Sheared LLaMA: Accelerating Language Model Pre-training via Structured Pruning

Mengzhou Xia[1], **Tianyu Gao**[1], **Zhiyuan Zeng**[2], **Danqi Chen**[1]
[1]Department of Computer Science & Princeton Language and Intelligence, Princeton University
[2]Department of Computer Science and Technology, Tsinghua University
{mengzhou,tianyug,danqic}@cs.princeton.edu

Abstract

The popularity of LLaMA [58, 59] and other recently emerged moderate-sized large language models (LLMs) highlights the potential of building smaller yet powerful LLMs. Regardless, the cost of training such models from scratch on trillions of tokens remains high. In this work, we study structured pruning as an effective means to develop smaller LLMs from pre-trained, larger models. Our approach employs two key techniques: (1) *targeted structured pruning*, which prunes a larger model to a specified target shape by removing layers, heads, intermediate and hidden dimensions in an end-to-end manner, and (2) *dynamic batch loading*, which dynamically updates the composition of sampled data in each training batch based on varying losses across different domains. We demonstrate the efficacy of our approach by presenting the **Sheared-LLaMA** series, pruning the LLaMA2-7B model down to 1.3B and 2.7B parameters. Sheared-LLaMA models outperform state-of-the-art open-source models of equivalent sizes, such as Pythia, INCITE, and OpenLLaMA models, on a wide range of downstream and instruction tuning evaluations, while requiring less than 3% of compute compared to training such models from scratch. This work provides compelling evidence that leveraging existing LLMs with structured pruning is a far more cost-effective approach for building smaller LLMs.

1 Introduction

Large language models (LLMs) are extremely performant on a wide range of natural language tasks, but they require enormous amounts of compute to train [44, 2]. As such, there is growing interest in building strong moderate-sized models, such as LLaMA [58, 59], MPT [43], and Falcon [1], that allow for efficient inference and fine-tuning. These LLMs are available in varied sizes suited for different use cases, but training each individual model from scratch—even the smallest billion-parameter models—requires substantial computational resources that are cost-prohibitive for most organizations. In this work, we seek to address the following question:

Can we produce a smaller, general-purpose, and competitive LLM by leveraging existing pre-trained LLMs, while using much less compute than training one from scratch?

We explore structured pruning as a means to achieve this goal. Pruning is commonly viewed as a solution for compressing task-specific models [23, 33, 66, 31], removing redundant parameters and accelerating inference without sacrificing task performance. However, for general-purpose LLMs, pruning inevitably results in performance degradation compared to original models [16, 53, 41], especially when without significant compute invested post-pruning. In this work, we propose pruning as an effective approach for developing smaller yet competitive LLMs that require only a fraction of the compute compared to training them from scratch.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

We identify two key technical challenges in this problem. First, how can we decide on final pruned architectures that are strong in performance and efficient for inference? Existing structured pruning techniques for LLMs [66, 41] do not specify targeted structures and lead to suboptimal pruned models in terms of performance and inference speed (Table 5 and Figure 8). Second, how can we continue pre-training the pruned model to reach desired performance? We observe that training using the original pre-training data leads to imbalanced rates of loss reduction across different domains, compared to a trained-from-scratch model. This indicates that the pruned model retains varying levels of knowledge for different domains (e.g., GitHub vs. C4) and simply using the pre-training domain proportion results in an inefficient use of data (Figure 6). To address these issues, we propose a "shearing" algorithm consisting of the following two components:

- We propose a novel pruning method, dubbed *targeted structured pruning*, which prunes a source model to a specified target architecture. The target architecture is determined by leveraging the configurations of existing pre-trained models. Our pruning approach searches for substructures within the source model that maximally preserve performance while adhering to the given constraints.
- We devise a *dynamic batch loading* algorithm that loads training data from each domain in proportion to its rate of loss reduction, thereby making the data use more efficient and accelerating overall performance improvement.

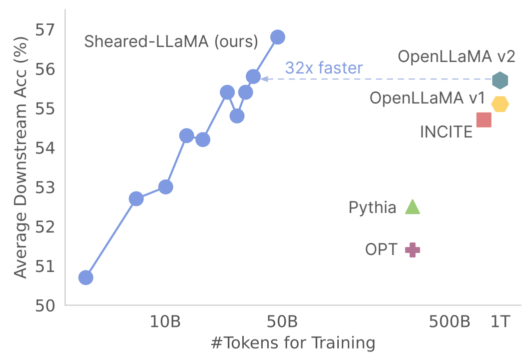

Figure 1: Our Sheared-LLaMA-2.7B surpasses a series of open-source models at a similar scale and only requires 1/32 of training tokens to achieve on-par performance with OpenLLaMA-3B-v2.

We demonstrate the efficacy of our proposed method by pruning a LLaMA2-7B model [59] into two smaller LLMs: Sheared-LLaMA-1.3B and Sheared-LLaMA-3B. Despite using only 50 billion tokens (i.e., 5% of OpenLLaMA's pre-training budget) for pruning and continued pre-training, Sheared-LLaMA-1.3B and Sheared-LLaMA-2.7B outperform other popular LLMs at similar scales, including Pythia [4], INCITE [56], and OpenLLaMA [19], on 11 representative downstream tasks (Figure 1; commonsense, reading comprehension, and world knowledge) and instruction tuning for open-ended generation. Furthermore, the trajectory implies that training the pruned model with more tokens into it will lead to even better performance. While we only conduct experiments with up to 7B parameter models, our shearing algorithm is highly generalizable and can be extended to large language models of any size in future work.

2 LLM Shearing Algorithm

Given an existing large model $\mathcal{M}_S$ (the *source* model), we study how to efficiently produce a smaller, strong model $\mathcal{M}_T$ (the *target* model). We consider this as a two-stage process: (1) Pruning $\mathcal{M}_S$ into $\mathcal{M}_T$. This reduces the number of parameters but incurs a performance drop inevitably. (2) Continually pre-training $\mathcal{M}_T$ with a standard language modeling objective to reach a target performance. Please refer to Appendix A for a full description of the method.

2.1 Targeted Structured Pruning

Structured pruning removes groups of model parameters to compress models and accelerate inference. In this work, we allow pruning the source model into configurations of existing pre-trained models as the target architecture, based on the intuition that these configurations have already been well-optimized to balance model expressivity and inference efficiency. For example, we use the INCITE-3B architecture [57] as the target when producing a 2.7B model.

