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, and Yuxiong He

Microsoft

{conglong.li, zheweiyao, xiaoxiawu, minjiaz, yuxhe}@microsoft.com

December 8, 2022

Abstract

Recent advances on deep learning models come at the price of formidable training cost. The increasing model size is one of the root cause, 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 library, a framework that makes better use of data, increases training efficiency, and improves model quality. Specifically, it provides efficient data sampling via curriculum learning, and efficient data routing via random layerwise token dropping. DeepSpeed Data Efficiency takes extensibility, flexibility and composability into consideration, so that users can easily utilize the framework to compose multiple techniques and apply customized strategies. By applying our solution to GPT-3 1.3B and BERT-Large language model pretraining, we can achieve similar model quality with up to 2x less data and 2x less time, or achieve better model quality under similar amount of data and time.

1 Introduction

Recently, large-scale deep learning models are empowering us to achieve more in many ways, such as improving programming efficiency by code generation [1] and providing art inspiration by text-to-image generation [2, 3]. To enable these services and keep improving the quality, deep learning model architecture evolves rapidly, and the model size is also growing at a tremendous speed. For example, from GPT to GPT-3 the model size increased 1500x in 2 years [4, 5]. The increasing model size leads to unprecedented training cost, which recently grows to 2 months on thousands of GPUs/TPUs [6, 7]. On the other hand, a less-emphasized perspective is that the training cost is proportional to both model scale and data scale, and recently data scale is increasing as fast as model scale. In Figure 1, we plot the model and data scales of several representative language models in the last 5 years. From the oldest model on the left to the

Code will be released soon as a part of `https://github.com/microsoft/DeepSpeed`.

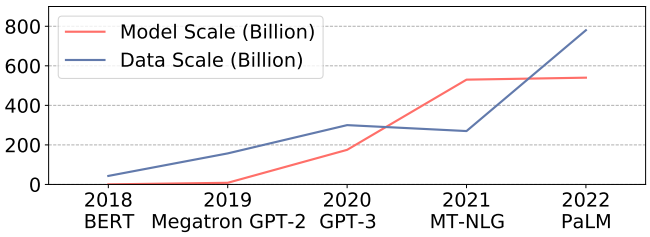


Figure 1: Model scale (number of parameters) and data scale (number of tokens consumed during training) of representative language models in the last 5 years [8, 9, 5, 6, 7].

newest models on the right, both the model and data scales increase at a similar speed. This demonstrates the importance of improving data efficiency: to achieve same model quality with less data and/or to achieve better model quality with the same amount of data.

Although there exist various techniques that improve data efficiency during training, applying them to large scale models still has many challenges in practice:

- **Lack of extensibility.** Techniques like Curriculum Learning improves data efficiency by indexing and sampling training data based on certain difficulty metric [10]. However, AI frameworks such as PyTorch only provides uniformly random data sampler, and it is difficult to find a general and extensible library to analyze, index, and sample large-scale training data based on arbitrary difficulty metric or even a combination of metrics.
- **Lack of flexibility.** Techniques like TokenBypass [11] provides data routing decisions and 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 using an importance score to determine which tokens to drop, and adding whitelist for special tokens (e.g., `[CLS]`). This could limit the possibility and benefit of applying it to other cases.
- **Limited composability.** 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.

To address these above challenges, we present DeepSpeed Data Efficiency library, a framework that makes better use of data, increases training efficiency, and improves model quality. Specifically, DeepSpeed Data Efficiency library demonstrates the following contributions:

- **Efficient data sampling via curriculum learning.** We present a general curriculum learning library which enables users to employ curriculum learning to their models at maximum extensibility: users can analyze, index, and sample their training data based on any custom difficulty metric or a combination of multiple metrics. Using this library, we are able to explore multiple metrics for GPT-3 and BERT pretraining, and identify the best metric (and the combination of metrics). This best metric provides up to 1.5x data and time saving while still achieving similar model quality compared to the baseline with full data. When using the same amount of data, curriculum learning with our best metric provides even better model quality than baseline.
- **Efficient data routing via random layerwise token dropping.** We employ a recently proposed technique called random layerwise token dropping (random-LTD), which skips the computation of a subset of the input tokens at all middle model layers [12]. Compared to previous work, random-LTD provides a more flexible data routing solution that can be easily applied to various model architecture and training tasks.
- **Seamlessly composing multiple methods.** The proposed DeepSpeed Data Efficiency

