Cramming: Training a Language Model on a Single GPU in One Day

Jonas Geiping [1] **Tom Goldstein** [1]

Abstract

Recent trends in language modeling have focused on increasing performance through scaling, and have resulted in an environment where training language models is out of reach for most researchers and practitioners. While most in the community are asking how to push the limits of extreme computation, we ask the opposite question: How far can we get with a single GPU in just one day?

We investigate the downstream performance achievable with a transformer-based language model trained completely from scratch with masked language modeling for a *single* day on a *single consumer* GPU. Aside from re-analyzing nearly all components of the pretraining pipeline for this scenario and providing a modified pipeline with performance close to BERT, we investigate why scaling down is hard, and which modifications actually improve performance in this scenario. We provide evidence that even in this constrained setting, performance closely follows scaling laws observed in large-compute settings. Through the lens of scaling laws, we categorize a range of recent improvements to training and architecture and discuss their merit and practical applicability (or lack thereof) for the limited compute setting.

1. Scaling Up and Scaling Down

Large-scale training of machine learning models with transformer architectures has lead to ground-breaking improvements in many sub-fields of natural language processing including language understanding and natural language generation (Vaswani et al., 2017; Dosovitskiy et al., 2021; Radford et al., 2019). The nowadays accepted (but historically surprising) key behavior of these systems is that they reliably *scale* – they continuously improve in performance when the number of model parameters and amount of data grow. These increases in performance are well-described by various power laws as studied by Kaplan et al. (2020). This sets up a dominant paradigm in which scaling is the key to performance improvement (Sutton, 2019).

Our goal is to turn this trend on its head and investigate how to best *scale down* language model training and what trade-offs emerge when doing so: *What downstream performance can be achieved by a modest researcher when training from scratch with a single GPU for a single day?* The ability to train a language model to the performance level of BERT with such modest resources has several interesting implications. For one, if scaled-down model pretraining is a viable analogue of large-compute pretraining, then this opens up a host of further academic investigations that are currently hard to realize for large-scale models. For example, research questions about the differences between existing and new pre-training tasks, tracing model predictions to data points (Ilyas et al., 2022), security questions such as membership inference (Carlini et al., 2022) and data poisoning (Geiping et al., 2021), and a wide range of empirical investigations into topics such as stability or generalization that arise during training (Nagarajan & Kolter, 2019; Jiang et al., 2019). At the same time, we can imagine situations in which legal requirements make it unclear whether models trained on public data with uncertain origin are permissible, and where a practitioner is interested in retraining their language models using a specialized or trustworthy data source (Wilka et al., 2017; Gold & Latonero, 2017).

To answer these questions, we consider a challenge we call "Cramming" – learning a whole language model the day before the test. Our studies begin by investigating many facets of the training pipeline to see which modifications actually improve performance in the scaled-down scenario. We provide evidence that even in this constrained setting, performance closely follows scaling laws observed in large-compute settings (Kaplan et al., 2020). An unsurprising consequence of these laws is that scaling down is hard; while smaller model architectures enable speeding up gradient computations, overall rates of model improvement over time remain nearly constant. Nonetheless, we can find changes to the training recipe that exploit scaling laws to yield improvements by improving the effective rate of gradient computations without compromising model

[1]Dep. of Computer Science, University of Maryland, College Park. Correspondence to: Jonas Geiping <jgeiping@umd.edu>.

Work presented at the ES-FoMo Workshop at ICML 2023., Honolulu, Hawaii, USA.

Table 1. Maximal Throughput available for select training runs of large language models. FLOP Counts for BERT reproductions and related models. Large-scale LMs included only for reference. `rtxa4000` compute estimated. See Appendix G.1 for details.

Group	Target	Accelerator	Time Limit	Total exaFLOP
(Devlin et al., 2019)	BERT	16 `TPU`	4 days	680
(Dettmers, 2018)	BERT	8 `V100`	11 days	950
(Narasimhan, 2019)	BERT-large	1472 `V100`	47 min	519
(Raffel et al., 2020)	T5-base	16 `TPUv3`	1 day	170
(Iandola et al., 2020)	squeezeBERT	8 `Titan RTX`	4 days	361
(Narang et al., 2021)	T5 variations	16 `TPUv3`	1.75 days	298
(Tay et al., 2021)	T5-small-L16	16 `TPUv3`	11.2 hours	82
(Izsak et al., 2021)	BERT variation	8 `V100`	1 day	86
(Liu et al., 2019)	roBERTa-base	1024 `V100`	1.25 day	13 824
(Chowdhery et al., 2022)	PaLM	6144 `TPUv4`	50 days	7 299 072
Our Setup 1	BERT variation	1 `rtx2080ti`	1 day	5
Our Setup 2	BERT variation	1 `rtxa4000`	1 day	8*
Our Setup 3	BERT variation	1 `rtxa6000`	1 day	13

size. In the end, we are able to train models that achieve respectable performance – often close to and sometimes exceeding BERT on GLUE tasks – on a shoestring budget.

2. Tying our hands behind our back: A setup with limited compute

Before we start this investigation, we want to outline the extent of limitations we are interested in. The rules for cramming are as follows:

- A transformer-based language model of arbitrary size is trained with masked-language modeling, completely from scratch.
- Existing pretrained models cannot be included in any part of the pipeline.
- Any raw text (excluding downstream data) can be included for training. This means that one can achieve speedups by making judicious choices about how and when to sample data, provided the sampling mechanism does not require a pre-trained model.
- The downloading and pre-processing of raw data is exempted from the total compute budget. Pre-processing may include CPU-based tokenizer construction, tokenization, and filtering, but cannot include representation learning (e.g. pre-training a word embedding is not allowed, unless it is counted towards the final runtime).
- Training proceeds on a single GPU for 24 hours.
- Downstream performance is evaluated on GLUE (Wang et al., 2018). Downstream finetuning on GLUE is limited to brief training with only the training data of the downstream task (we consider 5 epochs or less) and needs to work with hyperparameters set globally for all GLUE tasks. Downstream finetuning is excluded from the total compute budget.

In our implementation, we analyze a setup with a classical `rtx2080ti` GPU (released September 2018) and separate setups with a more modern `rtxa4000` or a `rtxa6000` GPU, 48GB version (released October 2020). We pair each unit with 4 CPU cores and 32GB of RAM.

Why these limitations? We are principally interested in re-investigating the original BERT setup of Devlin et al. (2019) with limited compute. The optimal architecture of the transformer is not fixed, as the optimal size and shape depends on scaling laws (Kaplan et al., 2020). The limitations on usage of existing models rule out distillation from an existing model (Turc et al., 2019; Jiao et al., 2020; Sun et al., 2020; Wang et al., 2020b; Kaliamoorthi et al., 2021) and data filtering based on existing large models (Golchin et al., 2022), both of which ultimately answer questions about compression and transfer of already processed information. Further, we do not want to limit data to the original dataset used to train BERT, wanting to allow for possible improvements through better data curation and quality. The `rtx2080ti` GPU is a natural candidate for this experiment, given that it was released before Devlin et al. (2019), but the more recent `rtxa4000` is also interesting, as a more recent consumer-grade workstation variant. Finally we also test the `rtxa6000`, being arguably the (current) upper limit of a single-user workstation. At the finetuning stage we want to mimic the original BERT finetuning and evaluation setup, but provide additional limits to prevent gains based on tuning of only the downstream procedure, for example via computationally extensive downstream training (Bahri et al., 2021a), use of multiple downstream datasets (for example continued pretraining with MNLI before finetuning other tasks (Izsak et al., 2021)), and extended hyperparameter optimization for each GLUE task (Devlin et al., 2019; Liu et al., 2019; Lan et al., 2019).

