DeepSpeed Data Efficiency: Improving Deep Learning Model Quality and Training Efficiency via Efficient Data Sampling and Routing

Conglong Li[*], **Zhewei Yao**[*] , **Xiaoxia Wu**[*] , **Minjia Zhang,**
Connor Holmes, Cheng Li, Yuxiong He

Abstract

Recent advances on deep learning models come at the price of formidable training cost. The increasing model size is one of the root causes, but another less-emphasized fact is that data scale is actually increasing at a similar speed as model scale, and the training cost is proportional to both of them. Compared to the rapidly evolving model architecture, how to efficiently use the training data (especially for the expensive foundation model pretraining) is both less explored and difficult to realize due to the lack of a convenient framework that focus on data efficiency capabilities. To this end, we present DeepSpeed Data Efficiency, a framework that makes better use of data, increases training efficiency, and improves model quality. Specifically, we propose and combine two data efficiency techniques: efficient data sampling via a general curriculum learning library, and efficient data routing via a novel random layerwise token dropping technique. For GPT-3 1.3B language model pretraining, our work achieves 12.5x less data/time/cost ($3.7K if rent on Azure), while still maintaining 95% of model quality compared to baseline with full data and cost ($46.3K). For GPT-3 1.3B and BERT-large pretraining, our work can also achieve the same model quality with up to 2x less data/time/cost, or achieve better model quality under same data/time/cost. DeepSpeed Data Efficiency is easy to use and tune, enabling us to easily apply it and verify its benefit on additional tasks including GPT-3 MoE model pretraining and small-scale GPT-2/ViT finetuning.

1 Introduction

Recently, large-scale deep learning models are empowering us to achieve more in many ways, such as code generation [17] and text-to-image generation [40, 41]. To keep improving the service quality, deep learning model architecture evolves rapidly, and the model size is also growing at a tremendous speed. The increasing model size leads to unprecedented training cost (especially for foundation model pretraining), which recently grows to 2 months on thousands of GPUs/TPUs [47, 9]. On the other hand, a less-emphasized perspective is that **data scale is actually increasing at a similar speed as model scale, and the training cost is proportional to both of them**. As plotted in Fig. 1, for several representative language models in the last 5 years both the model and data scales increase at a similar speed. Recent works including Chinchilla [20] and PaLM

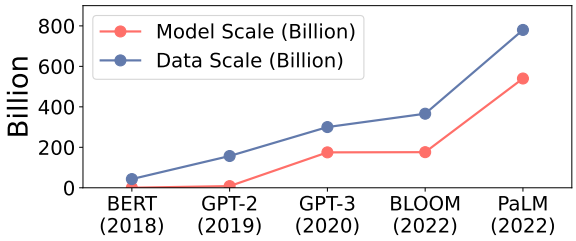

Figure 1: Model scale (number of parameters) and data scale (number of consumed training tokens) of representative language models in the last 5 years [14, 46, 7, 45, 9].

[*]equal contribution. Code has been released as a part of https://github.com/microsoft/DeepSpeed.

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

Table 1: Comparing DeepSpeed Data Efficiency with SOTAs.

	Efficient data sampling	Efficient data routing	Verified workloads	Key achievements
Sequence length warmup [29]	1 specific CL metric	N/A	GPT-2/GPT-3 pretraining	1.3x data/cost saving with 100% model quality
TokenBypass [21]	N/A	TokenBypass	BERT pretraining	1.33x data/cost saving with 100% model quality
Proposed XYZ Data Efficiency	general CL library support	random-LTD	GPT-3/BERT/MoE pretraining GPT-2/ViT finetuning	12.5x data/cost saving with 95% model quality 2x data/cost saving with 100% model quality

2 [18] emphasize the need of increasing data scale at an even faster speed. This demonstrates the importance of improving data efficiency: achieve same model quality with less data and reduced training cost, or achieve better model quality with the same amount of data and similar training cost.

There are two popular research directions among existing data efficiency techniques: Data sampling techniques aim to improve the convergence speed by sampling the most suitable next data batch from the whole data pool; Data routing techniques aim to reduce the computation by routing each data to only a subset of the model components. These techniques improve data and training efficiency, but existing solutions have several limitations:

- Techniques like curriculum learning improves data efficiency by indexing and sampling training data based on certain difficulty metric [3], and it is recently proved effective on large-scale pretraining tasks [29]. However, implementing different CL strategies for different user tasks can require a lot of code-refactoring, which is time-consuming and error-prone. In addition, existing implementations have less consideration on scalability, which makes it difficult to analyze and index large-scale training data based on different difficulty metrics.
- Existing data routing techniques such as token drop/bypass/pruning were mostly designed for inference and inapplicable to training. TokenBypass [21], to our knowledge the only data routing technique for foundation model pretraining, skips the compute of part of the input tokens at some middle layers during BERT pretraining, reducing pretraining cost while maintaining model quality. However, it requires several special implementations that may only work for the tested BERT pretraining case, such as the importance score-based token dropping decisions.
- Although promising data efficiency solutions have been proposed independently, combining multiple methods together for the best outcome is still a laborious process, requiring changes in multiple places in the training pipeline: data loader, data sampler, model architecture, etc. Another challenge is that existing techniques add additional hyperparameters but with no clear tuning strategy.

To address these above challenges, we present DeepSpeed Data Efficiency, a framework that makes better use of data, increases training efficiency, and improves model quality. Our **contribution** are:

- **Efficient data sampling via general curriculum learning library.** We present a general curriculum learning (CL) library that is both scalable and customizable: it includes a map-reduce based data analyzer that enables scalable analysis and indexing of massive data based on any possible CL metric; it includes a general CL-based data sampler and loader design for users to apply any customized CL strategies. Using this library, we are able to thoroughly explore different CL strategies for GPT-3 1.3B and BERT-large pretraining, and identify the best solution that provides better data and training efficiency than existing CL solution. This library (and the whole DeepSpeed Data Efficiency framework) has been open sourced in a deep learning acceleration library (name hidden for anonymity) that is fully compatible with PyTorch.
- **Efficient data routing via random layerwise token dropping.** We present a novel data routing technique called random layerwise token dropping (random-LTD) to skip the computation of a subset of the input tokens at all middle layers. Random-LTD employs a simple yet effective routing strategy and requires minimal model architecture change. It is very flexible to apply random-LTD to various tasks (GPT-3/GPT-3 MoE/BERT pretraining and GPT/ViT finetuning) which the SOTA technique (TokenBypass) does not explore or provides less improvement.

- **An easy to use/tune framework that maximizes data/training efficiency.** DeepSpeed Data Efficiency seamlessly composes the two proposed techniques, and only requires minimal changes on user side. To our knowledge, we are the first to demonstrate that composing data sampling and routing techniques can lead to even better data/training efficiency, especially for foundation model pretraining: For GPT-3 1.3B pretraining, Fig. 2 shows that our approach provides better model quality at all cost budgets, advancing the whole cost-quality Pareto frontier. In particular, we achieve up to 12.5x data/time/cost saving while still maintaining 95% of the model quality (zero-shot eval accuracy) compared to the baseline with full data, while baseline can only maintain 91% of the model quality, a 1.8x higher quality degradation. Based on measured training time, 12.5x would be a cost reduction from \$46.3K to \$3.7K if renting similar hardware on Azure [2], greatly democratizing research and usage of foundation models for AI community. For GPT-3 1.3B and BERT-large pretraining, we can also achieve up to 2x data and 2x time saving together with better or similar model quality as compared to the baseline training with full data, greatly surpassing state-of-the-art data efficiency solutions as summarized in Tab. 1. Both techniques under our framework are easy to use and tune, and we include a low-cost tuning strategy and a summarized usage guidelines.

2 Design

At high-level, the proposed DeepSpeed Data Efficiency framework has two components as shown in Fig. 5: First we have efficient data sampling, where instead of the baseline's random sampling, we aim to sample the most suitable next data batch from the whole data pool by a general curriculum learning (CL) library. Second we have efficient data routing, where instead of passing all input data to all model components, we aim to efficiently route each data through different components of model by leveraging the proposed random layerwise token dropping (random-LTD) technique. This section presents the design of the two techniques, how we compose them, together with a low-cost tuning strategy and a summarized usage guidelines.

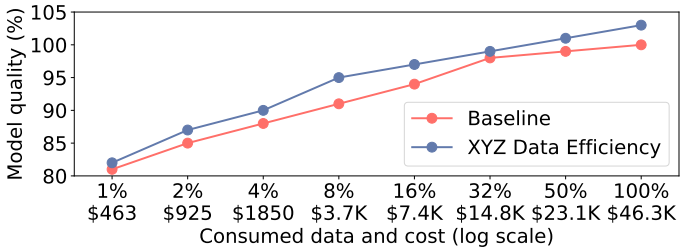

Figure 2: GPT-3 1.3B pretraining: relative model quality (baseline with full data as 100% quality) under different data consumption (1% to 100%) and training cost (when renting on Azure).

Efficient data sampling via curriculum learning. To solve the limitations of existing CL solutions as described in previous sections, we design and implement a general curriculum learning library emphasizing the scalability and customizability. It consists of three components as shown in top part of Fig. 5. First we use a data analyzer to perform the offline CPU-only data analysis which indexes the whole data pool based on any difficulty metric, which could be the sequence length, the vocabulary rarity, or anything defined by user. This data analyzer employs a Map-Reduce scheme: During the Map stage, user provides a function that computes the desired difficulty metric, the raw training dataset, and other configurations such as number of CPU nodes and number of threads per node. Then the data analyzer will automatically splits the dataset based on number of workers, compute the difficulty values in a batched fashion, and write the results to two indexes: one index maps each data sample to its difficulty value, and another index maps each distinct difficulty value to the corresponding samples. During the Reduce stage, the data analyzer will merge the index files produced by all workers. This Map-Reduce scheme is necessary since the training data could be huge thus has to be distributed. For instance, we have 173 million data samples (each with sequence length 2048) for GPT-3 pretraining and 2.5 billion data samples (each with sequence length $\leqslant$ 512) for BERT pretraining. To reduce the memory overhead when analyzing the huge dataset, we write the index files as numpy memory-mapped files. Using this data analyzer we are able to efficiently analyze GPT-3 and BERT pretraining data based on various difficulty metrics. Using 40 CPU threads on a single node with AMD EPYC 7V12 64-Core Processor, we can finish the analysis on one metric within 3/80 hours for GPT-3/BERT data, respectively. For more details, we refer to Sec. A.2

```
if meth == "baseline":
  hs = Layer(hs)
if meth == "random-LTD":
  k_hs, d_hs = gather(hs)
  k_hs = Layer(k_hs)
  hs = combine(k_hs, d_hs)
```

Figure 3: random-LTD requires a few lines of code. hs, k_{hs}, and d_{hs} means the full input, kept input, and dropped input. "gather", "Layer", "combine" means the functions for random selection, transformer layer, and order-preserved token combination.