framework seamlessly composes the curriculum learning and random-LTD techniques, and only requires minimal changes on user code side. Furthermore, by composing both methods we can achieve even better data and training efficiency: for GPT-3 1.3B pretraining we can achieve up to 2x data and 2x time saving together with better or similar model quality as compared to the baseline training. When using the same amount of data, our approach further improves the model quality than baseline. Users can also extend and contribute to the library by adding additional data efficiency techniques (e.g., different metrics for curriculum learning and different data routing methods) to compose together.

2 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 for the learning process to focus on different kinds of data at different training stages, and one example here is the curriculum learning technique [10]. Curriculum learning 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 curriculum 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 fine-tuning tasks, such as neural machine translation (NMT) [13, 14, 15, 16, 17] and natural language understanding (NLU) [18, 19, 20, 21]. There are also a few works that explore curriculum learning for language model pretraining [22, 23, 24, 25]. However, one common limitation among existing works is that there does not exist a generalized and extensible curriculum learning library, which allows practitioners to easily apply custom curriculum difficulty metrics, the combination of metrics, and pacing functions. One evidence is that for the curriculum learning works for language model pretraining, most of them 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. TokenBypass [11] skips the compute of part of the input tokens at some middle layers during BERT pretraining, reducing pretraining cost by 25% while maintaining model quality. However, as mentioned in Section 1 TokenBypass includes many specialized implementations that limit its flexibility. More recently, random layerwise token dropping (random-LTD) [12] proposes to skip the computation of a subset of the input tokens at all middle layers. Compared to previous works, random-LTD does not require any importance score or any special token treatment, and it can be applied to all middle layers except the first and last layers. As a result, random-LTD demonstrates broad workload coverage (GPT/BERT pretraining and GPT/ViT finetuning) and good data/training efficiency improvement (up to 1.5x data and 1.3x time savings). Random-LTD is a quite flexible method applicable to various training tasks, but one limitation is that it has no control on the data batch itself, which is still uniformly sampled from the data pool at random. This motivates us to explore how to compose it with the efficient data sampling technique.

Another direction (that is less related to our work but still) related to data efficiency is Mixture-of-Experts (MoE) [26, 27, 28, 29, 30, 31] which is a novel model architecture that has parallel and sparsely activated model components called experts. During training, each data

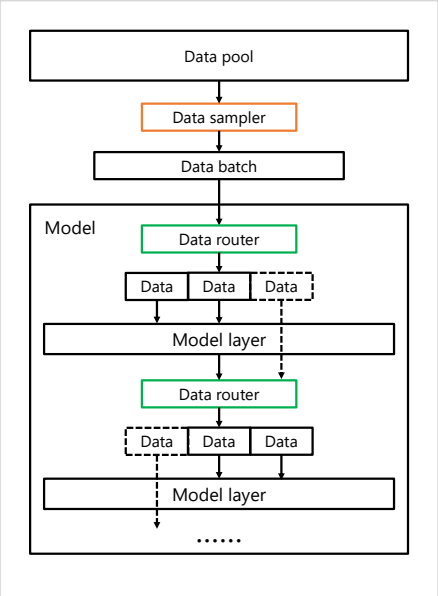


Figure 2: Design of the DeepSpeed Data Efficiency framework.

sample will only activate a subset of experts which leads to better training efficiency. However, one limitation of MoE is that it requires changing the model architecture which could lead to suboptimal model quality, and it could be difficult to make the architecture change (e.g., when finetuning a model that is already pretrained).

3 Design

At high-level, the proposed DeepSpeed Data Efficiency framework has two components as shown in Figure 2: 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. We will describe how we achieve so by a general curriculum learning library, and how it is different from existing curriculum learning solutions; Second we have efficient data routing, where instead of pass all input data to all model components, we aim to efficiently route each data through different components of model. We will describe how we achieve so by leveraging the recently proposed random layerwise token dropping (random-LTD) technique, and how we seamlessly compose it with curriculum learning. One thing to note is that curriculum learning and random-LTD are just one solution for efficient data sampling and routing, and we hope DeepSpeed Data Efficiency, with its extensibility/flexibility/composibility, could become a platform to host and compose more and more data efficiency techniques.