3. Investigations

For our experimental evaluation we implement and test a considerable number of proposed modifications to the setup of Devlin et al. (2019) for their merits in our limited compute setting as described in Section 2. We first clarify the common implementation and initial data setup, and then investigate architectural, training and dataset improvements.

3.1. Modifying the Architecture

The most obvious way to efficiently scale down training is by modifying the model architecture; intuitively, it seems likely that lower-capacity models will be optimal in the cramming regime, or that some modification or would provide significant gains when tailored to this setting. In this section, we study the relationship between model type and final efficiency. We find that scaling laws create a strong barrier to scaling down. Per-token efficiency of training depends strongly on model size, but not on transformer shape. Furthermore, smaller models learn less efficiently, and this largely mitigates any throughput gains. Fortunately, the fact that training efficiency is nearly constant across models of the same size means that we can boost performance by finding architecture modifications that speed up gradient computation while keeping the parameter count nearly constant. This makes architecture selection fairly straightforward as we can make design choices based primarily on how they affect computation time for a single gradient step.

As such, we keep the overall layout of the model broadly similar (aside from an increase in depth to 16 layers and a Pre-normalization setup), and focus on removing components such as all linear layer biases, QKV biases and the head block. We add gated linear units, additional normalization layers, and scaled sinusoidal embeddings. We relegate a detailed discussion of these to the appendix.

3.2. Modifying the Training Setup

We likewise study the impact of training hyper-parameters on the BERT-base architecture. The original BERT training recipe understandably results is poor model performance in the cramming setting, and so we revisit a number of choices. In brief, we find that the combination of a large batch size, batch size ramp-up, large learning rate and aggressive learning rate schedules with a model trained with dropout deactivated, leads to the most throughput during training, and a model that is effective at downstream tasks. More details can be found in the appendix.

3.3. Optimizing the Dataset

We found above that scaling laws create a barrier to making major gains (beyond computational efficiencies) with architectural modifications. However, scaling laws do not preclude us from training on better data. Once we have exhausted our ability to train on more tokens per second, we should seek to train on better tokens.

We consider two data based pathways to better downscaling. First, we can filter, process, or sort the existing data in various ways. Second, we can swap our data source. To this end, we experiment with replacements for the `bookcorpus-wikipedia` dataset. We test several subsets of *The Pile* (Gao et al., 2020). We draw random subset from all sources, denoted `pile` and one containing raw text from only *Gutenberg*, *Books3* and *Wikipedia (en)*, denoted `pile-N`. From these Pile datasets we tokenize the first 4×10^6 entries to generate enough tokens for our single pass. Another popular source of data is C4, the colossal, cleaned version of Common Crawl (Raffel et al., 2020), from which we stream the first 20×10^6 entries. Finally, we also include the 2019 release of the OSCAR dataset (Suárez et al., 2019), denoted by `oscar`. For each data source we regenerate its own WordPiece tokenizer as described in Appendix C.1.

Table 2. Interactions between sorting, deduplication and filtering strategies from Appendix C.1 for several data sources measured in GLUE performance. The first row corresponds to the original BERT data. `bw` denotes `bookcorpus-wikipedia`, `c4` is C4 (colossal-cleaned-common-crawl) (Raffel et al., 2020), `oscar` is the 2019 release of the OSCAR dataset (Suárez et al., 2019). `pile` is the subset of The Pile (Gao et al., 2020). `pile-N` is the subset drawn only from the natural sources in the Pile. For additional details see Table 7.

Source	Filtered	Sorted	Dedup.	GLUE
`bw`	✗	✗	✗	78.1
`bw`	✓	✗	✗	78.7
`bw`	✓	✓	✗	78.8
`c4`	✗	✗	✗	75.9
`c4`	✓	✗	✗	79.3
`c4`	✓	✓	✗	79.0
`oscar`	✗	✗	✗	79.1
`oscar`	✓	✗	✗	79.2
`oscar`	✓	✓	✗	79.2
`oscar`	✓	✓	✓	**80.1**
`pile`	✗	✗	✗	78.2
`pile`	✓	✗	✗	79.3
`pile`	✓	✓	✗	**80.1**
`pile`	✓	✓	✓	80.0
`pile-N`	✗	✗	✗	79.2
`pile-N`	✓	✗	✗	79.8
`pile-N`	✓	✓	✗	**80.1**

Of these four sources, we find the natural split of The Pile to perform best in terms of downstream GLUE performance out-of-the-box. However, we can further improve performance through additional processing. We evaluate deduplication as described in Lee et al. (2022) via exact substring deduplication (of substrings of length 75), but find this not to reliably help in downstream performance in our case, see Table 2. We then test filtering for incompressible data. We use the tokenizer itself to remove all training sequences from each data source that cannot be compressed well; we simply set a threshold t, here $t = 0.25$, and drop all entries from the dataset where the number of tokens in the entry is larger than t times the number of raw characters. This removes, for example, sequences consisting of hard-to-compress HTML.

Finally we experiment with sorting, where we re-order all

Table 3. Comparison in GLUE-dev performance of baseline BERT to the crammed model. Note that all runs abide by the finetuning protocol described in Section 2 with fixed hyperparameters for all tasks and an epoch limit of 5. Missing values are NaN. The protocol of (Izsak et al., 2021) is designed for an 8 GPU server blade, and is crammed onto a single GPU here. T denotes the number of tokens ingested during training in billions. The MNLI column shows evaluation results for both matched and mismatched sets. The GLUE column depicts the full average over the same tasks as in Devlin et al. (2019).

	T(10^9)	MNLI	SST-2	STSB	RTE	QNLI	QQP	MRPC	CoLA	GLUE
BERT-base (Fully trained)	-	83.3/83.5	92.0	86.7	**58.8**	**90.6**	**87.8**	**89.3**	**56.9**	**81.0**
BERT-base (No Pretraining)	0	34.1/34.1	79.9	17.8	47.3	50.0	68.6	77.9	0.0	45.5
Trained for 1 day on a **2080ti**:										
BERT (orig. run, stopped early)	3.4	64.7/64.6	78.8	18.8	50.4	57.2	75.1	74.7	8.7	54.8
BERT (Izsak et al., 2021)	1.2	74.9/75.7	-	-	52.3	84.6	84.4	82.2	33.5	69.7
crammed BERT	4.3	82.5/82.8	91.6	86.2	56.7	89.1	87.1	88.3	48.3	79.2
Trained for 1 day on an **A4000**:										
BERT (orig. run, stopped early)	3.8	66.1/66.1	78.2	18.1	50.5	58.1	75.1	73.5	8.7	54.9
BERT (Izsak et al., 2021)	1.4	58.9/59.9	-	-	-	-	-	81.4	32.9	58.3
crammed BERT	4.5	82.7/83.1	91.5	86.4	56.5	88.8	86.9	88.2	48.6	79.2
Trained for 1 day on an **A6000**:										
BERT (orig. run, stopped early)	7.3	64.4/63.9	79.3	20.8	49.8	58.3	75.1	74.2	7.7	54.8
BERT (Izsak et al., 2021)	2.5	76.7/76.9	87.4	78.7	50.9	85.9	84.5	81.8	39.8	73.6
crammed BERT	8.7	**84.0/84.4**	**92.3**	**87.0**	57.4	90.0	87.7	89.0	51.8	80.4

tokenized sequences by some metric. We find the optimal metric to be sentence length and we sort so that sequences containing short sentences come first. This is empirically beneficial over e.g. sorting all sequences by their average (unigram) token prevalence.

