The Pile Weights, in direct contradiction with their original findings. However, DoReMi-50k does improve over The Pile Weights by 2.6%, demonstrating that the domain weights determined by the DoReMi method do not transfer well across models.

Method	Accuracy
The Pile Weights	0.27469
DoReMi-256k	0.27596
DoReMi-50k	0.27887
ODM	0.28416

Table 1: **Average 5-shot accuracy on MMLU**

The effects of data mixing optimization objectives on individual domain performance. Here we compare the empirical effects of the contrasting optimization of objectives of ODM and DoReMi on individual domains. Recall that the reward function used in ODM favors dataset groups with the greatest information gain (highest loss) at each step, and that DoReMi's objective is to maximize the information gain of a "proxy" model over a "reference" model (i.e. "minimize the worst-case excess loss"). To see these different objectives in effect, we group the performance of each method into one of three buckets: best, worst, or in the middle, where the ideal method would have all 22 domains in the "best" category. Interestingly, we find that The Pile Weights are almost evenly distributed across all 3 buckets, doing worst in 7 domains, best in 7, and in the middle for the remaining 8. As expected from a method that optimizes for the best worst-case scenario, we find that DoReMi-50k's test perplexity is often not the best or the worst, but falls in the middle. In fact, 17/22 domains are in the middle, only performing best on three domains (PubMed_Abstracts, StackExchange, and Wikipedia_(en)), and worst on only two domains (BookCorpus2 and OpenSubtitles). On the other hand, using ODM leads to the best perplexity on 9 domains, with 9 more in the middle, and only performing the worst on 4 domains (Books3, Github, OpenWebText2, and Pile-CC). Notably, two of

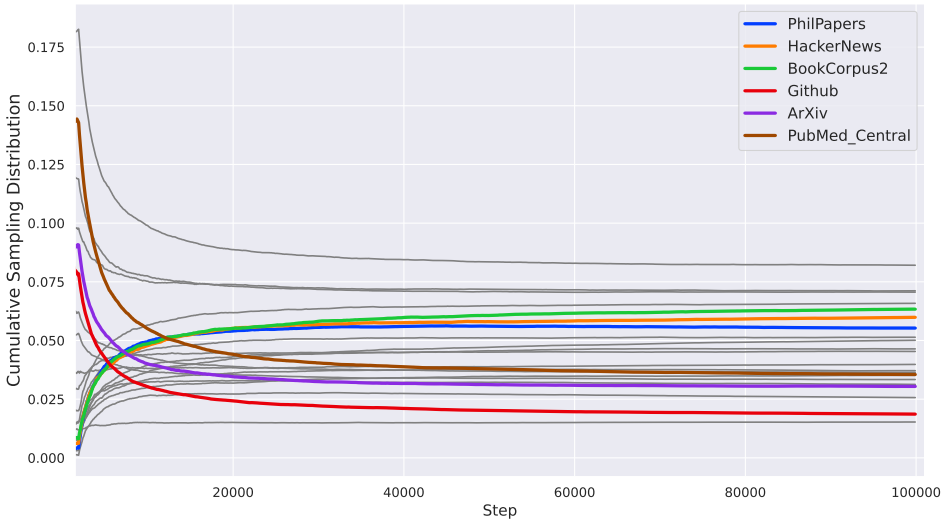

Figure 4: **The cumulative sampling distribution of ODM** calculated as the samples per domain out of the total number of samples trained on. Highlighted lines are the six domains whose final sampling distributions have increased/decreased the most from initialization.

the domains where ODM performs worst are web text domains but this decreased performance does not seem to have a negative impact on downstream performance.

What does ODM's sampling policy look like? In Figure 4 we show the cumulative sampling distribution of each domain over the course of training. Note that ODM is initialized with The Pile Weights, which are the initial values on the left. Figure 4 highlights the three datasets whose mixing ratio increased the most (PhilPapers, HackerNews, and BookCorpus2), and the three datasets whose mixing ratio decreased the most (Github, ArXiv, and PubMed_Central). It is evident from this figure that ODM finds a sampling distribution which is closer to uniform than The Pile Weights. We also see that the distribution for most domains stabilizes early on in training (~ 40000 iterations). Beyond the 40000 step, the distribution is still changing, but at a much lower rate. For example, we see that the mixing ratio for Github is still decreasing and the ratio for both BookCorpus2 and HackerNews are increasing all the way until the end of training.

Why does ODM's validation perplexity start off high? Figure 1 shows that although our method outperforms the baselines, at the beginning of training ODM actually has higher perplexity than other methods. We believe that this is due to the homogeneity of the micro-batches used in ODM, whereas other methods see a greater mixture of data in each batch. In preliminary experiments we trialed a version of ODM that uses data from a single domain in all gradient update steps, and found that this exacerbates the phenomena leading to a perplexity that starts even higher. This suggests that one of the weaknesses of our method is the requirement that each batch comes from the same grouped dataset. This problem can be alleviated by decreasing the micro-batch size, but this comes with technical considerations as simply decreasing micro-batch size will reduce GPU utilization, and lead to slower wall clock time. Likely, a better solution would be to mix domains within micro-batches during the warm-up phase, which would lead to validation perplexity exactly the same as The Pile Weights, but gaining the advantages of ODM after the warm-up.