3.1 Efficient data sampling via curriculum learning

In Section 2 we discussed that existing curriculum learning works tend to focus on a small set of difficulty metrics and pacing functions, which may work well for limited tested workloads but has no guarantee on other cases. Furthermore, existing implementations are usually highly specialized, making it difficult for practitioners to easily apply these methods to their own workloads, and explore various customized difficulty metrics or pacing functions.

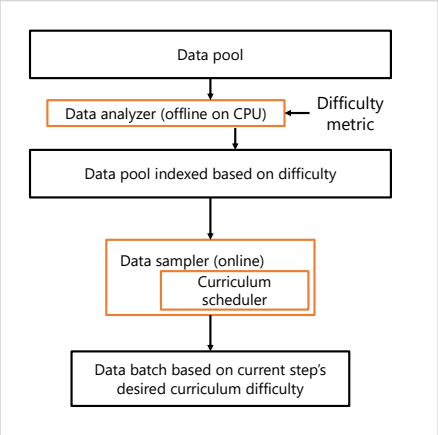


Figure 3: Design of the general curriculum learning library.

To solve these limitations, we design and implement a general curriculum learning library emphasizing the extensibility. It consists of three components as shown in Figure 3. 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 simple 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 hours for GPT-3 data or 80 hours for BERT data.

Next, during training, the curriculum scheduler will determine the difficulty threshold for the current step based 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 curriculum learning solution to a existing training pipeline, user just need to call an API and provide the raw training data, path to the difficulty metric index (computed in the offline analysis), and pacing function configurations. Based on those, the DeepSpeed Data Efficiency framework will provide a curriculum learing-based data loader that users can simply iterate at each step. Using this general and extensible curriculum learning solution for GPT and BERT-style model pretraining, we are able to easily analyze and index the huge training data based on up to 7 difficulty metrics and enable better data and training efficiency, which we will present in Section 4.

3.2 Efficient data routing via random layerwise token dropping

For efficient data routing, we choose to employ the recently proposed random-LTD technique due to its flexibility and broad workload coverage. To apply random-LTD to an existing training pipeline, user needs to provide some configurations for random-LTD (e.g., the dropping ratio to start with, and the pacing function of the dropping ratio at each step), together with the module class 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.

3.3 Composing curriculum learning and random-LTD

The curriculum learning and random-LTD techniques are complementary. During training, curriculum learning 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 enable/disable each technique. For example, some curriculum learning metrics would affect the actual sample sequence length, thus inside DeepSpeed Data Efficiency we make sure that 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. The composibility of DeepSpeed Data Efficiency enables us to leverage both data efficiency techniques and achieve even better data and training efficiency, which we will present in Section 4. Furthermore, this composibility also applies to data sampling and routing techniques in general, so that DeepSpeed Data Efficiency provides a platform to implement additional data efficiency techniques and compose them together.

4 Evaluation

In this section we evaluate the proposed DeepSpeed Data Efficiency framework by GPT-3 and BERT pretraining. We first evaluate the benefit of using the curriculum learning technique (efficient data sampling). Then we present the benefit of using both curriculum learning and random-LTD (efficient data routing) techniques.

4.1 Setup

Model and data. We use *the Pile* public dataset [32] to perform the pretraining of GPT-3 1.3B [5] (24 layers, 2048 hidden size, 16 attention heads) and BERT-Large [8] (24 layers, 1024 hidden size, 16 attention heads) models.

Hardware. For GPT-3 pretraining, all of the experiments are performed on 64 NVIDIA V100 GPUs (32GB memory). There are 4 nodes and 16 GPUs per node. For BERT-Large pretraining, all of the experiments are performed on 64 NVIDIA A100 GPUs (40GB memory). There are 8 nodes and 8 GPUs per node.

Pretraining hyperparameter. For GPT-3 pretraining, we set some of the hyperparameters the same as the original OpenAI work [5]: seqlen 2K, batch size 512, learning rate 2e-4. We set other hyperparameters differently: (1) OpenAI pretrains GPT-3 on 300B tokens. To evaluate data efficiency techniques, we pretrain with 3 different total training tokens: 300B, 200B (67%), and 150B (50%). (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 situation where we increase learning rate when increasing batch size. (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 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; (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 [33, 34], 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.