Overall, wins from post-processing in this manner are noticeable, leading in aggregate to an improvements of 2% over the original dataset and we choose The Pile with both filtering and sorting as our new dataset going forward.

4. Finetuning Performance on GLUE

Finally, we systematically evaluate performance on the GLUE benchmark of Wang et al. (2018), minus WNLI as in Devlin et al. (2019). We note that we only use MNLI (m) during the previous sections and do not tune hyperparameters based on the full GLUE scores. We finetune both the pretrained BERT-base checkpoint and our models under the same constraints laid out in Section 2, namely that downstream hyperparameters have to be fixed over all downstream tasks and train for 5 epochs or less. For BERT-base, we finetune all datasets for 5 epochs with a batch size of 32 and learning rate of 2×10^{-5}. For the crammed models, we find that this is not optimal and minor improvements can be gained from a batch size of 16 and learning rate of 4×10^{-5} with cosine decay (this change does not improve the pretrained BERT checkpoint).

Table 3 and Table 4 describe the performance of this setup on the GLUE downstream tasks (as median over 5 downstream trials). There we compare the original BERT-base checkpoint, a reproduction of the BERT pretraining settings stopped after our budget limit is reached, the setup described in (Izsak et al., 2021), and the modified recipe, trained for a single day for each GPU setup. The performance of the crammed model recipe is surprisingly decent, especially for the larger datasets of MNLI, QQP, QNLI and SST-2, where downstream finetuning can smooth out the remaining differences between the full BERT model and the crammed variants. Further, we find substantial gains over both a naive BERT training with limited budget, and over the recipe described in Izsak et al. (2021). For Izsak et al. (2021), the described recipe was originally designed for a full 8 GPU server blade, and squeezing the BERT-large model therein onto the smaller GPUs in this experiment is responsible for most of the performance degradation and instability of this recipe in our scenario.

5. Conclusions

We discuss what performance a transformer-based language model can achieve when crammed into a setting with very limited compute, finding that several strands of modification lead to decent downstream performance on GLUE. Overall though, cramming language models is hard, as we empirically find many implications of Kaplan et al. (2020) to still hold in this regime, e.g. improvements through larger models are nearly evened out by their slower speed, and different architecture types and transformer shapes have limited impact.

We hope that this work can provide a baseline for explorations of *the question of cramming* we formalize in Section 2 and cast an additional light on a number of improvements and tricks proposed for transformer architectures in recent years. We do not believe that results in this work represent the limit that is achievable within the cramming compute budget, and hope for further research and developments in this direction.

References

Anil, R., Gupta, V., Koren, T., Regan, K., and Singer, Y. Scalable Second Order Optimization for Deep Learning. *arXiv:2002.09018 [cs, math, stat]*, March 2021. URL http://arxiv.org/abs/2002.09018.

Araabi, A. and Monz, C. Optimizing Transformer for Low-Resource Neural Machine Translation. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 3429–3435, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.304. URL https://aclanthology.org/2020.coling-main.304.

Artetxe, M., Du, J., Goyal, N., Zettlemoyer, L., and Stoyanov, V. On the Role of Bidirectionality in Language Model Pre-Training. *arxiv:2205.11726[cs]*, May 2022. doi: 10.48550/arXiv.2205.11726. URL http://arxiv.org/abs/2205.11726.

Baevski, A. and Auli, M. Adaptive Input Representations for Neural Language Modeling. In *International Conference on Learning Representations*, September 2018. URL https://openreview.net/forum?id=ByxZX20qFQ.

Bahri, D., Mobahi, H., and Tay, Y. Sharpness-Aware Minimization Improves Language Model Generalization. *arXiv:2110.08529 [cs]*, October 2021a. URL http://arxiv.org/abs/2110.08529.

Bahri, Y., Dyer, E., Kaplan, J., Lee, J., and Sharma, U. Explaining Neural Scaling Laws. *arxiv:2102.06701[cond-mat, stat]*, February 2021b. doi: 10.48550/arXiv.2102.06701. URL http://arxiv.org/abs/2102.06701.

Bajaj, P., Xiong, C., Ke, G., Liu, X., He, D., Tiwary, S., Liu, T.-Y., Bennett, P., Song, X., and Gao, J. METRO: Efficient Denoising Pretraining of Large Scale Autoencoding Language Models with Model Generated Signals. *arXiv:2204.06644 [cs]*, April 2022. URL http://arxiv.org/abs/2204.06644.

Bandy, J. and Vincent, N. Addressing "Documentation Debt" in Machine Learning: A Retrospective Datasheet for BookCorpus. *NeurIPS 2021 Track Datasets and Benchmarks*, November 2021. URL https://openreview.net/forum?id=Qd_eU1wvJeu.

Bansal, Y., Ghorbani, B., Garg, A., Zhang, B., Krikun, M., Cherry, C., Neyshabur, B., and Firat, O. Data Scaling Laws in NMT: The Effect of Noise and Architecture. *arXiv:2202.01994 [cs]*, February 2022. URL https://arxiv.org/abs/2202.01994v1.

Bao, H., Dong, L., Wei, F., Wang, W., Yang, N., Liu, X., Wang, Y., Piao, S., Gao, J., Zhou, M., and Hon, H.-W. UniLMv2: Pseudo-Masked Language Models for Unified Language Model Pre-Training. *arXiv:2002.12804 [cs]*, February 2020. URL http://arxiv.org/abs/2002.12804.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The Long-Document Transformer. *arXiv:2004.05150 [cs]*, December 2020. URL http://arxiv.org/abs/2004.05150.

Bender, E. M. The #BenderRule: On Naming the Languages We Study and Why It Matters, September 2019. URL https://thegradient.pub/the-benderrule-on-naming-the-languages-we-study-and-why-it-matters/.

Black, S., Biderman, S., Hallahan, E., Anthony, Q., Gao, L., Golding, L., He, H., Leahy, C., McDonell, K., Phang, J., Pieler, M., Prashanth, U. S., Purohit, S., Reynolds, L., Tow, J., Wang, B., and Weinbach, S. GPT-NeoX-20B: An Open-Source Autoregressive Language Model. *arXiv:2204.06745 [cs]*, April 2022. URL http://arxiv.org/abs/2204.06745.