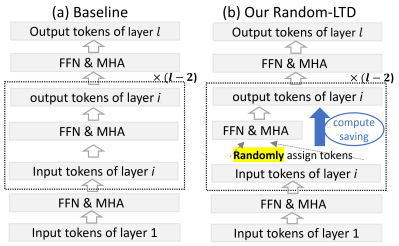

Figure 4: Transformer layers for baseline and random-LTD. The dash-line box is repeated by $l - 2$ times.

Efficient data routing via random-LTD. Existing token dropping methods for inference and training either permanently drop tokens from the compute graph at intermediate layers, or at least make some tokens fully skip a consecutive series of middle layers (Sec. A.1). However, several works [50, 31, 51] have shown that MHA focuses on different tokens at different layer depths and the attention map aligns with the dependency relation most strongly in the middle of transformer architectures. Therefore, fully skipping middle layers like TokenBypass [21] may hinder the learnability/generalization of the architecture during pretraining/inference. In order to overcome this problem, we propose a layerwise token dropping (LTD) mechanism. Instead of fully bypassing same tokens over all middle layers, each transformer layer independently drops/retains its own set of tokens. Various importance score-based metrics are used to determine the token dropping criterion. Most of them can be categorized in attention score-based or loss/frequency-based metrics. However, both of them introduce challenges that make LTD less practical: Instead of importance score, we propose to use *purely random* token dropping assignment and prove its effectiveness in all our experiments. For each transformer layer, we randomly (uniformly) select a small batch of tokens to proceed and drop the rest.

Combining layerwise token dropping with random token dropping, we have our final random and layerwise token dropping method (random-LTD), which can efficiently apply token dropping for each individual layer and can capture the attention dependency of each token with other others in middle layers with high probability. As a result, our experiments on BERT pretraining confirm that random-LTD does not require and won't benefit from special token treatment used by the TokenBypass work, further reducing the implementation complexity. Fig. 4 presents the comparison between standard baseline training and random-LTD. The pseudo-code is given in Fig. 3. For each layer, random-LTD randomly selects (function "gather") a subset of the tokens and feeds (function "Layer") them into the transformer layer. Afterward, we combine (function "combine") the output of transformer layer with the dropped tokens to recover the full sequence length in a order-preserved manner. Thus, the next layer still receives the full sequence and can repeat this process. To apply random-LTD to an existing training pipeline, user just needs to provide the module class name that they want to apply random-LTD (e.g., a TransformerLayer class). Then DeepSpeed Data Efficiency will wrap the module with a new module that includes token dropping capability, and drop some of the input tokens for this module during training. More details can be found in Sec. A.3

3 Evaluations and Conclusion

Evaluations. We evaluate DeepSpeed Data Efficiency in Sec. B by GPT-3/GPT-3 MoE/BERT pretraining and GPT-2/ViT finetuning. Sec. B.8 includes studies of the TokenBypass method on GPT finetuning and pretraining, further demonstrating the advantages of the proposed random-LTD method. We all achieve better results.

Conclusion. Unlike model scale which could reduce in the future with novel architecture, the amount of available training data will increase continuously and irreversibly. Language model pretraining is one of the first to reach a data scale that even training one full epoch is difficult, but sooner or later all machine learning tasks will face the same data efficiency challenge. In this work we propose the DeepSpeed Data Efficiency framework, which demonstrate the power of composing 2 novel data efficiency techniques together. This enables us to achieve an up 12.5x data/time/cost saving (from \$46.3K to \$3.7K on Azure) while maintaining 95% of model quality for GPT-3 pretraining, an up to 2x saving for GPT-3 and BERT pretraining while maintaining 100% model quality, or to achieve even better model quality under similar data and cost. DeepSpeed Data Efficiency is easy to use and tune, which enables us to apply it and verify the benefit on additional GPT-3 MoE pretraining and GPT-2/ViT finetuning tasks.

References

[1] Ardavan Afshar, Ioakeim Perros, Evangelos E Papalexakis, Elizabeth Searles, Joyce Ho, and Jimeng Sun. Copa: Constrained parafac2 for sparse & large datasets. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 793–802, 2018.

[2] Microsoft Azure. Pricing calculator. `https://azure.microsoft.com/en-us/pricing/calculator/`, 2023.

[3] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[4] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1533–1544, 2013.

[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*, pages 7432–7439, 2020.

[6] Ondřej Bojar, Jindřich Helcl, Tom Kocmi, Jindřich Libovickỳ, and Tomáš Musil. Results of the wmt17 neural mt training task. In *Proceedings of the second conference on machine translation*, pages 525–533, 2017.

[7] 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.

[8] Daniel Campos. Curriculum learning for language modeling. *arXiv preprint arXiv:2108.02170*, 2021.

[9] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[10] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

[11] 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.

[12] Ido Dagan, Dan Roth, Mark Sammons, and Fabio Massimo Zanzotto. Recognizing textual entailment: Models and applications. *Synthesis Lectures on Human Language Technologies*, 6(4):1–220, 2013.

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

[15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[16] 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.

[17] GitHub. Github copilot. `https://github.com/features/copilot/`, 2021.

[18] Google. Palm 2 technical report. `https://ai.google/static/documents/palm2techreport.pdf`, 2023.

[19] Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan Chakaravarthy, Yogish Sabharwal, and Ashish Verma. Power-bert: Accelerating bert inference via progressive word-vector elimination. In *International Conference on Machine Learning*, pages 3690–3699. PMLR, 2020.

[20] 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.

[21] Le Hou, Richard Yuanzhe Pang, Tianyi Zhou, Yuexin Wu, Xinying Song, Xiaodan Song, and Denny Zhou. Token dropping for efficient BERT pretraining. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3774–3784, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[22] Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

[23] Gyuwan Kim and Kyunghyun Cho. Length-adaptive transformer: Train once with length drop, use anytime with search. *arXiv preprint arXiv:2010.07003*, 2020.

[24] Sehoon Kim, Sheng Shen, David Thorsley, Amir Gholami, Woosuk Kwon, Joseph Hassoun, and Kurt Keutzer. Learned token pruning for transformers. *arXiv preprint arXiv:2107.00910*, 2021.

[25] Tom Kocmi and Ondřej Bojar. Curriculum learning and minibatch bucketing in neural machine translation. In *Proceedings of the International Conference Recent Advances in Natural Language Processing, RANLP 2017*, pages 379–386, 2017.

[26] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[27] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.

[28] Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*. Citeseer, 2012.

[29] Conglong Li, Minjia Zhang, and Yuxiong He. The stability-efficiency dilemma: Investigating sequence length warmup for training gpt models. In *Advances in Neural Information Processing Systems*, 2022.

[30] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[31] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019.

[32] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

[33] MosaicML. Sequence length warmup, mosaicml composer. `https://docs.mosaicml.com/en/v0.11.1/method_cards/seq_length_warmup.html`, 2022.

[34] Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. Adversarial nli: A new benchmark for natural language understanding. *arXiv preprint arXiv:1910.14599*, 2019.

[35] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

[36] Emmanouil Antonios Platanios, Otilia Stretcu, Graham Neubig, Barnabás Póczos, and Tom M Mitchell. Competence-based curriculum learning for neural machine translation. In *NAACL-HLT*, 2019.

[37] Ofir Press, Noah A Smith, and Mike Lewis. Shortformer: Better language modeling using shorter inputs. *arXiv preprint arXiv:2012.15832*, 2020.

[38] Ofir Press, Noah A Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021.

[39] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International Conference on Machine Learning*, pages 18332–18346. PMLR, 2022.

[40] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[41] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[42] Mrinmaya Sachan and Eric Xing. Easy questions first? a case study on curriculum learning for question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 453–463, 2016.

[43] Mrinmaya Sachan and Eric Xing. Self-training for jointly learning to ask and answer questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 629–640, 2018.

[44] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8732–8740, 2020.

[45] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

[46] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[47] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

[48] Yi Tay, Shuohang Wang, Anh Tuan Luu, Jie Fu, Minh C Phan, Xingdi Yuan, Jinfeng Rao, Siu Cheung Hui, and Aston Zhang. Simple and effective curriculum pointer-generator networks for reading comprehension over long narratives. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4922–4931, 2019.

[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[50] Jesse Vig and Yonatan Belinkov. Analyzing the structure of attention in a transformer language model. *arXiv preprint arXiv:1906.04284*, 2019.

[51] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*, 2019.

[52] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[53] Hanrui Wang, Zhekai Zhang, and Song Han. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 97–110. IEEE, 2021.

[54] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

[55] Benfeng Xu, Licheng Zhang, Zhendong Mao, Quan Wang, Hongtao Xie, and Yongdong Zhang. Curriculum learning for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6095–6104, 2020.

[56] Vikas Yadav, Steven Bethard, and Mihai Surdeanu. Quick and (not so) dirty: Unsupervised selection of justification sentences for multi-hop question answering. *arXiv preprint arXiv:1911.07176*, 2019.

[57] Greg Yang, Edward J Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *arXiv preprint arXiv:2203.03466*, 2022.

[58] Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1803–1811, 2019.

[59] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

[60] Sheng Zhang, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, Kevin Duh, and Benjamin Van Durme. Record: Bridging the gap between human and machine commonsense reading comprehension. *arXiv preprint arXiv:1810.12885*, 2018.

[61] Wei Zhang, Wei Wei, Wen Wang, Lingling Jin, and Zheng Cao. Reducing bert computation by padding removal and curriculum learning. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 90–92. IEEE, 2021.

[62] Xuan Zhang, Gaurav Kumar, Huda Khayrallah, Kenton Murray, Jeremy Gwinnup, Marianna J Martindale, Paul McNamee, Kevin Duh, and Marine Carpuat. An empirical exploration of curriculum learning for neural machine translation. *arXiv preprint arXiv:1811.00739*, 2018.

[63] Xuan Zhang, Pamela Shapiro, Gaurav Kumar, Paul McNamee, Marine Carpuat, and Kevin Duh. Curriculum learning for domain adaptation in neural machine translation. In *NAACL-HLT*, 2019.

A Appendix

A.1 Background and Related Works

Data sampling. For deep learning, the most common data sampling method for minibatch stochastic gradient descent is uniform sampling, where at each step a batch of data is drawn uniformly at random from the whole training data. However, it's potentially beneficial to focus on different kinds of data at different training stages. One example is the curriculum learning technique [3] which aims to improve training convergence speed by presenting relatively easier or simpler examples earlier during training. Building a curriculum learning solution usually requires two components: the difficulty metric (i.e., how to quantify the difficulty of each data sample) and the pacing function (i.e., how to decide the difficulty range when sampling next training data batch). In the NLP area, curriculum learning has been applied on small-scale one-stage tasks and downstream finetuning tasks, such as neural machine translation (NMT) [25, 6, 62, 36, 63] and natural language understanding (NLU) [42, 43, 48, 55]. There are also a few works that explore curriculum learning for language model pretraining [37, 61, 8, 29]. However, one common limitation among existing works is that there does not exist a scalable and customizable curriculum learning library, making it difficult to analyze large-scale data and explore custom difficulty metrics/pacing functions. One evidence is that most of the curriculum learning works for language model pretraining only focus on the sequence length metric due to the difficulty of exploring other metrics on the huge pretraining dataset.