Our method learns a set of pruning masks on model parameters at different granularity—from global ones like layers and hidden dimensions (persist across all layers), to local ones like attention heads and intermediate dimensions. Assume that the source model $\mathcal{M}_S$ has L_S layers, with each layer consisting of one multi-head attention module (MHA) and one feed-forward network (FFN). $\mathcal{M}_S$

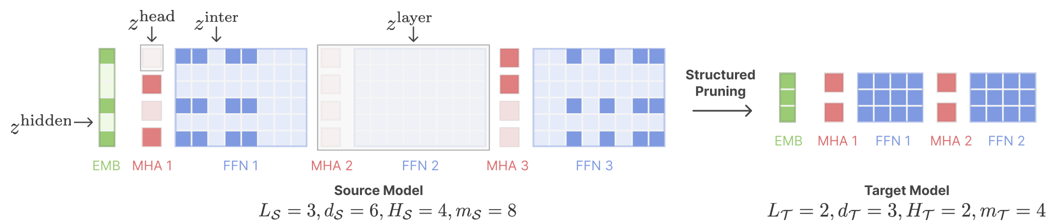

Figure 2: An illustration of *targeted structured pruning*, where we prune the model to a specified target structure. Light colors indicate pruned components.

has a hidden state dimension of $d_{\mathcal{S}}$, $H_{\mathcal{S}}$ heads in each MHA, and an intermediate dimension of $m_{\mathcal{S}}$ in each FFN. Each mask variable controls whether the associated structure is pruned or retained. For example, we remove a layer if its corresponding $z^{\text{layer}} = 0$. Figure 2 illustrates an example of how the pruning masks control the pruned structures.

We formulate pruning as a constrained optimization problem where we learn pruning masks to search for a subnetwork matching a pre-specified target architecture while maximizing performance.

2.2 Dynamic Batch Loading

Continued pre-training on a large amount of data is crucial for recovering the pruned model performance. However, we observe a surprising finding in our preliminary experiments: continuing pre-training our pruned models on the pre-training dataset RedPajama (56; LLaMA's pre-training dataset) reduces loss at different rates across domains compared to a model trained from scratch with the same data, which signifies an inefficient use of data. Inspired by [68], a recent work of reweighting data of different domains, we propose *dynamic batch loading*, a more efficient algorithm to simply adjust domain proportions on the fly based on the model performance. The goal is to ensure the model achieves the reference loss at a similar speed across all domains. We introduce the algorithm below.

Problem setup. The pre-training data comprises of k domains $D_1, D_2, \cdots, D_k$ and we have a held-out validation dataset for each domain, denoted as D_i^{val}. At each training step t, a proportion $w_t[i]$ of the data comes from domain D_i. We set a reference validation loss $\ell_{\text{ref}}(D_i)$ for each domain and train the pruned model to reach the reference loss.

Dynamic batch loading. We present the full algorithm in Algorithm 1. In a sketch, for every m steps, we evaluate the model to get the validation loss ℓ_t (step t) on D^{val}, and update w_t based on the difference $\Delta_t(D_i)$ between $\ell_{\text{ref}}[i]$ and $\ell_t[i]$ on each domain. The update rule is exponential ascent following [68] and we apply it to both the pruning and continued pre-training stage,

$$\alpha_t = \log(w_{t-m}) + \Delta_t; \quad w_t = \frac{\exp(\alpha_t)}{\sum_i \exp(\alpha_t[i])}.$$

3 Experiments

Setup We use the LLaMA2-7B model [59] as the source model throughout all of our main experiments.[1] We then conduct structured pruning experiments to compress this model down to two smaller target sizes—2.7B and 1.3B parameters. We compare to strong pre-trained language models of similar sizes, including OPT-1.3B [71], Pythia-1.4B [4], OPT-2.7B, Pythia-2.8B, INCITE-Base-3B [56], OpenLLaMA-3B-v1, and OpenLLaMA-3B-v2 [19]. Table 3 and Table 10 summarize model architecture and pre-training data of all these models. We test on a series of downstream tasks and and long-form instruction following tasks. Please find more experiment details in Appendix C.1.

Sheared-LLaMA Outperforms LMs of Equivalent Sizes We demonstrate, on both standard LM benchmarks and instruction tuning, Sheared-LLaMA significantly outperforms existing LLMs of similar sizes, while using only a fraction of the compute budget to train those models from scratch. In Table 1, we present the zero-shot and few-shot downstream task performance of both Sheared-LLaMA and existing pre-trained models of a similar size. Our experiments show that, even with a budget

[1]Please find results on LLaMA1 models in Appendix D.4.

Table 1: Sheared-LLaMA outperforms publicly available models of comparable size on downstream tasks. The shot number used is noted in parentheses, with 0-shot if not specified. Models with † use a different training data from RedPajama. Please refer to Table 3 for details.

	Commonsense & Reading Comprehension					
Model (#tokens for training)	**SciQ**	**PIQA**	**WinoGrande**	**ARC-E**	**ARC-C (25)**	**HellaSwag (10)**
LLaMA2-7B (2T)†	93.7	78.1	69.3	76.4	53.0	78.6
OPT-1.3B (300B)†	84.3	71.7	**59.6**	57.0	29.7	54.5
Pythia-1.4B (300B)†	86.4	70.9	57.4	60.7	31.2	53.0
Sheared-LLaMA-1.3B (50B)	**87.3**	**73.4**	57.9	**61.5**	**33.5**	**60.7**
OPT-2.7B (300B)†	85.8	73.7	60.8	60.8	34.0	61.5
Pythia-2.8B (300B)†	88.3	74.0	59.7	64.4	36.4	60.8
INCITE-Base-3B (800B)	90.7	74.6	63.5	**67.7**	40.2	64.8
Open-LLaMA-3B-v1 (1T)	91.3	73.7	61.5	67.6	39.6	62.6
Open-LLaMA-3B-v2 (1T)†	**91.8**	**76.2**	63.5	66.5	39.0	67.6
Sheared-LLaMA-2.7B (50B)	90.8	75.8	**64.2**	67.0	**41.2**	**70.8**

	Continued		LM	World Knowledge		
Model (#tokens for training)	**LogiQA**	**BoolQ (32)**	**LAMBADA**	**NQ (32)**	**MMLU (5)**	**Average**
LLaMA2-7B (2T)†	30.7	82.1	28.8	73.9	46.6	64.6
OPT-1.3B (300B)†	**26.9**	57.5	58.0	6.9	24.7	48.2
Pythia-1.4B (300B)†	27.3	57.4	**61.6**	6.2	**25.7**	48.9
Sheared-LLaMA-1.3B (50B)	**26.9**	**64.0**	61.0	**9.6**	**25.7**	**51.0**
OPT-2.7B (300B)†	26.0	63.4	63.6	10.1	25.9	51.4
Pythia-2.8B (300B)†	28.0	66.0	64.7	9.0	26.9	52.5
INCITE-3B (800B)	27.7	65.9	65.3	14.9	**27.0**	54.7
Open-LLaMA-3B-v1 (1T)	28.4	70.0	65.4	**18.6**	**27.0**	55.1
Open-LLaMA-3B-v2 (1T)†	28.1	69.6	66.5	17.1	26.9	55.7
Sheared-LLaMA-2.7B (50B)	**28.9**	**73.7**	**68.4**	16.5	26.4	**56.7**