Bollapragada, R., Byrd, R., and Nocedal, J. Adaptive Sampling Strategies for Stochastic Optimization. *SIAM Journal on Optimization*, 28(4):3312–3343, January 2018a. ISSN 1052-6234. doi: 10.1137/17M1154679. URL https://epubs.siam.org/doi/abs/10.1137/17M1154679.

Bollapragada, R., Mudigere, D., Nocedal, J., Shi, H.-J. M., and Tang, P. T. P. A Progressive Batching L-BFGS Method for Machine Learning. May 2018b. URL http://arxiv.org/abs/1802.05374.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language Models are Few-Shot Learners. In *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, December 2020. URL https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Carlini, N., Chien, S., Nasr, M., Song, S., Terzis, A., and Tramèr, F. Membership Inference Attacks From First Principles. In *2022 IEEE Symposium on Security and Privacy (SP)*, pp. 1897–1914, May 2022. doi: 10.1109/SP46214.2022.9833649.

Chelombiev, I., Justus, D., Orr, D., Dietrich, A., Gressmann, F., Koliousis, A., and Luschi, C. GroupBERT: Enhanced Transformer Architecture with Efficient Grouped Structures. *arxiv:2106.05822 [cs]*, June 2021. URL https://arxiv.org/abs/2106.05822v1.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A. M., Pillai, T. S., Pellat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Diaz, M., Firat, O., Catasta, M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S., and Fiedel, N. PaLM: Scaling Language Modeling with Pathways. *arXiv:2204.02311 [cs]*, April 2022. URL http://arxiv.org/abs/2204.02311.

Chung, H. W., Fevry, T., Tsai, H., Johnson, M., and Ruder, S. Rethinking Embedding Coupling in Pre-trained Language Models. In *International Conference on Learning Representations*, September 2020. URL https://openreview.net/forum?id=xpFFI_NtgpW.

Clark, A., de las Casas, D., Guy, A., Mensch, A., Paganini, M., Hoffmann, J., Damoc, B., Hechtman, B., Cai, T., Borgeaud, S., van den Driessche, G., Rutherford, E., Hennigan, T., Johnson, M., Millican, K., Cassirer, A., Jones, C., Buchatskaya, E., Budden, D., Sifre, L., Osindero, S., Vinyals, O., Rae, J., Elsen, E., Kavukcuoglu, K., and Simonyan, K. Unified Scaling Laws for Routed Language Models. *arXiv:2202.01169 [cs]*, February 2022. URL http://arxiv.org/abs/2202.01169.

Clark, K., Luong, M.-T., Le, Q. V., and Manning, C. D. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In *International Conference on Learning Representations*, September 2019. URL https://openreview.net/forum?id=r1xMH1BtvB.

Clark, K., Luong, M.-T., Le, Q. V., and Manning, C. D. Pre-Training Transformers as Energy-Based Cloze Models. *arXiv:2012.08561 [cs]*, December 2020. URL http://arxiv.org/abs/2012.08561.

Dai, Z., Lai, G., Yang, Y., and Le, Q. Funnel-Transformer: Filtering out Sequential Redundancy for Efficient Language Processing. In *Advances in Neural Information Processing Systems*, volume 33, pp. 4271–4282. Curran Associates, Inc., 2020. URL https://papers.nips.cc/paper/2020/hash/2cd2915e69546904e4e5d4a2ac9e1652-Abstract.html.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. *arxiv:2205.14135[cs]*, May 2022. doi: 10.48550/arXiv.2205.14135. URL http://arxiv.org/abs/2205.14135.

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. Language Modeling with Gated Convolutional Networks. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 933–941. PMLR, July 2017. URL https://proceedings.mlr.press/v70/dauphin17a.html.

Dayma, B., Patil, S., Cuenca, P., Saifullah, K., Abraham, T., Lê Khác, P., Melas, L., and Ghosh, R. DALL·E Mini, July 2021. URL https://github.com/borisdayma/dalle-mini.

De, S., Yadav, A., Jacobs, D., and Goldstein, T. Big Batch SGD: Automated Inference using Adaptive Batch Sizes. *arxiv:1610.05792[cs, math, stat]*, April 2017. URL http://arxiv.org/abs/1610.05792.

Dehghani, M., Tay, Y., Arnab, A., Beyer, L., and Vaswani, A. The Efficiency Misnomer. In *International Conference on Learning Representations*, September 2021. URL https://openreview.net/forum?id=iulEMLYh1uR.

Dettmers, T. TPUs vs GPUs for Transformers (BERT), October 2018. URL https://timdettmers.com/2018/10/17/tpus-vs-gpus-for-transformers-bert/.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*, May 2019. URL http://arxiv.org/abs/1810.04805.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv:2010.11929 [cs]*, June 2021. URL http://arxiv.org/abs/2010.11929.

Fusco, F., Pascual, D., and Staar, P. pNLP-Mixer: An Efficient all-MLP Architecture for Language. *arxiv:2202.04350 [cs]*, February 2022. URL https://arxiv.org/abs/2202.04350v1.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., Presser, S., and Leahy, C. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. *arXiv:2101.00027 [cs]*, December 2020. URL http://arxiv.org/abs/2101.00027.

Geiping, J., Fowl, L. H., Huang, W. R., Czaja, W., Taylor, G., Moeller, M., and Goldstein, T. Witches' Brew: Industrial Scale Data Poisoning via Gradient Matching. In *International Conference on Learning Representations*, April 2021. URL https://openreview.net/forum?id=01olnfLIbD.

Golchin, S., Surdeanu, M., Tavabi, N., and Kiapour, A. A Compact Pretraining Approach for Neural Language Models. *arxiv:2208.12367[cs]*, August 2022. doi: 10.48550/arXiv.2208.12367. URL http://arxiv.org/abs/2208.12367.

Gold, Z. and Latonero, M. Robots Welcome: Ethical and Legal Considerations for Web Crawling and Scraping. *Washington Journal of Law, Technology & Arts*, 13(3): 275–312, 2017. URL https://heinonline.org/HOL/P?h=hein.journals/washjolta13&i=283.

Gu, X., Liu, L., Yu, H., Li, J., Chen, C., and Han, J. On the Transformer Growth for Progressive BERT Training. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5174–5180, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.406. URL https://aclanthology.org/2021.naacl-main.406.

He, P., Gao, J., and Chen, W. DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing. *arXiv:2111.09543 [cs]*, December 2021. URL http://arxiv.org/abs/2111.09543.

Hernandez, D., Brown, T., Conerly, T., DasSarma, N., Drain, D., El-Showk, S., Elhage, N., Hatfield-Dodds, Z., Henighan, T., Hume, T., Johnston, S., Mann, B., Olah, C., Olsson, C., Amodei, D., Joseph, N., Kaplan, J., and McCandlish, S. Scaling Laws and Interpretability of Learning from Repeated Data. *arxiv:2205.10487[cs]*, May 2022. URL http://arxiv.org/abs/2205.10487.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., and Sifre, L. Training Compute-Optimal Large Language Models. *arXiv:2203.15556 [cs]*, March 2022. URL http://arxiv.org/abs/2203.15556.

Hooker, S. The hardware lottery. *Communications of the ACM*, 64(12):58–65, November 2021. ISSN 0001-0782. doi: 10.1145/3467017. URL https://doi.org/10.1145/3467017.