Data routing. In common deep learning training, the model is considered as a whole and all sampled data will be routed to all model components. However, it's potentially beneficial to route each data sample to only a subset of model components, improving the training efficiency. One direction of efficient data routing is to add data bypassing/skipping capability to existing model architectures such as Transformer. Transformer [49] architecture is a stack of transformer layers, each of which has two main ingredients, i.e., the multi-head attention (MHA) and the feed-forward connection network (FFC). Suppose the transformer has l layers denoted as $L_1, \ldots, L_l$. Let $X_i \in \mathbb{R}^{s \times d}$ be the output tensor of $i-$th transformer layer, and x_0 be the input (after embedding) of the transformer. Here s is the sequence length and d is the hidden dimension. Several token dropping/bypassing/pruning techniques [24, 19, 23, 38, 53] were proposed for BERT inference to reduce the computational overhead, but they are not practical for training. In these works, if a token i ($X_{j,i}$) is decided to be dropped at layer j (L_j), the compute cost of this token through all remaining layers (L_k where $k > j$) is eliminated. As such, the sequence length s_i of the i-th layer's input X_{i-1} will be a non-increasing array, i.e., $s_0 \geq s_1 \ldots \geq s_l$. However, such a configuration has been shown instability for adaptive token-dropping inference [23]. Therefore, [23] utilize the sandwich rule and distillation from [58] to stabilize training and boost accuracy. But these two methods also significantly increase the training cost. Thus, such techniques cannot be applied to speed up the pretraining procedure.

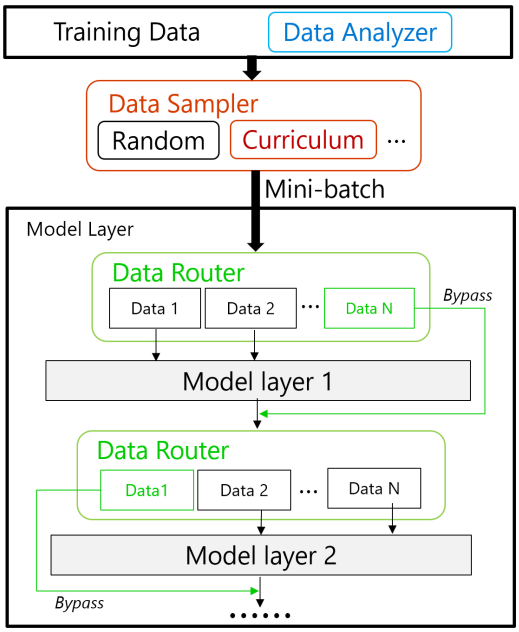

Figure 5: Design of the DeepSpeed Data Efficiency framework.

Recently, TokenBypass [21] enabled token dropping for BERT pretraining. It uses several importance scores/metrics to determine the dropped tokens (token frequency and cumulative loss). It proposed two main mechanisms to overcome the training instability issue: (1) the sandwich token dropping rule, where the first (L_1 to L_i) and the last few BERT layers (L_{l-j} to L_l) capture all tokens (no token dropping) and only bypass $s' \leq s$ tokens from L_i to L_{l-j} middle layers. Particularly, the authors (only) test on the encoder transformer (12-layer $\text{BERT}_{\text{base}}$ and 24-layer $\text{BERT}_{\text{large}}$), and let $i = l/2 - 1$, $j = 1$, $s' = s/2$. (2) special token treatment, where special tokens (e.g., `[MASK]`, `[CLS]`, `[SEP]`) are never dropped.

Compared to TokenBypass, our random-LTD (1) does not require importance score metric, special token treatment, or the sandwich token dropping rule, which dramatically reduces the manual design effort; (2) has been broadly tested on GPT-3/BERT pretraining tasks and GPT-2/ViT finetuning tasks, providing better data/training efficiency than TokenBypass.

A.2 Additional details for curriculum learning

Next, during training, the curriculum scheduler will determine the difficulty threshold for the current step based on a pacing function such as linear, rooted, or any strategy provided by user. Then the data sampler will sample the data with desired difficulty from the indexed data pool. To apply the proposed CL solution to a existing training pipeline, user just need to call an API and provide the raw training data, the difficulty metric index (computed in the offline analysis), and the pacing function configurations. Our framework will then provide a curriculum learing-based data loader that users can simply iterate at each step. Using our CL library for GPT-3/BERT pretraining, we are able to easily analyze and index the huge training data based on 7 difficulty metrics:

- **Truncation-based sequence length (seqtru), for GPT and BERT.** This metric starts with shorter data samples and gradually increases the sequence length during training. To change the sequence length, this metric will truncate the sequences (from the end of sequence) while keeping the number of samples unchanged, thus the number of tokens will decrease. This metric is recently applied to GPT-2 and GPT-3 models and demonstrate decent training efficiency gains [29].
- **Reshape-based sequence length (seqres), for GPT.** This metric is similar to seqtru metric, but instead of truncating we break the original sequences into segments based on the desired new sequence length. Thus we are essentially "reshaping" the input tensor into more samples and shorter lengths. This metric is proposed in MosaicML Composer as a variant of the seqtru metric [33], but their documentation does not describe which way provides better model quality. We don't apply the seqres to BERT case because unlike GPT data where all tokens are valid, BERT input sequences only include two natural sentences thus each sequence has different "effective sequence length" and then padded to 512. If we simply "reshape" BERT sequences, some of the new short sequences may only contain padding tokens.
- **Reorder-based sequence length (seqreo), for BERT.** This metric is similar to seqtru metric, but instead of truncating we adjust the sequence length by reordering the training data based on the "effective sequence length" in BERT training data sequences.
- **Vocabulary rarity (voc), for GPT and BERT.** This metric was proposed in a CL work for neural machine translation [36]. It computes the product of the unigram probabilities for each sequence by $-\sum_{k=1}^{N} log(p(w_k))$ where $p(w_k)$ is the vocabulary frequency (inside whole training data) of the kth word in the sequence. Lower value indicates that the sequence has more common vocabularies.
- **seqtru_voc, for GPT and BERT. seqres_voc, for GPT. seqreo_voc, for BERT.** These 3 metrics are combinations of above metrics. For seqtru_voc and seqres_voc, we first reorder the training data based on voc metric, then apply seqtru or seqres as a kind of post-processing. For seqreo_voc, we treat it as a single new metric and index the data based on it.

Besides the difficulty metrics, another set of CL hyperparameters is the pacing function: the start and end difficulty (d_s and d_e), total number of CL steps (T_c), and the kind of pacing function (linear, sqrt, or users can plug in any customized function to the proposed framework). For seqtru and seqres metrics, we set the d_s and d_e as value-based (e.g., $d_s = 80$, $d_e = 2048$) since the possible values of these two metrics are continuous. For other metrics, we set d_s and d_e as percentile-based (e.g., $d_s = 1\%$, $d_e = 100\%$) since the possible values of these metrics are discrete. For seqtru and seqres we use a linear pacing function ($d_t = d_s + (d_e - d_s) \times min(\frac{t}{T_c}, 1)$) following the preivous work [29], while for seqreo and voc we use a sqrt pacing function ($d_t = d_s + (d_e - d_s) \times min((\frac{t}{T_c})^{0.5}, 1)$). This is because seqreo and voc will only sample from a subset of data pool before reaching the end difficulty, and previous work finds that in such case it's beneficial to use a sqrt function to avoid sampling too much easy samples at the beginning [36]. Sec. A.4 includes low-cost tuning strategy and usage guidelines for our CL solutions.

A.3 Additional details for efficient data routing via random-LTD

Layers without Token Dropping. While TokenBypass needs to keep half of the layers in full sequence length training, random-LTD has no such limitation. Thanks to its attention-capture feature, we can apply random-LTD to most of the transformer layers except the first and last layers, enabling

further training efficiency gain. Our experiments show that keeping the first and last layers in full sequence length training usually leads to better performance since (1) the first layer directly connects to the embedding, and it can help refine the raw feature; (2) directly connected to the final prediction, the last layer provides a feature realignment for all tokens which could improve the model quality.

Monotonic Sequence Length Growth. In order to reduce the gradient variance introduced by random-LTD, we gradually increase the kept sequence length throughout training with a linear schedule (referred to as MSLG). Thus random-LTD has two hyperparameters similar to CL: starting from a sequence length r_s which denotes the size of kept token set K_i for each middle layer after dropping, random-LTD will gradually drop less tokens (following a linear function) and eventually stop dropping after T_r steps. Our experiments show that MSLG provides better model quality than constant drop schedule under similar data/compute savings. Sec. A.4 includes low-cost tuning strategy and usage guidelines for random-LTD.

A.4 Composing CL and random-LTD, tuning strategy, usage guidelines

CL and random-LTD are complementary: CL helps to sample the next data batch, and random-LTD helps to decide how to route each sampled data inside the model. DeepSpeed Data Efficiency hides several complexities when composing the two techniques so that users can easily enjoy the compound benefit. As one example, some CL metrics would affect the actual sample sequence length, thus inside our framework we make sure the random-LTD's token dropping mechanism is aware of this, and also adjust the calculation of number of actual consumed tokens which are affected by both techniques. This token consumption calculation is also critical to the learning rate schedule: previous CL work [29] finds that if a CL technique reduces the number of tokens on certain steps, it is desirable to use a learning rate decay schedule based on consumed tokens instead of consumed steps. This is because if baseline and CL use the same step-wise LR decay, it leads to much faster token-wise LR decay for CL which hurts model quality. In this work, we apply the token-based LR decay schedule for both CL and random-LTD. To our knowledge this is the first work to apply such LR schedule to token dropping/data routing techniques, and our experiments show that it does help improving random-LTD's performance. Our CL library's general data analyzer/sampler/loader and random-LTD's module wrapping design makes it easy to apply our framework to different model training tasks. And the overall composibility of DeepSpeed Data Efficiency enables us to leverage both data efficiency techniques and achieve even better data and training efficiency (Sec. B).

Table 2: CL and random-LTD usage guidelines.