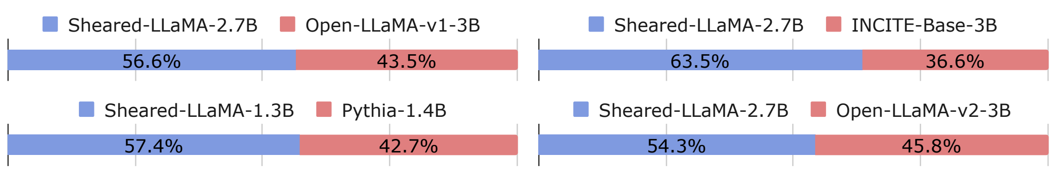

Figure 3: Sheared-LLaMAs outperform Pythia-1.4B, INCITE-Base-3B, OpenLLaMA-3B-v1 and OpenLLaMA-3B-v2 in instruction tuning.

as limited as approximately 50B tokens for pruning and continued pre-training, Sheared-LLaMA models outperform existing models that have been pre-trained on significantly larger compute. To elaborate further, Sheared-LLaMA-1.3B outperforms both the OPT-1.3B and Pythia-1.4B models, which were originally pre-trained with 300B tokens. Similarly, Sheared-LLaMA-2.7B outperforms INCITE-3B and OpenLLaMA-3B-v1, where were pre-trained on 800B and 1T RedPajama tokens respectively; Sheared-LLaMA-2.7B also surpasses OpenLLaMA-3B-v2, which was pre-trained on 1T tokens from a mixture of RedPajama, RefinedWeb, and StarCoder. As shown Figure 3, instruction-tuned Sheared-LLaMA achieves higher win rates compared to all the other pre-trained models at a comparable scale. This demonstrates that our 2.7B model can serve as a strong foundation for instruction tuning and has the capacity to generate long, coherent and informative responses (See examples in Appendix G).

4 Conclusion

In this work, we propose using structured pruning as an efficient way to produce competitive small LLMs. We produce a series of competitive Sheared-LLaMA models with a small amount of compute compared to standard pre-training. As more powerful LLMs and larger pre-training datasets become available, our approach can readily be applied to produce stronger small models.

References

[1] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Merouane Debbah, Etienne Goffinet, Daniel Heslow, Julien Launay, Quentin Malartic, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. Falcon-40B: an open large language model with state-of-the-art performance. 2023.

[2] Anthropic. Introducing claude, 2023.

[3] Jason Baumgartner, Savvas Zannettou, Brian Keegan, Megan Squire, and Jeremy Blackburn. The pushshift reddit dataset. *ArXiv*, abs/2001.08435, 2020.

[4] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR, 2023.

[5] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

[6] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2019.

[7] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained bert networks. In *Advances in Neural Information Processing Systems*, 2020.

[8] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, et al. Symbolic discovery of optimization algorithms. *arXiv preprint arXiv:2302.06675*, 2023.

[9] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[10] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[11] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020.

[12] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

[13] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

[14] Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Alpacafarm: A simulation framework for methods that learn from human feedback. *arXiv preprint arXiv:2305.14387*, 2023.

[15] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2018.

[16] Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot, 2023. *arXiv preprint arXiv:2301.00774*, 2023.

[17] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[18] Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, September 2021.

[19] Xinyang Geng and Hao Liu. Openllama: An open reproduction of llama, May 2023.

[20] Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tieyan Liu. Efficient training of bert by progressively stacking. In *International conference on machine learning*, pages 2337–2346. PMLR, 2019.

[21] Kshitij Gupta, Benjamin Thérien, Adam Ibrahim, Mats L Richter, Quentin Anthony, Eugene Belilovsky, Irina Rish, and Timothée Lesort. Continual pre-training of large language models: How to (re) warm your model? *arXiv preprint arXiv:2308.04014*, 2023.

[22] Felix Hamborg, Norman Meuschke, Corinna Breitinger, and Bela Gipp. news-please: A generic news crawler and extractor. In *Proceedings of the 15th International Symposium of Information Science*, pages 218–223, 2017.

[23] Song Han, Huizi Mao, Dally, and William Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*, 2016.

[24] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

[25] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[26] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[27] Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. Dynabert: Dynamic bert with adaptive width and depth. *Advances in Neural Information Processing Systems*, 33:9782–9793, 2020.

[28] Angela H Jiang, Daniel L-K Wong, Giulio Zhou, David G Andersen, Jeffrey Dean, Gregory R Ganger, Gauri Joshi, Michael Kaminksy, Michael Kozuch, Zachary C Lipton, et al. Accelerating deep learning by focusing on the biggest losers. *arXiv preprint arXiv:1910.00762*, 2019.

[29] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, 2020.

[30] Jean Kaddour, Oscar Key, Piotr Nawrot, Pasquale Minervini, and Matt J Kusner. No train no gain: Revisiting efficient training algorithms for transformer-based language models. *arXiv preprint arXiv:2307.06440*, 2023.

[31] Eldar Kurtic, Elias Frantar, and Dan Alistarh. Ziplm: Hardware-aware structured pruning of language models. *arXiv preprint arXiv:2302.04089*, 2023.

[32] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019.

[33] François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M Rush. Block pruning for faster transformers. *arXiv preprint arXiv:2109.04838*, 2021.

[34] Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. Deduplicating training data makes language models better. *arXiv preprint arXiv:2107.06499*, 2021.

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

[36] Hong Liu, Zhiyuan Li, David Hall, Percy Liang, and Tengyu Ma. Sophia: A scalable stochastic second-order optimizer for language model pre-training. *arXiv preprint arXiv:2305.14342*, 2023.

[37] Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 3622–3628, 2020.

[38] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744, 2017.

[39] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018.

[40] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[41] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*, 2023.

[42] Sören Mindermann, Jan M Brauner, Muhammed T Razzak, Mrinank Sharma, Andreas Kirsch, Winnie Xu, Benedikt Höltgen, Aidan N Gomez, Adrien Morisot, Sebastian Farquhar, et al. Prioritized training on points that are learnable, worth learning, and not yet learnt. In *International Conference on Machine Learning*, pages 15630–15649. PMLR, 2022.

[43] MosaicML. Introducing mpt-7b: A new standard for open-source, commercially usable llms, 2023. Accessed: 2023-05-05.

[44] OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023.

[45] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35, 2022.

[46] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534, 2016.

[47] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The refinedweb dataset for falcon llm: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*, 2023.

[48] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

[49] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

[50] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[51] Victor Sanh, Thomas Wolf, and Alexander Rush. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in Neural Information Processing Systems*, 33:20378–20389, 2020.