Hou, L., Pang, R. Y., Zhou, T., Wu, Y., Song, X., Song, X., and Zhou, D. Token Dropping for Efficient BERT Pretraining. *arXiv:2203.13240 [cs]*, March 2022. URL http://arxiv.org/abs/2203.13240.

Hua, W., Dai, Z., Liu, H., and Le, Q. Transformer Quality in Linear Time. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 9099–9117. PMLR, June 2022. URL https://proceedings.mlr.press/v162/hua22a.html.

Hui, L. and Belkin, M. Evaluation of Neural Architectures Trained with Square Loss vs Cross-Entropy in Classification Tasks. *arXiv:2006.07322 [cs, stat]*, October 2021. URL http://arxiv.org/abs/2006.07322.

Iandola, F. N., Shaw, A. E., Krishna, R., and Keutzer, K. W. SqueezeBERT: What can computer vision teach NLP about efficient neural networks? *arXiv:2006.11316 [cs]*, June 2020. URL http://arxiv.org/abs/2006.11316.

Ilyas, A., Park, S. M., Engstrom, L., Leclerc, G., and Madry, A. Datamodels: Predicting Predictions from Training Data. *arxiv:2202.00622[cs, stat]*, February 2022. doi: 10.48550/arXiv.2202.00622. URL http://arxiv.org/abs/2202.00622.

Izsak, P., Berchansky, M., and Levy, O. How to Train BERT with an Academic Budget. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 10644–10652, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.831. URL https://aclanthology.org/2021.emnlp-main.831.

Javaheripi, M., Shah, S., Mukherjee, S., Religa, T. L., Mendes, C. C. T., de Rosa, G. H., Bubeck, S., Koushanfar, F., and Dey, D. LiteTransformerSearch: Training-free On-device Search for Efficient Autoregressive Language Models. *arXiv:2203.02094 [cs]*, March 2022. URL http://arxiv.org/abs/2203.02094.

Jiang, Y., Neyshabur, B., Mobahi, H., Krishnan, D., and Bengio, S. Fantastic Generalization Measures and Where to Find Them. *arXiv:1912.02178 [cs, stat]*, December 2019. URL http://arxiv.org/abs/1912.02178.

Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., and Liu, Q. TinyBERT: Distilling BERT for Natural Language Understanding. *arXiv:1909.10351 [cs]*, October 2020. URL http://arxiv.org/abs/1909.10351.

Joshi, M., Chen, D., Liu, Y., Weld, D. S., Zettlemoyer, L., and Levy, O. SpanBERT: Improving Pre-training by Representing and Predicting Spans. *Transactions of the Association for Computational Linguistics*, 8:64–77, 2020. doi: 10.1162/tacl_a_00300. URL https://aclanthology.org/2020.tacl-1.5.

Kaliamoorthi, P., Siddhant, A., Li, E., and Johnson, M. Distilling Large Language Models into Tiny and Effective Students using pQRNN. *arxiv: 2101.08890 [cs]*, January 2021. URL https://arxiv.org/abs/2101.08890v1.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling Laws for Neural Language Models. *arxiv:2001.08361[cs, stat]*, January 2020. doi: 10.48550/arXiv.2001.08361. URL http://arxiv.org/abs/2001.08361.

Ke, G., He, D., and Liu, T.-Y. Rethinking Positional Encoding in Language Pre-training. In *International Conference on Learning Representations*, September 2020. URL https://openreview.net/forum?id=09-528y2Fgf.

Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*, San Diego, May 2015. URL http://arxiv.org/abs/1412.6980.

Komatsuzaki, A. One Epoch Is All You Need. *arXiv:1906.06669 [cs, stat]*, June 2019. URL http://arxiv.org/abs/1906.06669.

Kudo, T. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 66–75, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1007. URL https://aclanthology.org/P18-1007.

Kudo, T. and Richardson, J. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In *EMNLP (Demonstration)*, July 2019. URL https://openreview.net/forum?id=S1EyQGf_bH.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *International Conference on Learning Representations*, September 2019. URL https://openreview.net/forum?id=H1eA7AEtvS.

Laurençon, H., Saulnier, L., Wang, T., Akiki, C., del Moral, A. V., Scao, T. L., Von Werra, L., Mou, C., Ponferrada, E. G., Nguyen, H., Frohberg, J., Šaško, M., Lhoest, Q., McMillan-Major, A., Dupont, G., Biderman, S., Rogers, A., allal, L. B., De Toni, F., Pistilli, G., Nguyen, O., Nikpoor, S., Masoud, M., Colombo, P., de la Rosa, J., Villegas, P., Thrush, T., Longpre, S., Nagel, S., Weber, L., Muñoz, M., Zhu, J., Van Strien, D., Alyafeai, Z., Almubarak, K., Vu, M. C., Gonzalez-Dios, I., Soroa, A., Lo, K., Dey, M., Suarez, P. O., Gokaslan, A., Bose, S., Adelani, D., Phan, L., Tran, H., Yu, I., Pai, S., Chim, J., Lepercq, V., Ilic, S., Mitchell, M., Luccioni, S. A., and Jernite, Y. The BigScience ROOTS Corpus: A 1.6TB Composite Multilingual Dataset. *arxiv:2303.03915[cs]*, March 2023. doi: 10.48550/arXiv.2303.03915. URL http://arxiv.org/abs/2303.03915.

Lee, K., Ippolito, D., Nystrom, A., Zhang, C., Eck, D., Callison-Burch, C., and Carlini, N. Deduplicating Training Data Makes Language Models Better. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8424–8445, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.577. URL https://aclanthology.org/2022.acl-long.577.

Lee-Thorp, J., Ainslie, J., Eckstein, I., and Ontanon, S. FNet: Mixing Tokens with Fourier Transforms. *arxiv:2105.03824 [cs]*, May 2021. URL https://arxiv.org/abs/2105.03824v3.

Lei, T., Tian, R., Bastings, J., and Parikh, A. P. Simple Recurrence Improves Masked Language Models. *arxiv:2205.11588[cs]*, May 2022. URL http://arxiv.org/abs/2205.11588.

Li, C., Zhang, M., and He, Y. Curriculum Learning: A Regularization Method for Efficient and Stable Billion-Scale GPT Model Pre-Training. *arXiv:2108.06084 [cs]*, February 2022. URL http://arxiv.org/abs/2108.06084.

Liu, F., Shakeri, S., Yu, H., and Li, J. EncT5: Fine-tuning T5 Encoder for Non-autoregressive Tasks. *arxiv:2110.08426[cs]*, October 2021a. doi: 10.48550/arXiv.2110.08426. URL http://arxiv.org/abs/2110.08426.

Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Han, J. On the Variance of the Adaptive Learning Rate and Beyond. In *International Conference on Learning Representations*, March 2020a. URL https://openreview.net/forum?id=rkgz2aEKDr.

Liu, L., Liu, X., Gao, J., Chen, W., and Han, J. Understanding the Difficulty of Training Transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 5747–5763, Online, November 2020b. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.463. URL https://aclanthology.org/2020.emnlp-main.463.