Case	Guidelines
GPT-3 pretraining	CL: d_s = 80/1% (seqtru/voc), T_c = 40% of baseline's total steps random-LTD: r_s = 128, T_r = 70% of baseline's total steps
BERT pretraining	CL: d_s = 128/5% (seqtru/voc), T_c = 50% of baseline's total steps random-LTD: r_s = 128, T_r = 100% of baseline's total steps
GPT-2 finetuning	CL: d_s = 32 (seqres), T_c = 70% of baseline's total steps random-LTD: r_s = 128, T_r = 30% of baseline's total steps
ViT finetuning	random-LTD: r_s = 32/66, T_r = 80% of baseline's total steps

Tuning Strategy and Usage Guidelines. Both CL and random-LTD only have two parameters that need user tuning: the starting CL difficulty/random-LTD seqlen (d_s/r_s), and the total CL/random-LTD steps (T_c/T_r). [2] And for both CL and random-LTD we find that it's possible to apply a low-cost tuning strategy proposed in previous CL work [29], where we perform binary search on a very small portion (e.g., 2%) of training to find the smallest d_s/r_s and largest T_c/T_r that don't trigger substantial validation loss fluctuations ("whether the perplexity value becomes larger than 1.3x of the previous best perplexity"). For GPT-2 finetuning, given the low training cost we also perform full training of 16 different CL/random-LTD settings which confirm that (1) the low-cost tuning strategy is able to find very good hyperparameters; (2) both CL and random-LTD are not sensitive to hyperparameter choices. Tab. 2 summarizes the usage guidelines based on our tuning results, which we believe can be directly applied to any similar models (at least as a very good starting point for any further tuning).

B Evaluation

We evaluate DeepSpeed Data Efficiency by GPT-3/GPT-3 MoE/BERT pretraining and GPT-2/ViT finetuning. Appendix B.8 includes studies of the TokenBypass method on GPT finetuning and pretraining, further demonstrating the advantages of the proposed random-LTD method.

[2] For CL, the ending difficulty d_e is always the highest possible difficulty

Table 3: GPT-3 1.3B 0-shot evaluation results. The first column is the results of the original OpenAI GPT-3 1.3B model [7]. All the other columns are in the same order as the rows in main paper Tab. 7. OpenAI results are not directly comparable to ours because the training data are different.

Case	(0) OpenAI	(1) baseline	(2) CL seqtru	(3) CL seqres	(4) CL voc	(5) CL seqtru +voc	(6) CL seqres +voc	(7) rLTD	(8) CL seqtru +voc +rLTD	(9) baseline	(10) CL seqtru +voc	(11) rLTD	(12) baseline	(13) CL seqtru +voc	(14) rLTD	(15) CL seqtru +voc +rLTD
Train tokens	300B	300B	300B	300B	300B	300B	300B	300B	300B	200B	200B	200B	150B	150B	150B	150B
Avg.	47.9	42.5	43.4	43.0	42.3	43.6	43.0	43.7	43.8	41.9	42.7	43.1	42.0	42.6	42.7	42.8
(0) HellaSwag	54.7	51.9	52.3	52.4	51.8	52.7	52.2	54.1	54.3	50.9	52.0	52.9	49.9	50.6	51.6	52.1
(1) LAMBADA	63.6	62.0	61.2	61.7	60.6	61.9	61.1	62.9	62.3	59.8	61.4	62.3	59.5	59.6	61.3	61.7
(2) TriviaQA	19.7	7.0	7.91	7.63	6.66	7.65	6.07	7.9	7.55	6.15	6.46	7.54	5.9	7.2	6.37	7.42
(3) WebQs	4.63	1.38	1.62	2.07	2.56	1.38	2.02	3.15	2.17	2.46	1.67	2.31	1.03	2.26	2.66	3.2
(4) Winogrande	58.7	55.6	59.1	58.2	57.1	58.9	56.9	58.5	58.4	54.9	58.2	59.1	56.6	57.1	57.1	57.5
(5) PIQA	75.1	71.4	71.0	72.1	70.8	71.4	72.1	71.2	71.5	70.7	71.4	72.3	71.4	71.9	70.5	72.0
(6) ARC Challenge	35.5	29.4	29.6	29.3	28.8	30.1	28.9	28.7	30.1	28.5	28.2	29.7	27.2	27.0	28.7	27.6
(7) ARC Easy	53.8	53.7	54.3	55.0	54.0	55.2	55.0	54.4	56.4	53.5	53.2	52.7	52.7	53.7	54.1	54.0
(8) ANLI R1	33.4	31.6	33.3	30.7	33.4	33.5	31.6	33.0	31.6	31.6	29.8	31.9	33.0	32.9	32.1	33.7
(9) ANLI R2	33.3	33.7	33.8	32.8	33.0	33.3	32.9	32.5	31.5	30.4	33.2	34.8	31.8	33.9	34.6	33.6
(10) ANLI R3	33.4	33.1	35.2	33.5	33.2	33.3	33.9	33.4	35.2	33.7	35.8	35.3	32.4	34.8	34.9	35.0
(11) OpenBookQA	46.8	32.4	31.8	32.0	31.2	34.0	34.6	34.0	34.0	31.0	33.0	33.8	30.4	32.4	33.6	32.4
(12) RACE-h	40.9	35.2	34.2	35.7	35.3	35.3	34.3	35.4	36.4	34.6	33.9	35.0	34.3	34.2	34.6	34.9
(13) BoolQ	62.4	62.4	63.1	62.5	60.2	62.7	63.6	61.9	63.6	62.0	62.8	61.0	61.2	59.6	61.5	61.9
(14) Copa	77.0	72.0	70.0	75.0	72.0	73.0	77.0	76.0	75.0	71.0	74.0	73.0	72.0	75.0	71.0	71.0
(15) RTE	56.0	54.2	58.1	54.9	52.0	56.0	54.2	55.0	54.5	55.2	54.9	54.2	59.2	55.6	55.2	54.5
(16) WSC	61.5	36.5	42.3	36.5	34.6	43.3	36.5	43.3	40.4	36.5	37.5	36.5	36.5	36.5	37.5	36.5
(17) MultiRC	13.6	1.05	2.1	1.47	3.15	0.944	0.944	0.839	2.41	0.839	0.839	0.839	0.839	1.68	1.05	1.15
(18) ReCoRD	85.2	83.3	83.7	83.5	83.2	83.8	83.3	84.7	84.3	82.8	82.4	84.0	82.5	82.6	83.6	83.6

Table 4: GPT-3 1.3B 10-shot evaluation results. The first column is the results of the original OpenAI GPT-3 1.3B model [7]. All the other columns are in the same order as the rows in main paper Tab. 7. OpenAI results are not directly comparable to ours because the training data are different. Note that OpenAI used different number of shots for each task, while we use the same 10 shots for all tasks.

Case	(0) OpenAI	(1) baseline	(2) CL seqtru	(3) CL seqres	(4) CL voc	(5) CL seqtru +voc	(6) CL seqres +voc	(7) rLTD	(8) CL seqtru +voc +rLTD	(9) baseline	(10) CL seqtru +voc	(11) rLTD	(12) baseline	(13) CL seqtru +voc	(14) rLTD	(15) CL seqtru +voc +rLTD
Train tokens	300B	300B	300B	300B	300B	300B	300B	300B	300B	200B	200B	200B	150B	150B	150B	150B
Avg.	49.0	44.0	44.8	44.5	44.5	44.9	44.4	44.9	45.1	44.0	44.5	44.8	42.7	43.7	43.5	44.0
(0) HellaSwag	54.9	52.4	52.7	52.6	52.0	52.7	52.8	54.7	55.1	51.2	52.2	53.4	50.5	50.9	52.2	53.0
(1) LAMBADA	57.0	57.6	56.0	57.0	55.7	57.0	57.6	59.5	59.6	55.1	56.4	58.4	54.2	55.7	57.5	58.9
(2) TriviaQA	32.1	13.5	14.0	13.9	13.2	14.7	13.0	13.5	13.7	12.6	12.9	12.4	11.5	12.0	11.5	12.3
(3) WebQs	19.6	11.8	11.9	12.0	12.9	12.6	12.5	12.5	13.8	12.1	11.5	12.0	10.0	11.6	10.2	12.1
(4) Winogrande	59.1	57.4	56.7	58.9	58.2	60.0	58.2	58.7	58.1	55.9	59.2	59.0	56.8	58.0	58.4	58.4
(5) PIQA	74.3	71.5	71.4	71.5	71.4	71.5	72.3	71.6	72.6	71.1	72.0	71.9	71.2	71.7	71.4	71.4
(6) ARC Challenge	36.7	32.8	32.2	33.4	32.7	32.8	32.5	32.8	34.6	32.3	32.7	33.4	31.7	31.2	30.5	31.7
(7) ARC Easy	59.1	63.5	65.2	64.6	64.7	64.7	64.4	64.2	65.9	63.2	63.9	62.5	61.5	63.0	61.7	63.0
(8) ANLI R1	32.5	29.8	31.6	31.4	31.7	31.6	32.7	32.3	32.7	31.3	32.5	30.7	32.0	30.8	33.0	32.4
(9) ANLI R2	31.4	34.4	34.6	33.0	31.2	33.7	31.9	32.4	32.6	34.0	32.9	31.9	31.0	32.0	34.0	34.0
(10) ANLI R3	36.0	33.6	34.1	33.1	33.4	33.8	33.8	32.8	33.8	31.9	33.9	33.9	32.7	31.7	35.2	35.2
(11) OpenBookQA	50.6	32.4	34.0	34.6	34.0	35.4	35.2	33.6	32.6	33.0	33.2	33.2	33.4	33.4	32.2	29.8
(12) RACE-h	41.4	34.5	36.6	35.4	35.3	36.7	35.5	37.1	36.7	35.7	34.4	35.3	35.5	34.2	35.9	34.6
(13) BoolQ	64.1	60.8	63.5	59.4	63.1	62.1	63.1	64.2	64.0	62.8	62.1	63.8	58.8	63.4	58.2	62.0
(14) Copa	77.0	76.0	74.0	79.0	76.0	76.0	74.0	73.0	74.0	74.0	77.0	76.0	69.0	70.0	71.0	70.0
(15) RTE	50.9	48.0	55.2	50.5	53.8	52.7	49.1	53.1	52.0	56.0	54.5	55.6	48.0	56.0	48.4	51.2
(16) WSC	49.0	36.5	36.5	36.5	36.5	36.5	36.5	36.5	36.5	36.5	36.5	36.5	36.5	36.5	36.5	36.5
(17) MultiRC	20.8	5.88	7.24	5.35	6.93	5.77	5.98	6.19	5.35	4.51	5.67	6.72	4.51	6.19	5.67	6.4
(18) ReCoRD	84.0	83.0	83.4	83.3	82.4	83.6	83.2	84.6	84.0	82.3	82.7	83.9	82.2	82.4	83.8	83.3

B.1 GPT-3 and GPT-3 MoE pretraining

We use *the Pile* public dataset [16] to perform the pretraining of GPT-3 1.3B [7] (24 layers, 2048 hidden size, 16 attention heads) model. We also pretrain a GPT-3 Mixture-of-Experts (MoE) 6.7B model (24 layers, 1024 hidden size, 16 attention heads, 64 experts on every other layer) following related work [39]. We then perform 0-shot and 10-shot evaluations on 19 tasks to evaluate the model quality of the pretrained models. Detailed experimental setup is described in Appendix B.4.