[52] Noam M. Shazeer. Glu variants improve transformer. *ArXiv*, abs/2002.05202, 2020.

[53] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

[54] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2158–2170, 2020.

[55] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

[56] TogetherAI. Redpajama: An open source recipe to reproduce llama training dataset, 2023.

[57] TogetherAI. Redpajama-incite-base-3b-v1, 2023.

[58] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[59] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[60] Trieu H. Trinh and Quoc V. Le. A simple method for commonsense reasoning. *ArXiv*, abs/1806.02847, 2018.

[61] Peiyi Wang, Lei Li, Liang Chen, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. Large language models are not fair evaluators. *arXiv preprint arXiv:2305.17926*, 2023.

[62] Ziheng Wang, Jeremy Wohlwend, and Tao Lei. Structured pruning of large language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6151–6162, 2020.

[63] Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 94–106, 2017.

[64] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

[65] Mengzhou Xia, Mikel Artetxe, Chunting Zhou, Xi Victoria Lin, Ramakanth Pasunuru, Danqi Chen, Luke Zettlemoyer, and Veselin Stoyanov. Training trajectories of language models across scales. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13711–13738, Toronto, Canada, July 2023. Association for Computational Linguistics.

[66] Mengzhou Xia, Zexuan Zhong, and Danqi Chen. Structured pruning learns compact and accurate models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1513–1528, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[67] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023.

[68] 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. *arXiv preprint arXiv:2305.10429*, 2023.

[69] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, 2019.

[70] Minjia Zhang and Yuxiong He. Accelerating training of transformer-based language models with progressive layer dropping. *Advances in Neural Information Processing Systems*, 33:14011–14023, 2020.

[71] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[72] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.

[73] Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 19–27, 2015.

A Method

A.1 Targeted Structured Pruning

Our method learns a set of pruning masks on model parameters at different granularity—from global ones like layers and hidden dimensions (persist across all layers), to local ones like attention heads and intermediate dimensions. Assume that the source model $\mathcal{M}_{\mathcal{S}}$ has $L_{\mathcal{S}}$ layers, with each layer consisting of one multi-head attention module (MHA) and one feed-forward network (FFN). $\mathcal{M}_{\mathcal{S}}$ has a hidden state dimension of $d_{\mathcal{S}}$, $H_{\mathcal{S}}$ heads in each MHA, and an intermediate dimension of $m_{\mathcal{S}}$ in each FFN.

Granularity	Layer	Hidden dimension	Head	Intermediate dimension
Pruning masks	$z^{\text{layer}} \in \mathbb{R}^{L_{\mathcal{S}}}$	$z^{\text{hidden}} \in \mathbb{R}^{d_{\mathcal{S}}}$	$z^{\text{head}} \in \mathbb{R}^{H_{\mathcal{S}}}$ $(\times L_{\mathcal{S}})$	$z^{\text{int}} \in \mathbb{R}^{m_{\mathcal{S}}}$ $(\times L_{\mathcal{S}})$

Each mask variable controls whether the associated structure is pruned or retained. For example, we remove a layer if its corresponding $z^{\text{layer}} = 0$. Figure 2 illustrates an example of how the pruning masks control the pruned structures.

We formulate pruning as a constrained optimization problem where we learn pruning masks to search for a subnetwork matching a pre-specified target architecture while maximizing performance. Following the ℓ_0 regularization approach [39], we parametrize the pruning masks to model hard concrete distributions, which have a support of $[0, 1]$. While prior work usually control for a target sparsity [62, 66], we use a pair of Lagrange multipliers to impose constraints on the pruned model shape directly.

For example, for a target number of heads $H_{\mathcal{T}}$ (and we use $L_{\mathcal{T}}$, $d_{\mathcal{T}}$, and $m_{\mathcal{T}}$ to represent the target number of layers, hidden dimension, and intermediate dimension respectively), we have the imposed constraint on a single layer as:

$$\tilde{\mathcal{L}}^{\text{head}}(\lambda, \phi, z) = \lambda^{\text{head}} \cdot \left(\sum z^{\text{head}} - H_{\mathcal{T}}\right) + \phi^{\text{head}} \cdot \left(\sum z^{\text{head}} - H_{\mathcal{T}}\right)^2.$$

Similar constraints are applied to pruning other substructures. Overall, we jointly optimize the model weights and pruning masks by a min-max objective $\min_{\theta,z} \max_{\lambda,\phi} \mathcal{L}_{\text{prune}}(\theta, z, \lambda, \phi)$:

$$\mathcal{L}_{\text{prune}}(\theta, z, \lambda, \phi) = \mathcal{L}(\theta, z) + \sum_{j=1}^{L_{\mathcal{S}}} \tilde{\mathcal{L}}_j^{\text{head}} + \sum_{j=1}^{L_{\mathcal{S}}} \tilde{\mathcal{L}}_j^{\text{int}} + \tilde{\mathcal{L}}^{\text{layer}} + \tilde{\mathcal{L}}^{\text{hidden}},$$

where $\mathcal{L}(\theta, z)$ is the language modeling loss computed with the masked model weights. This objective will produce a pruned model with the target shape. Ideally, running this prune algorithm on a large amount of data will directly produce a strong compact model. In practice, the pruning stage is expensive (roughly $5\times$ slower compared to standard LM training), and we find that the learned masks often converge fast. Therefore, in our experiments, we allocate only a limited budget for the pruning process. Following pruning, we finalize the pruned architecture by preserving the highest-scoring components associated with the mask variables in each substructure, and continue training the pruned model with the language modeling objective. We refer to this second stage as continued pre-training.

A.2 Dynamic Batch Loading

Continued pre-training on a large amount of data is crucial for recovering the pruned model performance. However, we observe a surprising finding in our preliminary experiments: continuing pre-training our pruned models on the pre-training dataset RedPajama (56; LLaMA's pre-training dataset) reduces loss at different rates across domains compared to a model trained from scratch with the same data, which signifies an inefficient use of data.