Liu, L., Liu, J., and Han, J. Multi-head or Single-head? An Empirical Comparison for Transformer Training. *arxiv:2106.09650[cs]*, June 2021b. doi: 10.48550/arXiv.2106.09650. URL http://arxiv.org/abs/2106.09650.

Liu, X., Su, J., and Huang, F. Tuformer: Data-driven Design of Transformers for Improved Generalization or Efficiency. In *International Conference on Learning Representations*, September 2021c. URL https://openreview.net/forum?id=V0A5g83gdQ_.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv:1907.11692 [cs]*, July 2019. URL http://arxiv.org/abs/1907.11692.

Liu, Z., Wang, Y., Kasai, J., Hajishirzi, H., and Smith, N. A. Probing Across Time: What Does RoBERTa Know and When? In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 820–842, Punta Cana, Dominican Republic, November 2021d. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.71. URL https://aclanthology.org/2021.findings-emnlp.71.

Loshchilov, I. and Hutter, F. Decoupled Weight Decay Regularization. *arXiv:1711.05101 [cs, math]*, November 2017. URL http://arxiv.org/abs/1711.05101.

Merity, S. Single Headed Attention RNN: Stop Thinking With Your Head. *arXiv:1911.11423 [cs]*, November 2019. URL http://arxiv.org/abs/1911.11423.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., and Wu, H. Mixed Precision Training. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=r1gs9JgRZ.

Mindermann, S., Brauner, J., Razzak, M., Sharma, M., Kirsch, A., Xu, W., Höltgen, B., Gomez, A. N., Morisot, A., Farquhar, S., and Gal, Y. Prioritized Training on Points that are Learnable, Worth Learning, and Not Yet Learnt. *arxiv:2206.07137[cs]*, June 2022. doi: 10.48550/arXiv.2206.07137. URL http://arxiv.org/abs/2206.07137.

Mukherjee, S., Awadallah, A. H., and Gao, J. XtremeDistilTransformers: Task Transfer for Task-agnostic Distillation. *arXiv:2106.04563 [cs]*, June 2021. URL http://arxiv.org/abs/2106.04563.

Müller, R., Kornblith, S., and Hinton, G. E. When does label smoothing help? In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://papers.nips.cc/paper/2019/hash/f1748d6b0fd9d439f71450117eba2725-Abstract.html.

Nagarajan, V. and Kolter, J. Z. Generalization in Deep Networks: The Role of Distance from Initialization. *arXiv:1901.01672 [cs, stat]*, January 2019. URL http://arxiv.org/abs/1901.01672.

Narang, S., Chung, H. W., Tay, Y., Fedus, L., Fevry, T., Matena, M., Malkan, K., Fiedel, N., Shazeer, N., Lan, Z., Zhou, Y., Li, W., Ding, N., Marcus, J., Roberts, A., and Raffel, C. Do Transformer Modifications Transfer Across Implementations and Applications? In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 5758–5773, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.465. URL https://aclanthology.org/2021.emnlp-main.465.

Narasimhan, S. NVIDIA Clocks World's Fastest BERT Training Time and Largest Transformer Based Model, Paving Path For Advanced Conversational AI, August 2019. URL https://developer.nvidia.com/blog/training-bert-with-gpus/.

Nawrot, P., Tworkowski, S., Tyrolski, M., Kaiser, Ł., Wu, Y., Szegedy, C., and Michalewski, H. Hierarchical Transformers Are More Efficient Language Models. *arxiv:2110.13711[cs]*, April 2022. URL http://arxiv.org/abs/2110.13711.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in PyTorch. In *NIPS 2017 Autodiff Workshop*, Long Beach, CA, 2017. URL https://openreview.net/forum?id=BJJsrmfCZ.

Peng, B. RWKV-LM. Zenodo, August 2021. URL https://zenodo.org/record/5196577.

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Cao, H., Cheng, X., Chung, M., Grella, M., GV, K. K., He, X., Hou, H., Kazienko, P., Kocon, J., Kong, J., Koptyra, B., Lau, H., Mantri, K. S. I., Mom, F., Saito, A., Tang, X., Wang, B., Wind, J. S., Wozniak, S., Zhang, R., Zhang, Z., Zhao, Q., Zhou, P., Zhu, J., and Zhu, R.-J. RWKV: Reinventing RNNs for the Transformer Era. *arxiv:2305.13048[cs]*, May 2023. doi: 10.48550/arXiv.2305.13048. URL http://arxiv.org/abs/2305.13048.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language Models are Unsupervised Multitask Learners. *OpenAI*, pp. 24, 2019.

Rae, J. W., Borgeaud, S., Cai, T., Millican, K., Hoffmann, J., Song, F., Aslanides, J., Henderson, S., Ring, R., Young, S., Rutherford, E., Hennigan, T., Menick, J., Cassirer, A., Powell, R., van den Driessche, G., Hendricks, L. A., Rauh, M., Huang, P.-S., Glaese, A., Welbl, J., Dathathri, S., Huang, S., Uesato, J., Mellor, J., Higgins, I., Creswell, A., McAleese, N., Wu, A., Elsen, E., Jayakumar, S., Buchatskaya, E., Budden, D., Sutherland, E., Simonyan, K., Paganini, M., Sifre, L., Martens, L., Li, X. L., Kuncoro, A., Nematzadeh, A., Gribovskaya, E., Donato, D., Lazaridou, A., Mensch, A., Lespiau, J.-B., Tsimpoukelli, M., Grigorev, N., Fritz, D., Sottiaux, T., Pajarskas, M., Pohlen, T., Gong, Z., Toyama, D., d'Autume, C. d. M., Li, Y., Terzi, T., Mikulik, V., Babuschkin, I., Clark, A., Casas, D. d. L., Guy, A., Jones, C., Bradbury, J., Johnson, M., Hechtman, B., Weidinger, L., Gabriel, I., Isaac, W., Lockhart, E., Osindero, S., Rimell, L., Dyer, C., Vinyals, O., Ayoub, K., Stanway, J., Bennett, L., Hassabis, D., Kavukcuoglu, K., and Irving, G. Scaling Language Models: Methods, Analysis & Insights from Training Gopher. *arXiv:2112.11446 [cs]*, January 2022. URL http://arxiv.org/abs/2112.11446.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *arXiv:1910.10683 [cs, stat]*, July 2020. URL http://arxiv.org/abs/1910.10683.

Rasley, J., Rajbhandari, S., Ruwase, O., and He, Y. DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pp. 3505–3506, New York, NY, USA, August 2020. Association for Computing Machinery. ISBN 978-1-4503-7998-4. doi: 10.1145/3394486.3406703. URL https://doi.org/10.1145/3394486.3406703.

Ren, J., Rajbhandari, S., Aminabadi, R. Y., Ruwase, O., Yang, S., Zhang, M., Li, D., and He, Y. {ZeRO-Offload}: Democratizing {Billion-Scale} Model Training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 551–564, 2021. ISBN 978-1-939133-23-6. URL https://www.usenix.org/conference/atc21/presentation/ren-jie.

Richter, O. and Wattenhofer, R. Normalized Attention Without Probability Cage. *arXiv:2005.09561 [cs, stat]*, May 2020. URL http://arxiv.org/abs/2005.09561.