For BERT-Large pretraining, we set some of the hyperparameters the same as the Megatron-LM work [9] 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 2 different total training tokens: 1049B and 734B (70%). (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.

Curriculum learning (CL) hyperparameter. For GPT-3 and BERT-Large pretraining, we explore 7 CL difficulty metrics:

- **Truncation-based sequence length (seqtru), for GPT and BERT.** This metric start with shorter data samples and gradually increase 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 [35]. The original BERT pretraining recipe has two stages with sequence length 128/512, which is kind of similar to this metric but it is an abrupt length increase instead of continuous sequence length warmup.
- **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 [36], 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 is proposed in a work applying CL to neural machine translation [16]. 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. This value can be regarded as an approximate language model and it also implicitly incorporates the sequence length information (since longer sentences leads to product of more terms thus larger values).
- **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 reindex 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, etc.). For seqtru and seqres metrics, we set the d_s and d_e as value-based (e.g., $d_s = 80$ and $d_e = 2048$) since the possible values of these two metrics are continuous. For other metrics, we usually set d_s and d_e as percentile-based (e.g., $d_s = 1\%$ and $d_e = 100\%$) since the possible values of these metrics are discrete, but in some cases we found that it's beneficial to also set them as value-based. 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 [35], while for other metrics we use a sqrt pacing function ($d_t = d_s + (d_e - d_s) \times min((\frac{t}{T_c})^2, 1)$). This is because all other metrics 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 [16]. d_s and T_c are the only two parameters that need tuning (d_e is always the highest possible difficulty). For some cases, we follow the low-cost tuning strategy proposed in previous work [35], where we perform binary search on a very small portion of early training. For remaining cases, we simply use the same set of hyperparameters or scale T_c proportionally based on the total training token ratio.

Random-LTD hyperparameter. We follow the hyperparameter settings in the original random-LTD work [12], which is similar to CL: starting from a sequence length r_s which denotes how many tokens left 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. When using 100% training data, we simply use the same r_s and T_r as the original work. When using less data, we reduce the T_r proportionally.

Model quality evaluation. To evaluate the quality of pretrained GPT-3 models, we perform 0-shot and 10-shot evaluations on 19 tasks: HellaSwag [37], LAMBADA [38], TriviaQA [39], WebQs [40], Winogrande [41], PIQA [42], ARC Challenge/Easy [43], ANLI R1/R2/R3 [44], OpenBookQA [45], RACE-h [46], BoolQ [47], Copa [48], RTE [49], WSC [50], MultiRC [51], and ReCoRD [52]. To evaluate the quality of pretrained BERT-Large models, we finetune the models for 4 tasks: MNLI [53], QQP [54], and RACE (middle and high difficulty) [46]. We follow the fine-tuning hyperparameters from the Megatron-LM work [55]: for MNLI/QQP/RACE we finetune with 10/12/3 epochs, batch size 128/128/32, and learning rate 1e-5/5e-5/2e-5.

4.2 Evaluation of curriculum learning (CL)

GPT-3 pretraining. Among the 5 CL difficulty metrics we have for GPT-3 1.3B model, to find out which metric provides the best model quality we pretrain the model (with 100% data) 5 times (each time with 1 CL metric). For seqtru metric (the only metric that was previously applied to GPT-3 model pretraining to our knowledge), we tune the CL hyperparameters d_s and T_c based on the low-cost tuning strategy proposed in previous work [35]. Then for other metrics we use the same hyperparameters without retuning for fair comparison. For voc metric the d_s is changed to percentile-based (1%) as explained in Section 4.1. After pretraining we compare the model quality by 0-shot and 10-shot evaluation accuracy as presented in Table 1 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 swiftly 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).