For example, to produce a 2.7B model from a LLaMA2-7B model, we first fit a *scaling law* (26; details in Appendix B) on the series of LLaMA2 models for each domain. Then we predict the loss

Algorithm 1: Dynamic Batch Loading

Require: Training data of k domains $D_1, D_2, \cdots, D_k$, validation data $D_1^{\text{val}}, D_2^{\text{val}}, \cdots, D_k^{\text{val}}$, initial data loading weights $w_0 \in \mathbb{R}^k$, reference loss $\ell_{\text{ref}} \in \mathbb{R}^k$, LM loss function $\mathcal{L}$, training steps T, evaluation interval m, model parameters θ

```
for t = 1, ..., T do
    if t mod m = 0 then
        ℓ_t[i] ← L(θ, D_i^val)
        Δ_t[i] ← max{ℓ_t[i] − ℓ_ref[i], 0}
        w_t ← UpdateWeight(w_{t−m}, Δ_t)          ▷ Update data loading proportion
    end
    Sample a batch of data B from D_1, D_2, ..., D_k with proportion w_t;
    if pruning then
        Update θ, z, φ, λ with L_prune on B
    else
        Update θ with L(θ, B)
    end
end

Subroutine UpdateWeight(w, Δ)
    α ← w · exp(Δ)
    w ← α / Σ_i α[i]
    return w
return θ
```

that a hypothetical 2.7B LLaMA2 model, if trained from scratch on the same data, would achieve. We obtain these estimated *reference losses* across domains of the pre-training data and compare them to the losses of our pruned model after continued pre-training. As shown in Figure 6 (left), while our model's loss on GitHub is better than the reference loss, it is significantly worse than the reference loss on C4. This observation indicates that pruning preserves a greater amount of knowledge in low-entropy and smaller domains (e.g., GitHub) compared to high-entropy and larger domains (e.g., C4). As demonstrated later in Section D.2, simply reusing the original pre-training data distribution[2] results in an inefficient use of data and worse downstream performance, even if the overall loss is seemingly low.

Inspired by [68], a recent work of reweighting data of different domains, we propose *dynamic batch loading*, a more efficient algorithm to simply adjust domain proportions on the fly based on the model performance. The goal is to ensure the model achieves the reference loss at a similar speed across all domains. We introduce the algorithm below.

Problem setup. The pre-training data comprises of k domains $D_1, D_2, \cdots, D_k$ and we have a held-out validation dataset for each domain, denoted as D_i^{val}. At each training step t, a proportion $w_t[i]$ of the data comes from domain D_i. We set a reference validation loss $\ell_{\text{ref}}(D_i)$ for each domain and train the pruned model to reach the reference loss.

Dynamic batch loading. We present the full algorithm in Algorithm 1. In a sketch, for every m steps, we evaluate the model to get the validation loss ℓ_t (step t) on D^{val}, and update w_t based on the difference $\Delta_t(D_i)$ between $\ell_{\text{ref}}[i]$ and $\ell_t[i]$ on each domain. The update rule is exponential ascent following [68],

$$\alpha_t = \log(w_{t-m}) + \Delta_t; \quad w_t = \frac{\exp(\alpha_t)}{\sum_i \exp(\alpha_t[i])}.$$

We apply dynamic batch loading to both the pruning stage and the continued pre-training stage. For pruning, we use the original pre-trainig data's domain weights as w_0. For continued pre-training, we use the final weights from the pruning stage as w_0. Unlike [68], which requires training reference and proxy models to decide a fixed domain weight before the final run, dynamic batch loading leverages reference losses directly and adjusts the weights on the fly with minimal overhead, making it as efficient as standard pre-training. More broadly, dynamic batch loading has the potential to train an

[2]The LLaMA2 pre-training data is not public. However, we observe a similar phenomenon with LLaMA1 models, indicating this is a universal issue unrelated to specific pre-training data.

LLM to match reference losses from any model, even without a full access to the source model's training data.

Choices of reference loss. By default, we use the loss predicted by the scaling law as the reference (denoted as *scaling reference*). We also experiment with an alternative where we directly use the source model's domain validation loss as the reference (denoted as *source reference*). We show in Table 8 and Appendix D.4 that while both variants perform well, using scaling reference leads to slightly better downstream results, especially on math and coding tasks. However, source reference is a viable alternative when only one source model exists (cannot apply the scaling law).

B Reference Loss Predicted by Scaling Laws

The scaling law of language modeling is a function of model size N and dataset size D:

$$L(N, D) = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta}$$

where E captures the loss for the true language distribution in an ideal generation process, and A, α, B, β are scaling factors related to model scale or data size. Models in the same model family are usually trained with the same amount of tokens on the same data distribution. In this case, we need a minimum of three models to estimate the constant $E + \frac{B}{D^\beta}$, A and α. If the models are trained with different amount of tokens, we can estimate E, A, α, B, β with a minimal of 5 models. Note that we will estimate the scaling factors for each domain seperately.

It is known that LLAMA2 models have been trained on the same 2T tokens [59]. Therefore, we take the LLAMA2-7B, LLAMA2-13B and LLAMA2-70B checkpoints, evaluate them on the evaluation set of each domain, and fit the scaling factors with the corresponding loss. Since we have very limited data points for estimating the scaling law constant, we do not deny that the estimated results might be biased. We present the predicted loss in Table 2.

Table 2: Predicted reference loss of hypothetical LLaMA2-1.3B and LLaMA2-2.7B.

	CC	GitHub	Book	StackExchange	Wiki	ArXiv	C4
1.3B	1.964	0.746	2.139	1.612	1.759	1.445	2.125
2.7B	1.871	0.688	2.033	1.535	1.630	1.356	2.033

C Experiments

C.1 Setup

Data. As the training data for LLaMA2 is not publicly accessible, we use RedPajama [56], which is a replicated pre-training dataset of the LLaMA 1 models [58], for pruning and continued-pretraining. This dataset encompasses training data from seven domains: CommonCrawl, C4, Github, Wikipedia, Books, ArXiv, and StackExchange. We construct a held-out validation set with 2 million tokens (equivalent to 500 sequences of 4,096 tokens) for each domain. We allocate 0.4 billion tokens for the pruning phase and 50 billion tokens for the continued pre-training process. Following the conventions of LLaMA2, we maintain a sequence length of 4,096 tokens. Table 3 provides a summary of the pre-training data used by our models and the baseline models.

Table 3: A summary of pre-training dataset used by Sheared-LLaMA and other models.

Model	Pre-training Data	#Tokens
LLaMA	LLaMA data	1T
LLaMA-2	*Unknown*	2T
OPT	OPT data[3]	300B
Pythia	The Pile	300B
INCITE-Base	RedPajama	800B
OpenLLaMA v1	RedPajama	1T
OpenLLaMA v2	OpenLLaMA data[4]	1T
Sheared-LLaMA	RedPajama	50B

[3]OPT data contains BookCorpus [73], Stories [60], CCNews [22], the Pile [17], and PushShift.io Reddit [3].

[4]OpenLLaMA v2 is pre-trained with a mixture of RefinedWeb [47], StarCoder [35], and part of RedPajama.

Downstream task evaluation. We use the `lm-evaluation-harness` package [18] to evaluate on an extensive suite of downstream tasks:

- We follow Pythia and LLaMA2 to report the 0-shot accuracy of ARC easy (ARC-E; 9), LAMBADA [46], LogiQA [37], PIQA [5], SciQ [63], and WinoGrande [49].
- We report accuracy of the tasks used by Open LLM Leaderboard[5], including 10-shot HellaSwag [69], 25-shot ARC Challenge (ARC-C; 9), and 5-shot MMLU [24].
- We also report exact match of 32-shot Natural Questions (NQ; 32) to measure the factual knowledge in the model.