Roy, A., Anil, R., Lai, G., Lee, B., Zhao, J., Zhang, S., Wang, S., Zhang, Y., Wu, S., Swavely, R., Tao, Yu, Dao, P., Fifty, C., Chen, Z., and Wu, Y. N-Grammer: Augmenting Transformers with latent n-grams. *arxiv:2207.06366[cs]*, July 2022. doi: 10.48550/arXiv.2207.06366. URL http://arxiv.org/abs/2207.06366.

Sarofeen, C., Bialecki, P., Jiang, J., Stephano, K., Kozuki, M., Vaidya, N., and Bekman, S. Introducing nvFuser, a deep learning compiler for PyTorch, August 2022. URL https://pytorch.org/blog/introducing-nvfuser-a-deep-learning-compiler-for-pytorch/.

Scao, T. L., Wang, T., Hesslow, D., Saulnier, L., Bekman, S., Bari, M. S., Biderman, S., Elsahar, H., Phang, J., Press, O., Raffel, C., Sanh, V., Shen, S., Sutawika, L., Tae, J., Yong, Z. X., Launay, J., and Beltagy, I. What Language Model to Train if You Have One Million GPU Hours? In *Challenges* {\&, April 2022. URL https://openreview.net/forum?id=rI7BL3fHIZq.

Schwarzschild, A. Easy-To-Hard, October 2021. URL https://github.com/aks2203/easy-to-hard.

Sellam, T., Yadlowsky, S., Tenney, I., Wei, J., Saphra, N., D'Amour, A., Linzen, T., Bastings, J., Turc, I. R., Eisenstein, J., Das, D., and Pavlick, E. The MultiBERTs: BERT Reproductions for Robustness Analysis. In *International Conference on Learning Representations*, March 2022. URL https://openreview.net/forum?id=K0E_F0gFDgA.

Sennrich, R., Haddow, B., and Birch, A. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://aclanthology.org/P16-1162.

Shazeer, N. and Stern, M. Adafactor: Adaptive Learning Rates with Sublinear Memory Cost. *arxiv:1804.04235[cs,*

stat], April 2018. doi: 10.48550/arXiv.1804.04235. URL http://arxiv.org/abs/1804.04235.

Shen, S., Walsh, P., Keutzer, K., Dodge, J., Peters, M., and Beltagy, I. Staged Training for Transformer Language Models. *arXiv:2203.06211 [cs]*, March 2022. URL http://arxiv.org/abs/2203.06211.

Shleifer, S., Weston, J., and Ott, M. NormFormer: Improved Transformer Pretraining with Extra Normalization. *arXiv:2110.09456 [cs]*, November 2021. URL http://arxiv.org/abs/2110.09456.

Smith, L. N. and Topin, N. Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates. *arXiv:1708.07120 [cs, stat]*, May 2018. URL http://arxiv.org/abs/1708.07120.

So, D., Mańke, W., Liu, H., Dai, Z., Shazeer, N., and Le, Q. V. Searching for Efficient Transformers for Language Modeling. In *Advances in Neural Information Processing Systems*, May 2021. URL https://openreview.net/forum?id=bzpkxS_JVsI.

Sridhar, S. N., Sarah, A., and Sundaresan, S. TrimBERT: Tailoring BERT for Trade-offs. *arXiv:2202.12411 [cs]*, February 2022. URL http://arxiv.org/abs/2202.12411.

Su, J., Lu, Y., Pan, S., Wen, B., and Liu, Y. RoFormer: Enhanced Transformer with Rotary Position Embedding. *arxiv:2104.09864 [cs]*, April 2021. URL https://arxiv.org/abs/2104.09864v2.

Suárez, P. J. O., Sagot, B., and Romary, L. Asynchronous Pipeline for Processing Huge Corpora on Medium to Low Resource Infrastructures. In *7th Workshop on the Challenges in the Management of Large Corpora (CMLC-7)*. Leibniz-Institut für Deutsche Sprache, July 2019. doi: 10.14618/IDS-PUB-9021. URL https://inria.hal.science/hal-02148693.

Sukhbaatar, S., Grave, E., Bojanowski, P., and Joulin, A. Adaptive Attention Span in Transformers. *arXiv:1905.07799 [cs, stat]*, August 2019. URL http://arxiv.org/abs/1905.07799.

Sun, S., Cheng, Y., Gan, Z., and Liu, J. Patient Knowledge Distillation for BERT Model Compression. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4323–4332, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1441. URL https://aclanthology.org/D19-1441.

Sun, Z., Yu, H., Song, X., Liu, R., Yang, Y., and Zhou, D. MobileBERT: A Compact Task-Agnostic BERT for Resource-Limited Devices. *arXiv:2004.02984 [cs]*, April 2020. URL http://arxiv.org/abs/2004.02984.

Sutton, R. The Bitter Lesson. *Incomplete Ideas (blog)*, pp. 1, March 2019. URL http://www.incompleteideas.net/IncIdeas/BitterLesson.html.

Tan, L. What the bookcorpus?, December 2019. URL https://gist.github.com/alvations/4d2278e5a5fbcf2e07f49315c4ec1110.

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., and Metzler, D. Long Range Arena: A Benchmark for Efficient Transformers. *arXiv:2011.04006 [cs]*, November 2020a. URL http://arxiv.org/abs/2011.04006.

Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. Efficient Transformers: A Survey. *arXiv:2009.06732 [cs]*, September 2020b. URL http://arxiv.org/abs/2009.06732.

Tay, Y., Dehghani, M., Rao, J., Fedus, W., Abnar, S., Chung, H. W., Narang, S., Yogatama, D., Vaswani, A., and Metzler, D. Scale Efficiently: Insights from Pretraining and Finetuning Transformers. In *International Conference on Learning Representations*, September 2021. URL https://openreview.net/forum?id=f2OYVDyfIB.

Tay, Y., Dehghani, M., Abnar, S., Chung, H. W., Fedus, W., Rao, J., Narang, S., Tran, V. Q., Yogatama, D., and Metzler, D. Scaling Laws vs Model Architectures: How does Inductive Bias Influence Scaling? *arxiv:2207.10551[cs]*, July 2022a. doi: 10.48550/arXiv.2207.10551. URL http://arxiv.org/abs/2207.10551.

Tay, Y., Dehghani, M., Tran, V. Q., Garcia, X., Bahri, D., Schuster, T., Zheng, H. S., Houlsby, N., and Metzler, D. Unifying Language Learning Paradigms. *arxiv:2205.05131[cs]*, May 2022b. URL http://arxiv.org/abs/2205.05131.

Treviso, M., Ji, T., Lee, J.-U., van Aken, B., Cao, Q., Ciosici, M. R., Hassid, M., Heafield, K., Hooker, S., Martins, P. H., Martins, A. F. T., Milder, P., Raffel, C., Simpson, E., Slonim, N., Balasubramanian, N., Derczynski, L., and Schwartz, R. Efficient Methods for Natural Language Processing: A Survey. *arxiv:2209.00099[cs]*, August 2022. URL http://arxiv.org/abs/2209.00099.