Table 1: GPT-3 1.3B pretraining cost and average evaluation accuracy on 19 tasks. Accuracy results for each single task can be found in Appendix A.1

Case	CL/rLTD hyperparameter	Pretrain data (billon tokens)	Pretrain 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)rLTD	$r_s = 128$, $T_r = 200K$	300 (1x)	263 (0.99x)	43.7	44.9
(8)CL_seqtru_voc+rLTD	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)baseline	N/A	150 (2x)	130 (2.00x)	42.0	42.7
(12)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
(13)rLTD	$r_s = 128$, $T_r = 100K$	150 (2x)	131 (1.98x)	42.7	43.5
(14)CL_seqtru_voc+rLTD	same as (12) + (13)	**150 (2x)**	**130 (2.00x)**	42.8	44.0

All 5 CL cases have similar total training time as the baseline since they all use 300B tokens training data, although the seqres metric in cases 3 and 6 provides a slight time saving. This is because seqres change long sequences into more but shorter sequences, which helps reducing computation time because inside each Transformer block the self-attention and intermediate layers have time complexity of $O(B \times L^2 \times H)$ and $O(B \times L \times H^2)$, respectively (B, L, H represent batch size, sequence length, hidden size). The seqtru metric has less such benefit because it has less tokens per step than seqres due to truncation instead of reshape. Thus it requires more steps to reach the same 300B tokens which takes more time.

Next we pretrain the model with 67% data, comparing the baseline and the best CL metric we find (seqtru_voc). Results show that the average 0-shot evaluation accuracy drops from 42.5 to 41.9 when baseline use less data (Table 1 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.

BERT pretraining. Among the 5 CL difficulty metrics we have for BERT-Large model, to find out which metric provides the best model quality we pretrain the model (with 100% data) 5 times (each time with 1 CL metric). For seqtru metric (the only metric that was previously applied to GPT-3 model pretraining to our knowledge), we tune the CL hyperparameters d_s and T_c based on the low-cost tuning strategy proposed in previous work [35]. Then for other metrics we use the same hyperparameters without retuning for fair comparison. For seqreo and voc metrics the d_s is changed to percentile-based (5%). After pretraining we compare the model quality by finetuning accuracy as presented in Table 2 case 1 to 6. Results show that all 5 CL metrics provide better model quality than baseline, and the (4)CL_voc provides the best quality. One thing we notice about this metric is that the last 5% most difficult data have a much larger abosolute value change: the first 95% data has vocabulary rarity up to 3987, while the last 5% has vocabulary rarity up to 9069. This motivates us to try the case with the same CL metric but use value-based pacing function instead of percentile based. And the result (case 7) shows that it does provide better model quality (than case 4). All 5 CL cases have similar yet slightly higher total training time than the baseline. This is because curriculum learning requires extra indexing and sampling overhead than uniform sampling, which is an fixed latency regardless of the model size (about 0.005 ms per step in our GPT/BERT cases). 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 in BERT-Large case. The seqtru metric (in case 2, 5) requires longest time because it has less tokens per step due to truncation. Thus it requires more steps to reach the same tokens. On the other hand, we believe that when the goal is to maximize model quality using all data, it's still beneficial to employ curriculum learning (with the voc metric) and take

Table 2: BERT-Large pretraining cost and average finetuning accuracy on 4 tasks. Accuracy results for each finetuning task can be found in Appendix A.2

Case	CL/rLTD hyperparameter	Pretrain data (billon tokens)	Pretrain time (hours on 64 A100)	Avg finetune accuracy
(1)baseline	N/A	1049 (1x)	141 (1x)	85.42±0.31
(2)CL_seqtru	d_s = 128, T_c = 960K	1049 (1x)	153 (0.92x)	85.77±0.28
(3)CL_seqreo	d_s = 5%, T_c = 960K	1049 (1x)	144 (0.98x)	85.46±0.33
(4)CL_voc	d_s = 5%, T_c = 960K	1049 (1x)	144 (0.98x)	85.91±0.24
(5)CL_seqtru_voc	same as (2) + (4)	1049 (1x)	152 (0.93x)	85.8±0.27
(6)CL_seqreo_voc	same as (3) + (4)	1049 (1x)	144 (0.98x)	85.61±0.32
(7)CL_voc value-based	d_s = 600, T_c = 960K	1049 (1x)	144 (0.98x)	**86.13±0.27**
(8)CL_voc	d_s = 5%, T_c = 700K	734 (1.4x)	101 (1.40x)	85.02±0.29
(9)CL_voc value-based	d_s = 600, T_c = 700K	**734 (1.4x)**	**101 (1.40x)**	85.59±0.25

the 2% extra training time.