Instruction tuning evaluation. As training models to follow instructions has become a crucial application of LLMs [45, 55], we evaluate our models on instruction tuning and fine-tune both Sheared-LLaMA and baseline models on 10,000 instruction-response pairs sampled from the ShareGPT dataset[6]. For evaluation, we sample another 1,000 instructions from ShareGPT, generate responses from our fine-tuned models and other baseline models, and use GPT-4 as an evaluator to compare the two responses [14]. We report the win rate of our model compared to the baseline model (more details in Appendix G).

D Analysis

D.1 Comparison to further pre-training an existing LM.

We examine which initialization leads to better performance for continued pre-training—our pruned models or an existing LLM of equivalent size. We continue pre-training an INCITE-Base-3B model on the same data and compare it to Sheared-LLaMA-2.7B. We conduct an initial grid search to evaluate various learning rates, encompassing values of 1×10^{-4}, 5×10^{-5}, and 1×10^{-5}. Our initial results reveal that employing the first two learning rates resulted in a noticeable decline in model performance compared to the original model. Consequently, we opt to continue pre-training with a learning rate of 1×10^{-5}. The remaining hyperparameters remain consistent with those outlined in Table 9. It is worth acknowledging that our choice of continued pre-training setup may not be optimal according to recent research [21]; however, it represents the best approach within our constraints.

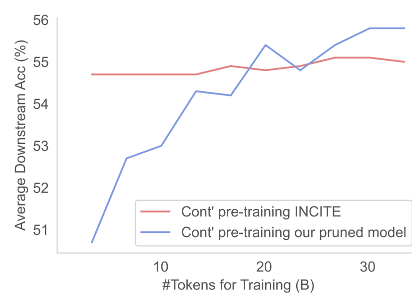

Figure 4: Average downstream performance of continuing pre-training Sheared-LLaMA vs INCITE-Base-3B.

Figure 4 shows that the INCITE-Base-3B model starts off with much higher accuracy, but its performance plateaus throughout continued pre-training. In contract, Sheared-LLaMA starts at a lower accuracy but rapidly improves, eventually surpassing the INCITE-Base-3B model. This suggests that pruned models from a strong base model serve as a better initialization point for continued pre-training.

D.2 Effectiveness of Dynamic Batch Loading

We analyze the effectiveness of dynamic batch loading by examining its impact on three aspects: the final LM loss across domains, the data usage of each domain throughout training, and the downstream task performance. All results in this section are based on Sheared-LLaMA-1.3B.

Loss differences across domains. Dynamic batch loading is designed to balance the rate of loss reduction across domains, so that the losses reach the reference value at approximately the same time. In Figure 6, we plot the difference between the loss of our model (with both original and dynamic batch loading) and the reference loss, estimated by fitting a scaling function to a hypothetical 2.7B parameter LLaMA2 model. With the original batch loading, the loss differences vary dramatically

[5] `https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard`

[6] `https://sharegpt.com`. We only use the first round in the multi-turn chat history.

Table 4: Domain data usage with dynamic loading compared to the original RedPajama proportions.

	CC	GitHub	Book	StackExchange	Wiki	ArXiv	C4
RedPajama	67.0%	4.5%	4.5%	2.0%	4.5%	2.5%	15.0%
Dynamic Batch Loading	36.1%	0.8%	9.1%	1.0%	3.1%	0.7%	49.2%

across domains. For instance, the GitHub loss decreases below the reference value, while the C4 loss lags behind. In contrast, dynamic batch loading reduces losses evenly and shows very similar loss differences across domains, indicating a more efficient data use.

Data usage. Table 4 compares the original data proportion of RedPajama and the domain data usage of our dynamic loading (Figure 5 shows the evolution of domain weights throughout the training). We see that dynamic batch loading increases the weights for the Book and C4 domains versus other domains—suggesting that they are more difficult to recover for a pruned model.

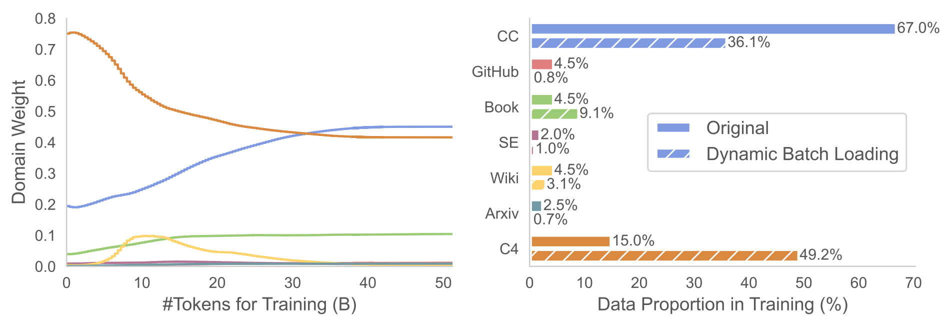

Figure 5: Data weight of each batch during the continued pre-training stage.

Downstream performance. As shown in Figure 7, pruned models trained with dynamic batch loading achieve better downstream performance than when trained on the original RedPajama distribution. This suggests that the more balanced loss reduction from dynamic loading transfers to improved downstream capabilities.

D.3 Comparison to Other Pruning Approaches

We compare our LLM shearing method to other pruning approaches and report validation perplexity, which serves as a strong indicator of overall model capabilities [65].

Targeted pruned models have a higher inference throughput. Previous works like Block Pruning [33] or CoFiPruning [66] are experimented on BERT-scale LMs, and the final model architectures, though structured, usually have non-uniform layer configurations, e.g., different layers have different number of heads or intermediate size. While bringing performance gains, non-uniformity also introduces training and inference overhead due to irregularities in model architectures. As shown in Table 5, our targeted pruned models have a higher inference throughput compard to the non-uniformly pruned CoFiPruning model at the same sparsity, despite having slightly higher perplexity.

Table 5: Validation perplexity and inference throughout (tokens/second) of targeted structured pruning (without continued pre-training) with a uniform layer configuration, and CoFiPruning, with a non-uniform layer configuration. Throughput is measured on a Nvidia A100 (80G) GPU, with a batch size of 1 and a sequence length of 512.

	Layer Config	PPL ↓	Throughput ↑		Layer Config	PPL ↓	Throughput ↑
1.3B	CoFiPruning	9.1	51	3B	CoFiPruning	7.0	37
	Ours	10.3	58		Ours	7.7	43

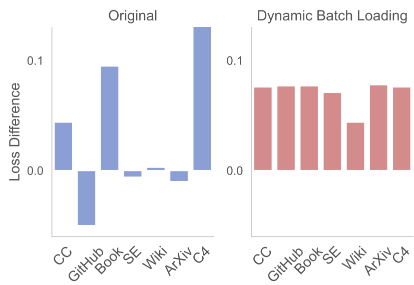

Figure 6: Loss difference between the pruned model (1.3B) and estimated reference loss, with original vs. dynamic batch loading.