5 Conclusion

The method proposed in this work demonstrates the effectiveness of online data mixing formulated as a multi-armed bandit problem. Additionally, we showed that by designing a reward which attempts to maximize information gain, we can train models that achieve lower perplexity on held-out data in fewer iterations than baseline methods. Furthermore, by utilizing the training loss as a reward, the proposed method is very computationally efficient, adding a trivial 0.000007% additional wall-clock time to training.

References

[1] 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. The pile: An 800gb dataset of diverse text for language modeling, 2020.

[2] Nathan Habib Sheon Han Nathan Lambert Nazneen Rajani Omar Sanseviero Lewis Tunstall Thomas Wolf Edward Beeching, Clémentine Fourrier. Open llm leaderboard. `https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard`, 2023.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[4] Alon Albalak, Colin Raffel, and William Yang Wang. Improving few-shot generalization by exploring and exploiting auxiliary data, 2023.

[5] Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022.

[6] Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences, 2020.

[7] Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy Liang, Quoc V. Le, Tengyu Ma, and Adams Wei Yu. Doremi: Optimizing data mixtures speeds up language model pretraining, 2023.

[8] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[9] Suchin Gururangan, Margaret Li, Mike Lewis, Weijia Shi, Tim Althoff, Noah A. Smith, and Luke Zettlemoyer. Scaling expert language models with unsupervised domain discovery, 2023.

[10] RICHARD BELLMAN. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[11] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multiarmed bandit problem. *SIAM journal on computing*, 32(1):48–77, 2002.

[12] Yevgeny Seldin, Csaba Szepesvári, Peter Auer, and Yasin Abbasi-Yadkori. Evaluation and analysis of the performance of the exp3 algorithm in stochastic environments. In Marc Peter Deisenroth, Csaba Szepesvári, and Jan Peters, editors, *Proceedings of the Tenth European Workshop on Reinforcement Learning*, volume 24 of *Proceedings of Machine Learning Research*, pages 103–116, Edinburgh, Scotland, 30 Jun–01 Jul 2013. PMLR.

[13] Alex Andonian, Quentin Anthony, Stella Biderman, Sid Black, Preetham Gali, Leo Gao, Eric Hallahan, Josh Levy-Kramer, Connor Leahy, Lucas Nestler, Kip Parker, Michael Pieler, Shivanshu Purohit, Tri Songz, Wang Phil, and Samuel Weinbach. GPT-NeoX: Large Scale Autoregressive Language Modeling in PyTorch, 8 2021.

[14] Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. Pythia: A suite for analyzing large language models across training and scaling, 2023.

[15] Adam Roberts, Hyung Won Chung, Anselm Levskaya, Gaurav Mishra, James Bradbury, Daniel Andor, Sharan Narang, Brian Lester, Colin Gaffney, Afroz Mohiuddin, Curtis Hawthorne, Aitor Lewkowycz, Alex Salcianu, Marc van Zee, Jacob Austin, Sebastian Goodman, Livio Baldini Soares, Haitang Hu, Sasha Tsvyashchenko, Aakanksha Chowdhery, Jasmijn Bastings, Jannis Bulian, Xavier Garcia, Jianmo Ni, Andrew Chen, Kathleen Kenealy, Jonathan H. Clark, Stephan Lee, Dan Garrette, James Lee-Thorp, Colin Raffel, Noam Shazeer, Marvin Ritter, Maarten Bosma, Alexandre Passos, Jeremy Maitin-Shepard, Noah Fiedel, Mark Omernick, Brennan Saeta, Ryan Sepassi, Alexander Spiridonov, Joshua Newlan, and Andrea Gesmundo. Scaling up models and data with `t5x` and `seqio`, 2022.

[16] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2022.

[17] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022.

[18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[19] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. GPT-NeoX-20B: An open-source autoregressive language model. In *Proceedings of the ACL Workshop on Challenges & Perspectives in Creating Large Language Models*, 2022.

A Model Configuration

Our 1-billion parameter model uses a sequence length of 1024, has 16 layers with a hidden size of 2048, 16 attention heads, and rotary positional embeddings [16]. We use FlashAttention [17] to reduce training time. We use the Adam optimizer [18] with a linear warmup over 1000 iterations from a minimum learning rate of 2.5e-5 to a maximum learning rate of 2.5e-4, and then decay the learning rate with a cosine schedule down to the minimum of 2.5e-5 again. We use the GPT-NeoX-20B tokenizer [19].