Next we pretrain the model with 70% data, comparing the baseline and the best CL metric we find (voc). For CL we again test both percentile-based and value-based pacing function. Results show that the value-based CL again provides better model quality, and it matches the model quality of baseline with 100% data (case 8, 9). This demonstrates a 1.4x data and time saving by CL.

4.3 Evaluation of composed DeepSpeed Data Efficiency solution

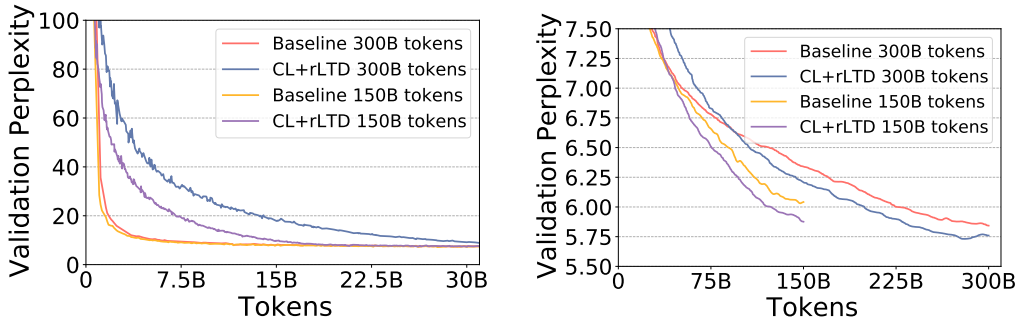
GPT-3 pretraining. To explore whether composing CL and random-LTD could achieve even better data and training efficiency, first we pretrain the model (same 100% training data) with CL or random-LTD only and with both techniques composed together. Results (Table 1 case 5, 7, 8) show that using either of the technique it's possible to achieve better model quality than baseline, but when composed together it's possible to further improve the model quality. This demonstrate the composability strength of the proposed DeepSpeed Data Efficiency framework. The random-LTD only case has 1% higher training time than baseline because the token dropping mechanism introduces a small computation overhead.

Next we pretrain the model with 50% data. Results (Table 1 case 11 to 14) 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, Figure 4 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.

5 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 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



(a) Begining of training (first 30B tokens)

(b) End of training

Figure 4: Validation perplexity during GPT-3 1.3B pretraining, comparing the baseline and the best DeepSpeed Data Efficiency solution under 100% and 50% training data.

different data efficiency techniques together. This enables us to achieve a 2x data and time saving for GPT-3 and BERT pretraining, or to achieve even better data quality under similar data and time. We hope this work could motivate more research on data efficiency, and we hope DeepSpeed Data Efficiency could become a data efficiency platform that accelerate these research.

References

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

[2] 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.

[3] 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.

[4] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] Zhewei Yao, Xiaoxia Wu, Conglong Li, Connor Holmes, Minjia Zhang, Cheng Li, and Yuxiong He. Random-ltd: Random and layerwise token dropping brings efficient training for large-scale transformers. *arXiv preprint arXiv:2211.11586*, 2022.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

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

[25] 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.

[26] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[27] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

[28] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

[29] 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 Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 18332–18346. PMLR, 17–23 Jul 2022.

[30] Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. *arXiv preprint arXiv:2112.06905*, 2021.

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

[32] 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.

[33] 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.

[34] 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.

[35] Conglong Li, Minjia Zhang, and Yuxiong He. Curriculum learning: A regularization method for efficient and stable billion-scale gpt model pre-training. *arXiv preprint arXiv:2108.06084*, 2021.

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

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

[54] Shankar Iyer, Nikhil Dandekar, and Kornl Csernai. First quora dataset release: Question pairs, 2017. *URL https://data. quora. com/First-Quora-Dataset-Release-Question-Pairs*, 2017.

[55] 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.

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 [5]. All the other columns are in the same order as the rows in main paper Table 1. OpenAI results are not directly comparable to ours because the training data are different.