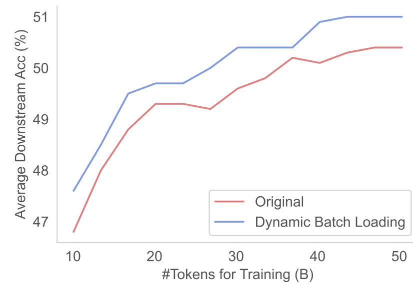

Figure 7: Downstream task performance of Sheared-LLaMA-1.3B with original data proportion and dynamic batch loading.

Table 6: Data budget allocation to pruning and continued pre-training (CT) and corresponding perplexity.

# Tokens		**PPL**	
Pruning	CT	Pruning	CT
0.2B	4.6B	12.99	7.46
0.4B	4.4B	10.29	7.32
0.8B	4.0B	9.01	7.23
1.6B	3.2B	8.04	7.08

Comparison to LLM-Pruner [41]. We compare our pruning method to LLM-Pruner, a recent work in uniform layer configuration structured pruning, in **??**. It shows the model configurations of an LLM-Pruner [41] pruned model and our pruned model. Figure 8 compares LLM-Pruner and ours in continued pre-training. Our model achieves lower perplexity than LLM-Pruner with a similar parameter count and the same amount of continued pre-training, demonstrating the effectiveness of our method.

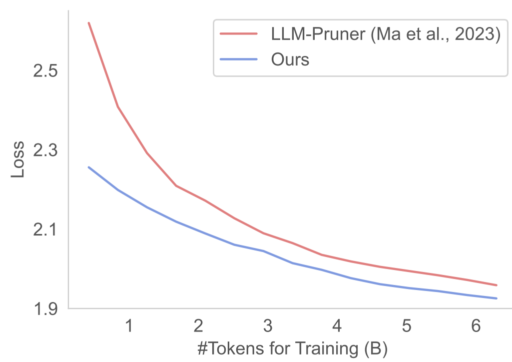

Figure 8: The loss of the 1.3B model with original loading and dynamic loading and the downstream trajectory.

D.4 Additional Analysis

Pruning vs. continued pre-training budget. Intuitively, allocating more compute to the pruning stage helps identify better subnetwork structures. We explore distributing data across pruning and continued pre-training stages differently, within a fixed budget of 5B tokens. Table 6 shows that when controlling the total amount of tokens, increasing the pruning budget consistently improves perplexity. However, since pruning is more expensive than continued pre-training (Appendix E for details on training throughputs), we decide to allocate 0.4B tokens to pruning.

Performance on math and coding tasks. We examine the math and coding abilities of our pruned models compared to other language models. We find that the math ability of existing 3B parameter

Table 7: Model structure of Pythia-1.4B, LLM-pruner (1.3B), and Ours (1.3B). With a similar parameter count, our pruned model structure has a lower perplexity when fine-tuned with the same amount of tokens (around 6B tokens).

	Layers	Heads	Head size	Intermediate size	Hidden size	Params	PPL
Pythia-1.4B	24	16	128	8192	2048	1.4B	-
LLM-pruner (1.6B)	32	7	128	2201	4096	1.6B	7.09
Ours (1.3B)	24	16	128	5504	2048	1.3B	6.85

Table 8: Evaluation results on GSM8K and HumanEval and training percentage and tokens in ArXiv and GitHub.

Models	**GSM8K (8)**	**HumanEval**		**ArXiv**	**Github**	**ArXiv**	**GitHub**
	EM	**Pass@1**	**Pass@5**	**Percentage**	**Percentage**	**Tokens**	**Tokens**
LLaMA2-7B	13.7	12.8	23.8	-	-	-	-
OPT-2.7B	0.1	0.0	0.0	-	-	-	-
Pythia-2.8B	1.7	5.1	14.6	9.0%	7.6%	26.9	22.8
INCITE-Base-3B	1.8	4.3	4.9	2%	4.5%	16.0	36.0
Open-LLaMA-3B-v1	2.5	0.0	1.2	2%	4.5%	20.0	45.0
Open-LLaMA-3B-v2	2.7	10.4	20.1	-	-	-	-
Sheared-LLaMA-3B (Source)	2.7	3.7	5.5	0.7%	0.4%	0.3	0.2
Sheared-LLaMA-3B (Scaling)	2.4	4.9	9.2	1.0%	0.8%	0.5	0.4

models, including Sheared-LLaMA, is still far below that of larger models. We also find that Sheared-LLaMA's coding ability lags behind models known to be trained on more code data, like Pythia-1.4B and Open-LLaMA-3B-v2. Sheared-LLaMA's coding ability likely comes from the original LLaMA2 model, speculated to have used more code data, and the minimal code data used in our pruning experiments.

Scaling Reference vs. Source Reference Figure 9 compares Sheared-LLaMA with the scaling reference and the source reference. While both methods are effective in efficiently training the model, the scaling reference performs consistently (slightly) better in terms of downstream performance.

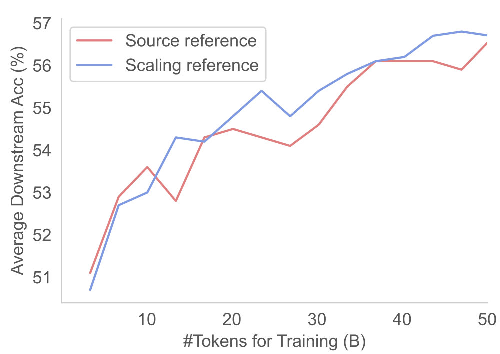

Figure 9: Average downstream peformance of Sheared-LLaMA with the scaling reference and the source reference.

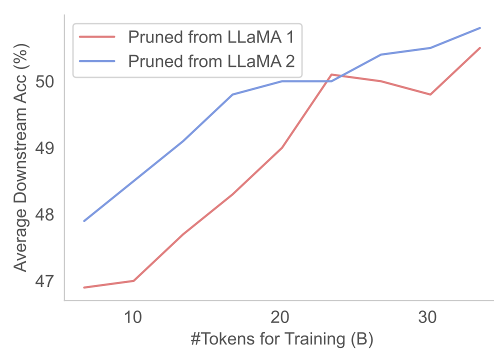

Figure 10: A comparison between pruning from LLaMA1 and LLaMA2 with dynamic loading to match the source reference loss.

Pruning from LLaMA1 vs LLaMA2 Appendix D.4 compares the performance of pruning from LLaMA 1 and LLaMA 2. Both models demonstrate strong downstream task performance, though not surprisingly, pruning from LLaMA 2 yields a slight advantage.