Among the 5 CL difficulty metrics we have for GPT-3 model, to find out which metric provides the best model quality we pretrain the model (with 100% data) 5 times (each with 1 CL metric). For seqtru metric (to our knowledge the only metric previously applied to GPT-3 pretraining), we tune the CL hyperparameters d_s and T_c based on the tuning strategy proposed in previous work [29].

Table 5: GPT-3 1.3B 0-shot evaluation results when pretraining with 1%, 2%, 4%, 8%, 16%, and 32% of data.

Case	(1) baseline	(2) CL seqtru +voc +rLTD	(3) baseline	(4) CL seqtru +voc +rLTD	(5) baseline	(6) CL seqtru +voc +rLTD	(7) baseline	(8) CL seqtru +voc +rLTD	(9) baseline	(10) CL seqtru +voc +rLTD	(11) baseline	(12) CL seqtru +voc +rLTD
Model size	1.3B	1.3B	1.3B	1.3B	1.3B	1.3B	1.3B	1.3B	1.3B	1.3B	1.3B	1.3B
Train tokens	3B	3B	6B	6B	12B	12B	24B	24B	48B	48B	96B	96B
Avg.	34.5	35.0	36.3	36.8	37.2	38.4	38.8	40.2	39.8	41.2	41.5	42.2
(0) HellaSwag	28.7	29.3	30.8	33.2	35.4	38.1	39.0	42.7	43.5	46.9	47.8	49.9
(1) LAMBADA	28.9	32.0	38.0	41.4	43.5	49.5	50.3	53.9	54.3	58.0	57.8	60.4
(2) TriviaQA	1.18	1.4	1.58	1.56	1.79	1.89	2.28	3.91	3.5	4.82	6.29	6.16
(3) WebQs	0	0.148	0.443	0.738	1.03	0.935	0.984	0.984	1.08	2.36	2.21	2.51
(4) Winogrande	51.3	50.8	52.2	51.0	49.5	51.8	50.7	54.1	53.5	54.9	53.3	56.5
(5) PIQA	62.1	61.6	62.5	63.5	64.9	66.6	66.8	68.5	68.6	69.6	70.1	71.3
(6) ARC Challenge	22.2	22.9	24.9	23.0	24.7	24.6	24.1	26.2	26.7	26.6	28.5	28.2
(7) ARC Easy	38.8	38.4	40.5	41.0	44.1	45.2	46.4	47.7	48.6	50.7	51.2	52.7
(8) ANLI R1	33.3	33.3	32.6	33.3	31.5	31.5	31.7	32.7	33.2	33.7	33.4	33.0
(9) ANLI R2	33.2	34.6	35.8	32.7	31.7	32.8	32.6	33.6	33.1	34.0	34.1	34.4
(10) ANLI R3	32.8	33.9	35.4	32.9	34.4	34.9	35.4	34.5	32.2	35.1	33.7	33.5
(11) OpenBookQA	25.6	24.4	26.2	27.2	28.2	28.0	28.8	29.6	30.4	31.6	32.2	31.6
(12) RACE-h	27.1	28.5	28.9	29.4	30.0	31.2	32.2	32.5	31.8	33.5	34.5	35.2
(13) BoolQ	58.4	56.4	53.3	56.8	56.0	57.3	59.2	62.0	58.7	60.3	61.9	60.1
(14) Copa	61.0	64.0	66.0	71.0	68.0	69.0	70.0	72.0	69.0	69.0	70.0	71.0
(15) RTE	52.7	52.3	53.4	53.1	53.4	54.2	54.2	53.4	52.3	53.1	53.4	55.6
(16) WSC	36.5	36.5	39.4	36.5	36.5	36.5	36.5	36.5	36.5	36.5	36.5	36.5
(17) MultiRC	0.839	0.839	1.15	0.839	1.47	0.839	0.839	0.839	0.839	1.36	0.839	1.47
(18) ReCoRD	60.6	63.4	66.6	70.3	71.5	75.6	75.8	78.8	78.7	81.3	81.4	82.3

Table 6: GPT-3 1.3B 10-shot evaluation results when pretraining with 1%, 2%, 4%, 8%, 16%, and 32% of data.

Case	(1) baseline	(2) CL seqtru +voc +rLTD	(3) baseline	(4) CL seqtru +voc +rLTD	(5) baseline	(6) CL seqtru +voc +rLTD	(7) baseline	(8) CL seqtru +voc +rLTD	(9) baseline	(10) CL seqtru +voc +rLTD	(11) baseline	(12) CL seqtru +voc +rLTD
Model size	1.3B	1.3B	1.3B	1.3B	1.3B	1.3B	1.3B	1.3B	1.3B	1.3B	1.3B	1.3B
Train tokens	3B	3B	6B	6B	12B	12B	24B	24B	48B	48B	96B	96B
Avg.	33.9	35.0	35.6	36.6	37.3	38.8	38.8	40.7	40.7	42.3	43.0	43.2
(0) HellaSwag	28.9	29.5	31.3	33.2	35.2	38.2	39.3	43.1	43.6	47.0	47.9	50.3
(1) LAMBADA	24.5	27.5	32.2	36.2	37.6	44.9	44.0	50.7	47.0	53.2	51.8	57.0
(2) TriviaQA	0.804	1.36	1.75	3.05	3.21	4.93	5.27	6.96	7.51	9.45	10.6	11.0
(3) WebQs	1.08	1.72	2.17	2.9	3.44	5.22	4.87	6.94	7.73	8.66	10.4	11.4
(4) Winogrande	51.6	51.0	52.2	50.2	51.8	54.0	51.7	55.2	57.0	55.1	57.0	56.1
(5) PIQA	60.9	62.0	62.1	63.9	65.3	66.5	66.0	67.9	68.8	69.7	69.8	71.1
(6) ARC Challenge	21.9	23.2	24.0	24.3	24.8	24.9	26.5	27.7	28.0	29.8	31.5	32.1
(7) ARC Easy	38.7	41.9	44.9	47.1	50.0	52.4	54.1	55.6	56.4	59.8	60.6	62.5
(8) ANLI R1	31.7	33.5	33.4	32.8	34.1	32.6	35.2	33.0	31.6	33.7	33.0	31.2
(9) ANLI R2	33.1	35.0	30.3	34.7	35.6	34.4	34.2	31.0	33.6	34.4	32.4	32.5
(10) ANLI R3	33.9	34.8	35.1	33.2	33.5	34.2	33.4	33.2	34.5	33.2	34.2	32.8
(11) OpenBookQA	25.0	26.0	27.2	28.4	28.8	26.0	27.2	28.6	29.2	31.2	32.6	33.0
(12) RACE-h	26.9	27.8	29.1	28.9	29.1	30.5	32.3	31.9	32.0	34.3	34.4	35.0
(13) BoolQ	49.1	50.0	45.6	49.1	45.4	56.2	48.0	56.3	55.6	60.2	62.1	58.3
(14) Copa	62.0	66.0	70.0	66.0	69.0	67.0	71.0	70.0	66.0	70.0	72.0	72.0
(15) RTE	53.1	49.5	47.3	50.2	48.4	48.7	48.0	56.3	55.6	50.9	54.2	49.1
(16) WSC	36.5	36.5	36.5	36.5	36.5	36.5	36.5	36.5	36.5	36.5	36.5	36.5
(17) MultiRC	5.25	4.72	4.41	4.3	5.35	4.3	5.14	3.88	5.04	5.56	5.67	6.93
(18) ReCoRD	59.8	63.0	66.0	69.6	70.8	75.0	74.6	78.7	77.8	80.9	80.7	82.1

Then for other metrics we use the same hyperparameters without retuning for fair comparison. As presented in Tab. 7 case 1 to 6, results show that all 5 CL metrics provide better model quality than baseline (except (4)CL_voc's 0-shot accuracy), and the (5)CL_seqtru_voc provides the best quality. The extensibility of our general CL library enables us to easily apply different CL metrics to this large-scale model pretraining with huge training data, and identify a new CL metric that provides better model quality than existing solution (2)CL_seqtru. Next we pretrain the model with 67% data, comparing the baseline and the best CL metric we find. Results show that the average 0-shot evaluation accuracy drops from 42.5 to 41.9 when baseline use less data (Tab. 7 case 1, 9). On the other hand, our CL solution (case 10) with 67% data is able to achieve better 0-shot and 10-shot accuracy than baseline with 100% data, achieving a 1.5x data and time saving.

When applying the proposed random-LTD technique, results show similar benefit as CL: better model quality when using 100% data (Tab. 7 case 7), and 1.5x data/time saving while maintaining model quality (case 11). To explore whether composing CL and random-LTD could achieve even better data

Table 7: GPT-3 1.3B (case 1 to 15) and GPT-3 MoE 6.7B (case 16, 17) pretraining cost and average evaluation accuracy on 19 tasks. GPT-3 MoE only has 0-shot accuracy due to time constraints. Accuracy results for each single task can be found in Appendix B.4

Case	CL/ random-LTD hyperparameter	Data (billon tokens)	Time (hours on 64 V100)	Avg 0-shot accuracy	Avg 10-shot accuracy
(1)baseline	N/A	300 (1x)	260 (1x)	42.5	44.0
(2)CL_seqtru	$d_s = 80, T_c = 110K$	300 (1x)	257 (1.01x)	43.4	44.8
(3)CL_seqres	$d_s = 80, T_c = 110K$	300 (1x)	248 (1.05x)	43.0	44.5
(4)CL_voc	$d_s = 1\%, T_c = 110K$	300 (1x)	257 (1.01x)	42.3	44.5
(5)CL_seqtru_voc	same as (2) + (4)	300 (1x)	259 (1.00x)	43.6	44.9
(6)CL_seqres_voc	same as (3) + (4)	300 (1x)	248 (1.05x)	43.0	44.4
(7)random-LTD	$r_s = 128, T_r = 200K$	300 (1x)	263 (0.99x)	43.7	44.9
(8)CL_seqtru_voc +random-LTD	same as (5) + (7)	300 (1x)	260 (1.00x)	**43.8**	**45.1**
(9)baseline	N/A	200 (1.5x)	174 (1.49x)	41.9	44.0
(10)CL_seqtru_voc	seqtru: $d_s = 80, T_c = 73K$ voc: $d_s = 1\%, T_c = 73K$	200 (1.5x)	171 (1.52x)	42.7	44.5
(11)random-LTD	$r_s = 128, T_r = 133K$	200 (1.5x)	176 (1.48x)	43.1	44.8
(12)baseline	N/A	150 (2x)	130 (2.00x)	42.0	42.7
(13)CL_seqtru_voc	seqtru: $d_s = 80, T_c = 55K$ voc: $d_s = 1\%, T_c = 55K$	150 (2x)	129 (2.02x)	42.6	43.7
(14)random-LTD	$r_s = 128, T_r = 100K$	150 (2x)	131 (1.98x)	42.7	43.5
(15)CL_seqtru_voc +random-LTD	same as (13) + (14)	**150 (2x)**	**130 (2.00x)**	42.8	44.0
(16)baseline	N/A	300 (1x)	111 (1x)	42.8	
(17)CL_seqtru_voc +random-LTD	same as (5) + (7) but with 2x T_c and T_r due to batch size	300 (1x)	111 (1.00x)	**43.5**	