Case Train tokens	(0) OpenAI 300B	(1) baseline 300B	(2) CL seqtru 300B	(3) CL seqres 300B	(4) CL voc 300B	(5) CL seqtru +voc 300B	(6) CL seqres +voc 300B	(7) rLTD 300B	(8) CL seqtru +voc +rLTD 300B	(9) baseline 200B	(10) CL seqtru +voc 200B	(11) baseline 150B	(12) CL seqtru +voc 150B	(13) rLTD 150B	(14) CL seqtru +voc +rLTD 150B
Avg.	47.9	42.5	43.4	43.0	42.3	43.6	43.0	43.7	43.8	41.9	42.7	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	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	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	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	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	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	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	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	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	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	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	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	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	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	61.2	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	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	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	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	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	82.5	82.6	83.6	83.6

A Appendix

A.1 GPT-3 1.3B evaluation results

Table 3 and 4 present the 0-shot and 10-shot accuracy results for each task achieved by the pretrained GPT-3 1.3B models.

A.2 BERT-Large finetuning results

Table 5 presents the finetuning results for each task achieved by the pretrained BERT-Large models.

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 [5]. All the other columns are in the same order as the rows in main paper Table 1. 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 Train tokens	(0) OpenAI 300B	(1) baseline 300B	(2) CL seqtru 300B	(3) CL seqres 300B	(4) CL voc 300B	(5) CL seqtru +voc 300B	(6) CL seqres +voc 300B	(7) rLTD 300B	(8) CL seqtru +voc +rLTD 300B	(9) baseline 200B	(10) CL seqtru +voc 200B	(11) baseline 150B	(12) CL seqtru +voc 150B	(13) rLTD 150B	(14) CL seqtru +voc +rLTD 150B
Avg.	49.0	44.0	44.8	44.5	44.5	44.9	44.4	44.9	45.1	44.0	44.5	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	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	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	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	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	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	71.2	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	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	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	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	31.0	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	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	33.4	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	35.5	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	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	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	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
(17) MultiRC	20.8	5.88	7.24	5.35	6.93	5.77	5.98	6.19	5.35	4.51	5.67	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	82.2	82.4	83.8	83.3

Table 5: BERT-Large finetuning results. The first row is the results of the original Megatron-LM BERT-Large model [9]. All the other rows are in the same order as the rows in main paper Table 2. Megatron-LM results are not directly comparable to ours because the training data are different, and because Megatron-LM reports median while we report mean and std.

Case	Train tokens	MNLI-m (dev)	MNLI-mm (dev)	QQP (dev)	RACE-m (dev)	RACE-m (test)	RACE-h (dev)	RACE-h (test)	Average
(0)Megatron-LM	1049B	89.7	90.0	92.3	N/A	86.9	N/A	81.5	N/A
(1)baseline	1049B	89.09±0.05	89.52±0.23	92.29±0.13	84.58±0.28	82.88±0.63	80.54±0.32	79.01±0.51	85.42±0.31
(2)CL_seqtru	1049B	89.15±0.36	89.65±0.1	92.25±0.06	84.42±0.26	83.79±0.51	81.39±0.41	79.72±0.29	85.77±0.28
(3)CL_seqreo	1049B	89.08±0.2	89.29±0.04	92.29±0.06	84.03±0.43	83.14±0.85	81.2±0.36	79.18±0.38	85.46±0.33
(4)CL_voc	1049B	89.16±0.11	89.55±0.17	92.23±0.06	84.75±0.29	83.96±0.29	81.69±0.27	80.0±0.48	85.91±0.24
(5)CL_seqtru_voc	1049B	89.19±0.13	89.5±0.06	92.37±0.04	85.18±0.76	83.49±0.51	81.12±0.23	79.77±0.16	85.8±0.27
(6)CL_seqreo_voc	1049B	89.07±0.16	89.25±0.12	92.3±0.1	84.44±0.47	83.57±0.76	80.94±0.23	79.7±0.39	85.61±0.32
(7)CL_voc value-based	1049B	89.28±0.08	89.44±0.13	92.29±0.05	85.42±0.36	84.0±0.61	81.83±0.39	80.65±0.3	86.13±0.27
(8)CL_voc	734B	88.89±0.13	89.4±0.11	92.35±0.04	82.96±0.6	83.1±0.44	79.99±0.44	78.46±0.25	85.02±0.29
(9)CL_voc value-based	734B	89.06±0.15	89.33±0.05	92.36±0.06	84.47±0.24	83.26±0.6	81.44±0.44	79.24±0.18	85.59±0.25