E Training Details

We present the hyperparameters used in our experiments in Table 9. We use fully sharded data parallel [72] to train our models in parallel. We use FlashAttention V1 [10] to speed up training. We use a cosine learning rate scheduler and decay to a minimum of 10% of the peak value·

Table 9: Training hyper-parameters and throughput.

	Pruning	Contined Pre-training
Training budget	0.4B	50B
Learning rate of z, ϕ, λ	1.0	-
Learning Rate of θ	0.0001	0.0001
LR warmup ratio	10%	3%
Batch size (tokens)	131K	1M
Evaluation interval m (steps)	50	400
Steps	$3,200$	$51,200$
# GPUs	8	16
Throughput (tokens/s)	15K	145K (1.3B) / 77K (2.7B)

F Model Configurations

Table 10 shows the model configurations of all the baselines we use. Note that one constraint of our pruned models is that it needs to have the same head dimension (hidden dimension / #head) as the source model.

In this section, we provide the model configurations for both our Sheared-LLaMA model and the baseline models, as illustrated in Table 10. Our design closely adheres to the architecture of Pythia-1.4B and INCITE-Base-3B, albeit with some nuanced distinctions. A noteworthy difference is found in the intermediate size of Sheared-LLaMA, which is a consequence of its lineage from LLaMA2-7B. Notably, LLaMA2-7B employs a GLU variant [52] within its feed-forward layer, comprising a gate matrix, an upward-projection matrix, and a downward-projection matrix. In contrast, other models within this context employ the conventional double-matrix feed-forward layer structure. Furthermore, we acknowledge that our algorithm must operate within the constraints defined by the head dimension of the source model, which our pruned model must align with. Instead of explicitly specifying the number of heads, we implement adjustments based on the specified hidden dimensions.

Table 10: Model configurations of our Sheared-LLaMA and baseline models.

Model	**#Param**	**#Layers**	**Hidden**	**Intermediate**	**#Heads**	**Head Dim**
OPT-1.3B	1.3B	24	2048	8192	32	64
Pythia-1.4	1.4B	24	2048	8192	16	128
Sheared-LLaMA-1.3B	1.3B	24	2048	5504	16	128
OPT-2.7B	2.7B	32	2560	10240	32	80
Pythia-2.8B	2.8B	32	2560	10240	32	80
INCITE-Base-3B	2.8B	32	2560	10240	32	80
OpenLLaMA-3B	2.7B	26	3200	8640	32	100
Sheared-LLaMA-2.7B	2.7B	32	2560	6912	20	128
LLaMA2-7B	6.7B	32	4096	11008	32	128

G Instruction Tuning

As [61] observes the judge could change its preference when swapping the presentation order of the two outputs, we compare each output pair twice by swapping the presentation order of the two outputs and finally report the average win-rate of the two rounds. In this analysis, we randomly select an output generated by Sheared-LLaMA-1.3B and Sheared-LLaMA-2.7B in response to a given instruction, and the results are presented in Table 11. Our findings demonstrate that, after instruction tuning, Sheared-LLaMA-2.7B consistently produces long, coherent, and informative outputs in response to the instruction. During training, the prompt is prepended with `You are a helpful assistant. Write a response that appropriately completes the request.`.

Table 11: An example output from Sheared-LLaMA-1.3B and Sheared-LLaMA-2.7B. It shows that Sheared-LLaMA can generate long, coherent, and informative responses after instruction tuning.

```
Prompt:  Act as semiconductor industry analyst

Sheared-LLaMA-1.3B: As a semiconductor industry analyst, I would be responsible
for conducting market research and analysis to help companies in the industry
understand and respond to market trends, customer preferences, and competitive
dynamics.  This would include analyzing market data such as sales, market share,
and industry trends, as well as conducting interviews with industry leaders,
executives, and customers to gain insights into customer needs and preferences.  I
would also be responsible for developing and implementing marketing strategies to
promote products and services, and for identifying and pursuing new business
opportunities.

Sheared-LLaMA-2.7B: As a semiconductor industry analyst, my primary responsibility
is to provide insightful and accurate analysis on the global semiconductor market.
This includes evaluating the performance of individual companies, industries, and
trends within the market.

Here are some key areas where I would focus my analysis:

1.  Market size and growth:  I would track the size of the global semiconductor
market and analyze the growth prospects for each segment, such as memory, logic,
and sensors.
2.  Competitive landscape:  I would assess the competitive landscape, including
the top players, their market share, and strategic moves.
3.  Technology trends:  I would monitor the development of new technologies, such
as 5G, AI, and IoT, and their impact on the semiconductor industry.
4.  Industry trends:  I would track industry trends, such as the shift towards
sustainable technologies, and evaluate their impact on the market.
5.  Market forecasts:  I would provide forecasts for the semiconductor market,
including revenue and unit shipment growth, based on a range of scenarios and
assumptions.

In addition to these areas, I would also keep track of industry news and
developments, and provide insights on their impact on the market.
```

H Related Work

Pruning. Structured pruning has been extensively studied as a model compression technique in computer vision and natural language processing, where task-specific models like classification ones are often overparameterized and can be pruned significantly with minimal impact on performance [23, 64, 38, 40, 6, 11, 27, 62, 33, 66, 31]. Unstructured pruning [15, 7, 51] prunes individual neurons instead of structured blocks. Though unstructured pruning usually achieve higher compression rates, they are not practical for model speedup.

In the era of LLMs, the prevalent NLP pipeline has shifted from task-specific models to general-purpose LMs, which leaves little room for redundancy. Both unstructured pruning, semi-structured pruning [16, 53], and structured pruning [41] lead to significant performance drops on LLM even at a modest sparsity. Noticeably, all the aforementioned works fix the original model parameters or tune them minimally. In our work, we see pruning as an initialization and consider it necessary to expend substantial compute to continually pre-training the model to recover performance.

Efficient pre-training approaches. As orthogonal to our pruning approach, There is an extensive body of work on improving efficiency of training LLMs. For example, quantization reduces the numeric precision of model weights and activations and speeds up training and inference [12, 13, 67]. Knowledge distillation [25, 50, 29, 54], which trains a smaller model on a larger model's prediction, is shown to be effective for task-specific models [66]; nonetheless, there is little evidence showing that it is a more efficient way to train general-purpose LLMs given its exceeding compute cost [48]. More methods have been introduced to enhance the efficiency of training LMs, such as dynamic architectures [20, 70] and efficient optimizers [8, 36]. However, as indicated by [30], the promised gains in training efficiency may not be consistently realized.

There are also data-based approaches to enhance training efficiency. Eliminating duplicated data is found to be effective [34]. Various batch selection techniques propose to prioritize data based on criteria such as higher losses [28] or a greater reducible loss [42]. [68] propose to optimize data mixtures by training a proxy model to estimate the optimal data weight of each domain.