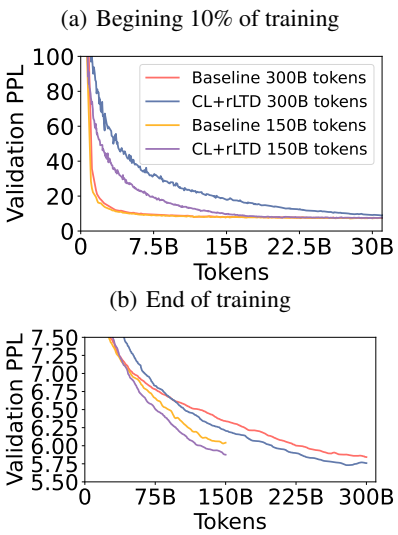

Figure 6: Validation perplexity during GPT-3 1.3B pretraining, comparing the baseline and the best DeepSpeed Data Efficiency solution under 100% and 50% training data.

and training efficiency, first we pretrain the model with both techniques under 100% training data. Results (case 5, 7, 8) show that using both techniques together further improves the model quality, demonstrating the benefit of composability by our framework. Next we pretrain the model with 50% data. Results (case 12 to 15) show that the baseline has worse 0-shot and 10-shot evaluation accuracy under 2x less data. Using CL or random-LTD can only recover part of the accuracy loss. On the other hand, the composed data efficiency solution is able to achieve the same or better accuracy results as baseline with 100% data, demonstrating a 2x data and 2x time saving.

To better understand how the proposed approach influences the model convergence, Fig. 6 plots the token-wise validation perplexity during pretraining. At the beginning of the training the proposed approach has slower convergence since we focus on easier/simpler data samples (CL) and drop more tokens (random-LTD) at the beginning. On the other hand, at the later stage of training the proposed approach is able to provide faster convergence speed than baseline. Our approach with 50% data is able to achieve similar final validation perplexity as baseline with 100% data (while baseline with 50% data cannot). Our approach with 100% data is able to achieve even better final validation perplexity which leads to the highest model quality.

As presented in Sec. 1 and Fig. 2, we also compare baseline and proposed work when using even less data during GPT-3 pretraining (Detailed accuracy results can be found in Appendix B.4). Results show that our approach provides better model quality at all cost budgets, advancing the whole cost-quality Pareto frontier. In particular, we achieve up to 12.5x data/time/cost saving while still maintaining 95% of the model quality (zero-shot eval accuracy) compared to the baseline with full data. Based on measured training time, this would be a cost reduction from $46.3K to $3.7K if renting similar hardware on Azure [2], greatly democratizing research and usage of foundation models.

Recent work shows that applying Mixture-of-Experts (MoE) to GPT-style model pretraining could lead to better training efficiency while reaching similar model quality [39]. Thus we also pretrain a GPT-3 MoE 6.7B model (350M base model, together with 64 experts on every other layer) to compare baseline and proposed work. Results show that MoE model does achieve similar model quality with less training cost (Tab. 7 case 1, 16). On the other hand, our approach can further improve MoE model's model quality (case 17), confirming its broad applicability.

B.2 BERT-large pretraining

We use *the Pile* public dataset [16] to perform the pretraining of BERT-large [14] (24 layers, 1024 hidden size, 16 attention heads) model. We then perform GLUE finetuning to evaluate the model quality of the pretrained models. Detailed experimental setup is described in Appendix B.5.

Table 8: BERT-large pretraining cost and GLUE finetuning score (median±std, details in Appendix B.5).

Case	CL/ random-LTD hyperparameter	Data (billon tokens)	Time (hours on 64 V100)	GLUE finetune score
(1)baseline	N/A	1049 (1x)	261 (1x)	87.29±0.53
(2)CL_seqtru	$d_s = 128, T_c = 960K$	1049 (1x)	265 (0.98x)	87.31±0.57
(3)CL_seqreo	$d_s = 5\%, T_c = 960K$	1049 (1x)	261 (1.00x)	87.48±0.61
(4)CL_voc	$d_s = 5\%, T_c = 960K$	1049 (1x)	261 (1.00x)	87.36±0.64
(5)CL_seqtru_voc	same as (2) + (4)	1049 (1x)	266 (0.98x)	87.60±0.34
(6)CL_seqreo_voc	same as (3) + (4)	1049 (1x)	262 (1.00x)	87.06±0.52
(7)random-LTD	$r_s = 128, T_r = 2M$	1049 (1x)	302 (0.86x)	**88.17±0.48**
(8)CL_seqtru_voc +random-LTD	same as (5) + (7)	1049 (1x)	290 (0.90x)	87.69±0.32
(9)baseline	N/A	703 (1.5x)	175 (1.49x)	87.19±0.49
(10)CL_seqtru_voc	seqtru: $d_s = 128, T_c = 640K$ voc: $d_s = 5\%, T_c = 640K$	703 (1.5x)	178 (1.47x)	87.29±0.62
(11)random-LTD	$r_s = 128, T_r = 1.34M$	703 (1.5x)	201 (1.3x)	87.99±0.38
(12)baseline	N/A	524 (2x)	131 (1.99x)	86.61±0.5
(13)CL_seqtru_voc	seqtru: $d_s = 128, T_c = 480K$ voc: $d_s = 5\%, T_c = 480K$	524 (2x)	133 (1.96x)	86.9±0.33
(14)random-LTD	$r_s = 128, T_r = 1M$	524 (2x)	150 (1.74x)	87.32±0.48
(15)CL_seqtru_voc +random-LTD	same as (13) + (14)	**524 (2x)**	**144 (1.81x)**	87.44±0.46

Table 9: GPT-2 finetuning on PTB results.

Case	Best PPL at seed 1234	Num. combinations surpass baseline	PPL median/std over 5 seeds
(1)baseline	16.077	N/A	16.077±0.028
(2)CL_seqtru	15.888	9 out of 16	
(3)CL_seqres	15.795	16 out of 16	**15.818±0.032**
(4)CL_voc	16.031	4 out of 16	
(5)CL_seqtru_voc	16.005	3 out of 16	
(6)CL_seqres_voc	15.981	8 out of 16	
(7)random-LTD	15.910	16 out of 16	15.948±0.040
(8)CL_seqres +random-LTD	15.831	N/A	15.831±0.014

Table 10: ViT finetuning results.

	CIFAR datasets on 24-layer ViT		
	Data saving	Top-1 (CIFAR100)	Top-1 (CIFAR10)
baseline	N/A	93.93±0.30	99.32±0.05
random-LTD	1.4x	94.02±0.40	99.30±0.03
	ImageNet datasets on 12-layer ViT		
	Data saving	Top-1	Top-5
baseline	N/A	84.65±0.04	97.41±0.02
random-LTD	1.3x	84.70±0.04	97.48±0.02

Similar to the GPT-3 case, for CL we first investigate which metric (among 5 metrics we have for BERT model) provides the best model quality by pretraining the model (with 100% data) 5 times. Tab. 8 case 1 to 6 results show that 4 CL metrics provide better model quality than baseline, and the (5)CL_seqtru_voc provides the best quality. Next we pretrain with 67% data, comparing the baseline and our best CL metric. Results show that the GLUE score drops from 87.29 to 87.19 when baseline use less data (case 1, 9). On the other hand, our CL solution (case 10) with 67% data is able to achieve on-par GLUE score as baseline with 100% data, achieving a 1.5x data and time saving.

Tab. 8 case 7, 11, 14 present the case when applying random-LTD only. In terms of data saving random-LTD performs better than CL: it is able to achieve better GLUE score even with 2x less data than baseline (case 14), greatly surpassing the 1.33x data saving by the state-of-the-art TokenBypass method. However, the time saving is less than data saving because the token dropping mechanism adds a computation overhead at each step. Because the BERT-large is a smaller model than GPT-3 1.3B, this fixed latency overhead has a larger relative impact to the training time. However, even with this overhead random-LTD is still a more data/time-efficient solution than baseline/TokenBypass.

Tab. 8 case 8 and 15 present the case when applying both CL and random-LTD. At 50% data, the composed solution further improves the GLUE score from the CL/random-LTD-only cases (case 15), achieving a 2x data and 1.8x time saving while maintaining the GLUE score compared to baseline. Another thing to note is that this case also has more time saving than the random-LTD-only case. This is because CL will first truncate the sequences before random-LTD perform the random token selection, and the shorter sequences reduces random-LTD's computation overhead. At 100% data, the composed solution (case 8) improves the GLUE score from the CL-only case, but is worse than the random-LTD-only case. One hypothesis is that for BERT pretraining when composing the two techniques it's preferable to reduce the CL duration, but exhaustively testing all hyperparameters is out of our resource budget and this work's scope.

B.3 GPT-2 and ViT finetuning

To verify the effectiveness of the proposed work on small-scale tasks, we apply our techniques to PTB finetuning task [30] for an already-pretrained GPT-2_{350M} model checkpoint from Huggingface. Given the much smaller training cost, we focus on improving the model quality under the same amount of data. Detailed experimental setup and hyperparameter tuning are described in Appendix B.6. As shown in Tab. 9, seqres provides the best model quality among the 5 CL metrics (case 3), unlike the two pretraining tasks where the seqtru_voc is the best metric. This is because this finetuning task has much smaller batch size and number of tokens per batch. seqtru will reduce number of tokens per batch, which is less desirable under small-batch training. The small batch also prevents the voc metric to include sufficient number of samples with different vocabulary rarity, limiting its benefit. Applying random-LTD also improves the model quality (case 7). Both CL best metric and random-LTD are able to surpass baseline on all 16 combinations of their hyperparameters, demonstrating that they are not sensitive to the hyperparameter choices. At last we try another 4 seeds for the baseline, CL best metric, random-LTD, and the CL+random-LTD case. The composed CL+random-LTD case (case 8) further improves model quality from random-LTD-only case, but is only on-par with CL-only

case. One hypothesis is that for tasks with such small-scale training data, it's less possible to further improve model quality by composing multiple data efficiency techniques.

We also try finetune the vision transformer (ViT) on both ImageNet (with a 12-layer pretrained ViT) and CIFAR (with a 24-layer pretrained ViT). Due to time/resource limitation, we only test random-LTD for this task. Detailed experimental setup is described in Appendix B.7. As presented in Tab. 10, results show that random-LTD is able to achieve 1.3-1.4x data savings while maintaining the model quality, demonstrating its broad applicability.

B.4 GPT-3 pretraining experimental setup and detailed results

For GPT-3 pretraining, we set some of the hyperparameters the same as the original OpenAI work [7]: seqlen 2K, batch size 512, learning rate 2e-4 (batch size 256 and learning rate 3e-4 for the GPT-3 MoE 6.7B model since we use 350M as the base model). We set other hyperparameters differently: (1) OpenAI pretrains GPT-3 on 300B tokens. To evaluate data efficiency techniques, we pretrain with 9 different total training tokens: 300B, 200B (67%), 150B (50%), 96B (32%), 48B (16%), 24B (8%), 12B (4%), 6B (2%), 3B (1%). (2) When using less than 300B training tokens, we increase the peak learning rate proportionally (e.g., 2x LR when using 50% data). This is similar to the traditional learning rate scaling when using different batch sizes. However, when using extremely small amount of data (e.g., 1% data), we find that using too larger learning rate (e.g., 100x) could lead to divergence. In such case we keep halving learning rate until the training succeed. (3) We do not use OpenAI's batch size warmup method because our GPT-3 125M model pretraining experiments show that it does not help on model quality under the same total training tokens. And the smaller batch sizes prevent us to pretrain on large number of GPUs at the beginning, which leads to longer training wall-clock time; (4) Since we don't use the batch size warmup, our training has more tokens at early steps. Thus we increase the linear learning rate warmup duration from OpenAI's 375M tokens to 3B tokens (except when using 3B tokens in total, where we use first 1.5B tokens for warmup); (5) OpenAI uses a single cycle cosine learning rate decay over 260B tokens, and the min learning rate is 10% of peak learning rate. However, based on our experiments and related works [57, 20], we changed the decay duration to always equal to total training token and the min learning rate to always equal to 1e-6, which provide better model quality. When calculating the total consumed training token, we take CL and random-LTD (which change number of tokens on certain steps) into consideration. For CL and random-LTD hyperparameters, we use the low-cost tuning strategy described in Sec. 2.

To evaluate the quality of pretrained GPT-3 models, we perform 0-shot and 10-shot evaluations on 19 tasks used by original OpenAI work: HellaSwag [59], LAMBADA [35], TriviaQA [22], WebQs [4], Winogrande [44], PIQA [5], ARC Challenge/Easy [11], ANLI R1/R2/R3 [34], OpenBookQA [32], RACE-h [27], BoolQ [10], Copa [1], RTE [12], WSC [28], MultiRC [56], and ReCoRD [60]. Since there is no additional training involved in 0/10-shot evaluations, it's impossible to try multiple seeds thus each task only has one accuracy result. We then take the average accuracy over the 19 tasks.

Tab. 3 and 4 present the 0-shot and 10-shot accuracy results for each task achieved by the pretrained GPT-3 1.3B models. Tab. 5 and 6 present the 0-shot and 10-shot accuracy results for the same GPT-3 1.3B model but pretrained with even less data as discussed in main paper Fig. 2, Sec. 1, and Sec. B.1. Tab. 11 presents the 0-shot accuracy results for each task achieved by the pretrained GPT-3 MoE 6.7B models, as discussed in main paper Sec. B.1.

B.5 BERT-large pretraining experimental setup and detailed results

For BERT-large pretraining, we set some of the hyperparameters the same as the Megatron-LM work [46] since it achieves better model quality than original BERT: seqlen 512, batch size 1024, learning rate 1e-4 with linear warmup up at first 10000 steps and then linearly decay to 1e-5. We set other hyperparameters differently: (1) Megatron-LM pretrains over 2M steps (1049B tokens). To evaluate data efficiency techniques, we pretrain with 3 different total training tokens: 1049B, 703B (67%), and 524B (50%). (2) When using less than 1049B training tokens, we increase the peak learning rate proportionally. (3) Megatron-LM decays the learning rate over 2M steps. Since our techniques could change the number of tokens at some steps, we change the decay to token-based and set the decay duration always the same as total training tokens. For CL and random-LTD hyperparameters, we use the low-cost tuning strategy described in Sec. 2.

Table 11: GPT-3 MoE 6.7B 0-shot evaluation results.

Case	(1) baseline	(2) CL seqtru +voc +rLTD
Model size	6.7B	6.7B
Train tokens	300B	300B
Avg.	42.8	43.5
(0) HellaSwag	53.0	53.3
(1) LAMBADA	60.1	59.6
(2) TriviaQA	11.0	9.31
(3) WebQs	2.95	2.31
(4) Winogrande	56.0	56.8
(5) PIQA	72.0	71.8
(6) ARC Challenge	28.9	28.9
(7) ARC Easy	54.5	54.2
(8) ANLI R1	33.6	30.8
(9) ANLI R2	32.8	34.1
(10) ANLI R3	33.6	35.5
(11) OpenBookQA	33.6	32.4
(12) RACE-h	33.8	35.0
(13) BoolQ	61.5	57.5
(14) Copa	71.0	74.0
(15) RTE	54.5	55.2
(16) WSC	36.5	51.0
(17) MultiRC	1.89	1.78
(18) ReCoRD	82.4	82.6

Table 12: BERT-large finetuning results. The first row is the results of the original BERT-large model [14]. All the other rows are in the same order as the rows in main paper Tab. 8. Original BERT results are not directly comparable to ours because the training data and total training token are different.

Case	Train tokens	Average	MNLI-m	MNLI-mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE
(0)original	43B	82.1	86.7	85.9	72.1	92.7	94.9	60.5	86.5	89.3	70.1
(1)baseline	1049B	87.29±0.53	88.54±0.16	89.25±0.13	92.1±0.07	94.12±0.15	94.33±0.48	64.36±1.59	90.43±0.21	89.32±0.81	83.2±1.16
(2)CL_seqtru	1049B	87.31±0.57	89.03±0.14	89.35±0.24	92.21±0.03	94.12±0.11	94.68±0.1	62.08±2.06	90.72±0.27	89.58±0.52	83.98±1.64
(3)CL_seqreo	1049B	87.48±0.61	88.81±0.16	89.27±0.19	92.2±0.12	93.99±0.28	94.79±0.42	62.86±1.85	90.51±0.25	89.32±0.85	85.55±1.34
(4)CL_voc	1049B	87.36±0.64	88.64±0.23	89.24±0.16	92.32±0.05	94.03±0.09	95.14±0.31	63.34±1.82	90.07±0.18	89.84±1.06	83.59±1.83
(5)CL_seqtru_voc	1049B	87.6±0.34	88.9±0.1	89.29±0.17	92.26±0.05	94.26±0.19	95.25±0.4	64.6±0.6	90.38±0.25	90.62±0.22	82.81±1.05
(6)CL_seqreo_voc	1049B	87.06±0.52	88.73±0.13	88.91±0.26	92.32±0.07	93.92±0.08	94.91±0.25	61.05±1.15	90.36±0.23	89.32±1.13	83.98±1.34
(7)random-LTD	1049B	88.17±0.48	88.74±0.25	89.18±0.21	92.27±0.1	94.32±0.21	95.02±0.38	67.3±1.5	90.65±0.15	90.1±0.63	85.94±0.89
(8)CL_seqtru_voc+random-LTD	1049B	87.69±0.32	88.79±0.13	89.26±0.04	92.34±0.08	94.21±0.23	95.14±0.36	65.46±0.68	90.44±0.19	89.58±0.56	83.98±0.59
(9)baseline	703B	87.19±0.49	88.75±0.18	89.11±0.19	92.13±0.08	93.99±0.16	95.14±0.46	62.07±1.44	90.08±0.31	89.84±0.68	83.59±0.87
(10)CL_seqtru_voc	703B	87.29±0.62	88.96±0.07	89.15±0.25	92.21±0.09	94.23±0.08	95.25±0.33	62.19±1.75	89.92±0.21	90.1±0.55	83.59±2.25
(11)random-LTD	703B	87.99±0.38	88.86±0.1	88.79±0.12	92.01±0.12	94.25±0.17	94.68±0.32	67.1±0.9	90.55±0.19	89.32±0.39	86.33±1.12
(12)baseline	524B	86.61±0.5	88.53±0.14	88.77±0.17	92.04±0.11	93.93±0.19	95.02±0.25	61.05±1.22	89.88±0.25	88.28±1.08	82.03±1.13
(13)CL_seqtru_voc	524B	86.9±0.33	88.66±0.14	89.25±0.21	92.08±0.05	93.99±0.26	95.02±0.17	63.34±0.52	89.96±0.25	88.54±0.22	81.25±1.14
(14)random-LTD	524B	87.32±0.48	88.81±0.15	88.9±0.13	91.96±0.04	94.28±0.14	94.91±0.43	64.41±1.32	90.39±0.25	89.06±0.18	83.2±1.67
(15)CL_seqtru_voc+random-LTD	524B	87.44±0.46	88.9±0.19	88.9±0.13	92.19±0.09	94.17±0.12	94.68±0.35	65.97±1.09	90.31±0.22	89.06±0.79	82.81±1.13

To evaluate the quality of pretrained BERT-large models, we finetune the models for 8 tasks from the GLUE benchmark [52]: MNLI, QQP, QNLI, SST-2, CoLA, STS-B, MRPC, RTE. We follow the finetuning hyperparameters from the original BERT work [14]: 3 epochs, batch size 32. For learning rate we test 5e-5, 4e-5, 3e-5, 2e-5 on the baseline and find that 3e-5 provides the best average GLUE score, thus we select LR=3e-5 for the comparison between baseline and proposed work. We perform finetuning on 5 seeds (1234 to 1238) and take the median/std on each task, then we take the average of the median scores as the average GLUE score, and take the average of std scores as the overall std.

Tab. 12 presents the finetuning results for each task achieved by the pretrained BERT-large models.

B.6 GPT-2 finetuning experimental setup

Due to the lack of published training recipe, we first perform a hyperparameter search for the baseline case (256 combinations of batch size, LR schedule, number of epochs). Then using the combination that provides best baseline validation perplexity, we apply CL and random-LTD (each with 16 different combinations of their two hyperparameters) to verify if they could further improve the model quality.

For GPT-2_{350M} finetuning on PTB [30], we use an already-pretrained GPT-2_{350M} model checkpoint and an example script [3] from Huggingface. Given the much smaller training cost (about 38min on a single V100 for 5 epochs), we focus on improving the model quality under the same amount of data. Due to the lack of published training recipe, we first perform a hyperparameter search for the baseline case: we tried 256 combinations of batch size (4, 8, 16, 32), learning rate (2e-5, 3e-5, 5e-5, 10e-5), learning rate warmup (0% and 10% linear warmup steps), learning rate decay (no decay, linear decay), and number of epochs (2, 3, 5, 10). For this sweep we only use one seed (1234) due to the number of combinations. Results show that the best combination among the 256 cases is: batch size 4, learning rate 10e-5, 0% learning rate warmup, linear learning rate decay, and 5 epochs. Results also show that for this task using more epochs (5 or 10) leads to better validation perplexity than less epochs (2 or 3).

Then using this combination that provides best baseline validation perplexity, we apply CL and random-LTD (each with 16 different combinations of their two hyperparameters) to verify if they could further improve the model quality. For CL we test 5 metrics (seqtru, seqres, voc, seqtru_voc, seqres_voc), each with 16 different combinations of its two hyperparameters: start difficulty d_s (8, 32, 128, 512 for seqtru/seqres, and 1%, 10%, 30%, 50% for voc) and total CL steps T_c (10%, 30%, 50%, 70% of the baseline's total steps). Results show that the seqres metric provides the best model quality, and its best hyperparameter combination is $d_s = 32, T_c = 70\%$ of baseline steps. For random-LTD we test 16 different combinations of its two hyperparameters: start seqlen r_s (8, 32, 128, 512) and total steps T_r (10%, 30%, 50%, 70% of the baseline's total steps). Results show that the best hyperparameter combination is $r_s = 128, T_r = 30\%$ of baseline steps. For CL+random-LTD composed case, we re-tuned the combination of T_c and T_r (CL will first adjust seqlen before random-LTD. To have a meaningful composition, it essentially requires $T_c < T_r$) and the best combination is $d_s = 32, r_s = 128, T_c = 10\%, T_r = 30\%$ of baseline steps. At last, for the best case of baseline, CL, random-LTD, and CL+random-LTD, we run another 4 seeds (1235 to 1238) and then calculate the median/std of the validation perplexity.

B.7 ViT finetuning experimental setup

We apply random-LTD to the vision transformer (ViT) [15] on finetuning tasks to demonstrate the broader applications of our method across different domains. We use the pretrained models published in [54] and test on two small image recognition benchmarks— CIFAR10 and CIFAR100 [26], and one large-scale dataset—ImageNet [13]. For ImageNet (CIFAR10/100), we use the 12-layer (24-layer) pretrained ViT with an input resolution 224×224 in which each patch of size 16×16 such that the sequence length becomes $196 + 1$ (the extra token is for position). ImageNet (CIFAR10/100) is trained on 8-GPU (1-GPU) and the batch size is 32 (128) images per GPU. The training budget for all three datasets is 14 epochs and a small constant learning rate is used based on grid search. Particularly, the best learning rate for ImageNet (CIFAR) is 5e-5 (1e-4). For ImageNet (CIFAR), when applying random-LTD the sequence length is started with 66 (32) and linearly reaches to the 197 full sequence length at 80% of the total baseline training iterations, equivalent to a 1.3x (1.4x) data saving.

B.8 Comparing random-LTD with the TokenBypass work

In main paper Sec. B.2 we demonstrate that random-LTD achieves 2x data saving while maintaining model quality for BERT pretraining, greatly surpassing the 1.3x data saving achieved by the state-of-the-art TokenBypass work [21]. In this section we provide additional discussion and evaluation to compare random-LTD with TokenBypass.

We include the illustration of the comparison between baseline, TokenBypass, and random-LTD in Fig. 7. First, the takeaway from TokenBypass can be summarized into (1) drop unimportant tokens starting from an intermediate layer of the model, (2) the dropping schedules is a fixed constant function (drop half of the tokens), and (3) the dropping criterion based on the "accumulated masked language modeling loss" (which is referred to as "token loss" since it needs each token's loss)

However, TokenBypass have several limitations (1) only tested on BERT pretraining (we find that it's less effective in GPT pretraining and finetuning), (2) the bypass layer starting only from an

[3] `https://github.com/huggingface/transformers/blob/main/examples/pytorch/language-modeling/run_clm_no_trainer.py`

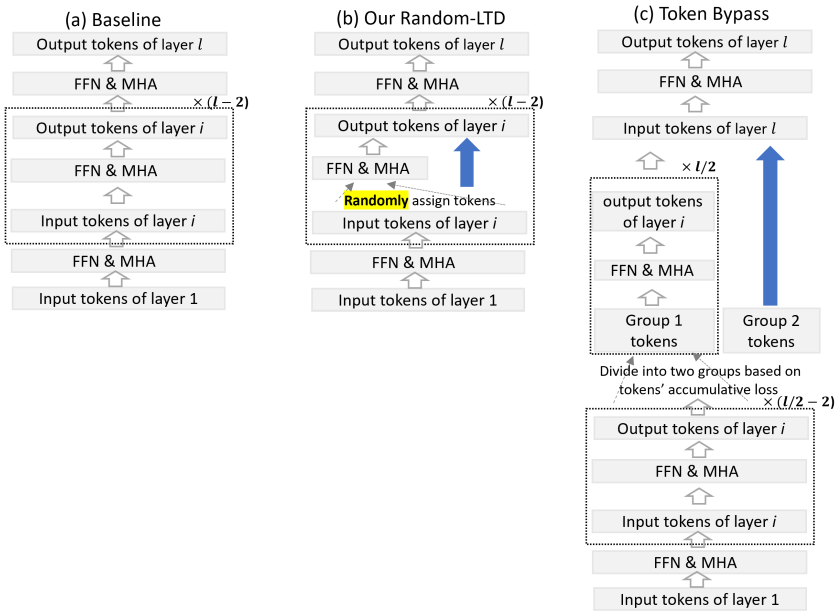

Figure 7: Illustration of the transformer model for the baseline training (left), TokenBypass training (right) and random-LTD training (middle). Compared to TokenBypass, random-LTD requires no criterion on the dropped tokens and trains well for all middle layers. The box with dash line is a repeated block. For both (a) and (b), the block is repeated by $l-2$ times, while for (c), the block is repeated by $l/2$. In the box, "Output tokens of layer i" is the same as "Input tokens of layer $i+1$".

Table 13: Comparing random-LTD (w/o MSLG) and TokenBypass under various constant dropping schedule. Baseline achieves a perplexity of 16.11±0.04.

Token saving ratio	1.88%	12.75%	23.72%	34.59%	45.45%	56.43%
random-LTD (w/o MSLG)	16.15±0.01	16.83±0.06	17.95±0.08	20.02±0.05	23.35±0.16	30.65±0.78
TokenBypass	16.4±0.04	17.3±0.06	18.59±0.19	23.09±0.23	28.56±0.24	35.91±0.26

intermediate layer (e.g., 6L for BERT-base), and (3) the dropping criterion based on "token loss" may not be accessible for some tasks, like classification problems.

Acknowledging that we are inspired by their excellent work and trying to solve their limitations, we believe random-LTD consists of three differences: (1) drop tokens starting from the 2nd layer of the model, (2) propose a linear increasing dropping schedule to close the training and inference discrepancy, and (3) the new random dropping criterion (which has lower overhead and can be easily applied to tasks without "token loss", such as vision transformer). Next, we provide more direct comparisons between random-LTD and TokenBypass on GPT-2 finetuning and GPT-3 pretraining tasks. Note that because this study was performed in parallel with other experiments, the hyperparameter choices are different from the experiments in main paper.

GPT-2 finetuning on PTB with various constant dropping schedule. To better demonstrate the benefit of random selection per layer, we provide a study with various constant dropping schedule. Particularly, from the second layer to the last second layer, we use one of the sequence lengths from 921, 819, 716, 614, 512, 409, of which the corresponding token saving ratio are shown in Tab. 13. We finetune GPT-2_{350M} (24 layers) on the PTB dataset with constant learning rate 5e-5 and Adam optimizer for 15 epochs (batch-size 8). The results are the best validations (average of three runs and one standard deviation) of random-LTD (without Monotonic Sequence Length Growth, MSLG) and TokenBypass.

As shown in Tab. 13, for all cases random-LTD has better performance than TokenBypass, even without one of the key contributions, Monotonic Sequence Length Growth (MSLG). This further verifies the conjecture we made in the main paper: "However, several works [50, 31, 51] have shown that MHA focuses on different tokens at different layer depths and the attention map aligns with the dependency relation most strongly in the middle of transformer architectures. Therefore, TokenBypass used in [21], i.e., fully skipping middle layers, may hinder the learnability/generalization of the architecture during pretraining/inference."

Table 14: Comparing random-LTD and TokenBypass (both with our proposed MSLG applied) under various token saving ratios. Baseline achieves a perplexity of 16.11±0.04.

Token saving ratio	8%	16%	24%	32%	40%	47%	52%	55%
random-LTD	15.91±0	15.86±0.06	15.86±0.01	15.85±0.02	16.05±0.06	17.02±0.05	18.41±0.04	20.01±0.06
TokenBypass (w/ MSLG)	16.1±0.02	16.09±0.05	16.21±0.03	16.54±0.01	17.06±0.04	18.64±0.04	23.12±0.22	25.77±0.57

Table 15: Comparing random-LTD and TokenBypass (both with our proposed MSLG applied) on GPT-3 pretraining.

	Validation loss
baseline	8.22
random-LTD (37.76% token saving)	8.26
TokenBypass (w/ MSLG, 37.76% token saving)	9.62

GPT-2 finetuning on PTB with our proposed MSLG. We are also curious if MSLG can help boost the performance of TokenBypass. Therefore, we also perform the comparison between random-LTD (with MSLG) and TokenBypass (with MSLG) on GPT-2 finetuning. We start at sequence length from 128 and linearly increase to full sequence 1024, with a different total steps to achieve different token saving ratios shown in Tab. 14. The rest of the hyperparameters are the same as the previous experiment.

Note that under MSLG it is hard to control the overall token saving ratio to be the same as constant dropping schedule case. But comparing Tab. 14's 24%/47%/55% with Tab. 13's 23.72%/45.45%/56.43%, we can clearly see the benefit of MSLG. Meanwhile, comparing the results of random-LTD and TokenBypass (with MSLG), it is clear that random-LTD still has better performance than TokenBypass for all cases. This shows that the other components of random-LTD, particularly the layerwise dropping mechanism, has its unique advantage over accumulated token loss for auto-regressive generative models.

GPT-3 pretraining. To directly compare the two techniques on pretraining tasks, we pretrain a GPT-3 350M model with 30B tokens. Due to limited time and resource, this is a smaller model and 10% of data compared to our other GPT-3 pretraining experiments. And due to the same reason we only compare the validation loss at the end of pretraining, but our experience shows that this metric has strong correlation with downstream task zero/few-shot evaluation performance. Based on the last GPT-2 finetuning experiment, here we again apply MSLG to TokenBypass. Results in Tab. 15 shows that under the same token saving ratio, random-LTD provides significantly better model quality than TokenBypass.

Other downstream tasks. TokenBypass cannot be easily extended to various downstream tasks. The reason is that the TokenBypass criterion is based on the "token loss", but downstream tasks, e.g., classification and regression (GLUE benchmark), do not have "token loss". Therefore, we did not find an easy way to apply TokenBypass on those tasks.