Turc, I., Chang, M.-W., Lee, K., and Toutanova, K. Well-Read Students Learn Better: On the Importance of

Pre-training Compact Models. *arXiv:1908.08962 [cs]*, September 2019. URL http://arxiv.org/abs/1908.08962.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention Is All You Need. *arXiv:1706.03762 [cs]*, December 2017. URL http://arxiv.org/abs/1706.03762.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *International Conference on Learning Representations*, September 2018. URL https://openreview.net/forum?id=rJ4km2R5t7.

Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/hash/4496bf24afe7fab6f046bf4923da8de6-Abstract.html.

Wang, S. and Kanwar, P. BFloat16: The secret to high performance on Cloud TPUs, August 2019. URL https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus/.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-Attention with Linear Complexity. *arxiv:2006.04768v3 [cs]*, June 2020a. URL https://arxiv.org/abs/2006.04768v3.

Wang, T., Roberts, A., Hesslow, D., Scao, T. L., Chung, H. W., Beltagy, I., Launay, J., and Raffel, C. What Language Model Architecture and Pretraining Objective Work Best for Zero-Shot Generalization? *arXiv:2204.05832 [cs, stat]*, April 2022. URL http://arxiv.org/abs/2204.05832.

Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., and Zhou, M. MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers. *arXiv:2002.10957 [cs]*, April 2020b. URL http://arxiv.org/abs/2002.10957.

Warstadt, A., Singh, A., and Bowman, S. R. Neural Network Acceptability Judgments. *arxiv:1805.12471[cs]*, October 2019. doi: 10.48550/arXiv.1805.12471. URL http://arxiv.org/abs/1805.12471.

Wettig, A., Gao, T., Zhong, Z., and Chen, D. Should You Mask 15% in Masked Language Modeling? *arxiv:2202.08005 [cs]*, February 2022. URL https://arxiv.org/abs/2202.08005v1.

Wies, N., Levine, Y., Jannai, D., and Shashua, A. Which transformer architecture fits my data? A vocabulary bottleneck in self-attention. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 11170–11181. PMLR, July 2021. URL https://proceedings.mlr.press/v139/wies21a.html.

Wilka, R., Landy, R., and McKinney, S. A. How Machines Learn: Where Do Companies Get Data for Machine Learning and What Licenses Do They Need. *Washington Journal of Law, Technology & Arts*, 13(3):217–244, 2017. URL https://heinonline.org/HOL/P?h=hein.journals/washjolta13&i=226.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, Ł., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arxiv:1609.08144[cs]*, October 2016. URL http://arxiv.org/abs/1609.08144.

Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T.-Y. On Layer Normalization in the Transformer Architecture. *arXiv:2002.04745 [cs, stat]*, June 2020. URL http://arxiv.org/abs/2002.04745.

Yadav, A. Making L-BFGS Work with Industrial-Strength Nets. In *BMVC 2020*, pp. 13, 2020.

You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K., and Hsieh, C.-J. Large Batch Optimization for Deep Learning: Training BERT in 76 minutes. In *International Conference on Learning Representations*, September 2019. URL https://openreview.net/forum?id=Syx4wnEtvH.

Zhang, B. and Sennrich, R. Root Mean Square Layer Normalization. *arXiv:1910.07467 [cs, stat]*, October 2019. URL http://arxiv.org/abs/1910.07467.

Zhu, C., Ni, R., Xu, Z., Kong, K., Huang, W. R., and Goldstein, T. GradInit: Learning to Initialize Neural Networks for Stable and Efficient Training. *arxiv:2102.08098[cs]*, November 2021. doi: 10.48550/arXiv.2102.08098. URL http://arxiv.org/abs/2102.08098.

Appendix Table of Contents

A. Related Work on Efficient Transformers

How long does it take to train BERT? In general, this question is hard to answer, due to wildly varying hardware and software setups and differing measures of efficiency (Dehghani et al., 2021). An upper bound on the compute of a training run can be established by finding the total number of (low-precision) floating point operations available over the wallclock budget of the run. This peak of total FLOPs in a given time interval is generally not reached in actual compute, even for highly optimized models (Chowdhery et al., 2022), but represents the paid budget required to realize a training run. We summarize budgets for a few select training runs in Table 1. After the original training run for BERT on TPUs, initial reactions estimated up to 11 days of compute for comparable results on GPUs (Dettmers, 2018). However, sustained improvements, especially in software, have reduced the upper limit significantly (You et al., 2019; Narasimhan, 2019). Yet, recipes and implementations generally require entire server nodes (for GPUs) or TPU slices and target larger BERT architectures.

Other work discussing improvements to BERT targets compute settings closer to the original BERT, for example SqueezeBERT (Iandola et al., 2020) employs 8 `Titan RTX` cards for four days. Sellam et al. (2022) note that the original BERT training run is an outlier and doubling its training time more reliably reproduces the original results.

Our central point of comparison for BERT training with limited resources is the work of Izsak et al. (2021) who also attempt the goal of training BERT within 24 hours with overall similar limitations, but use a full server node with 8 `V100` GPUs. Izsak et al. (2021) choose a $\text{BERT}_{\text{LARGE}}$ architecture variant and train with sequence length of 128, including a range of tweaks such as modified learning rates schedules, large batch sizes, sparse prediction and packed sequences. We re-evaluate this setup as a key baseline setting for our own compute budget (which is about 15x smaller).

Studies of Efficient Transformers. Recent years have seen a flurry of research working to improve and modify the transformer architecture proposed in Vaswani et al. (2017) and we refer to Treviso et al. (2022) for a recent categorization and review of research in this area. Several meta-studies have investigated proposed improvements and modifications: Narang et al. (2021) evaluate a large range of architectural modifications applied to the T5 model pipeline of Raffel et al. (2020) on tasks in both language understanding and translation. The encoder-decoder structure of T5 is closer in spirit to the original transformer setup, but is understood to behave similarly to BERT when using the encoder component (Liu et al., 2021a). Evaluating modifications with 1.75 days of compute on `TPU` slices they find that most improvements do not reliably materialize gains in final accuracy. Tay et al. (2021) work in the same setting and evaluate the optimal shape of T5 derived architectures and its relative effects on downstream performance as models are scaled. Further exploration of the scaling behavior of various architectural improvements in Tay et al. (2022a) find that only few modifications outperform the original architecture of Vaswani et al. (2017) at all scales, especially when evaluating downstream accuracy. The meta-study investigating improvements in preparation for extreme-scale training in Scao et al. (2022) focuses on minor modifications to layout, positional embeddings and data sources for autoregressive models, and other extremely-large scale training runs have so far been similarly conservative in their settings (Brown et al., 2020; Black et al., 2022; Rae et al., 2022).

In general, these evaluations target larger compute settings than we intend to use and therefore we are concerned with whether improvements (often from academic sources and proposed with evaluations on small scales) translate to larger scales. In this work, we set aside the question of (up)scaling and focus only on limited compute.

Scaling Laws. The difficulty in finding tangible improvements is echoed in the scaling laws of Kaplan et al. (2020). Over a wide range of transformer model shapes, Kaplan et al. (2020) find that only model size (as number of parameters in non-embedding layers) strongly predicts performance. Further, for a fixed compute budget, an optimal model size can be derived, but performance is only mildly connected to model size - larger models process less data per unit of compute, but

improve faster by almost the same margin. While the precise coefficients of these scaling laws continue to be iterated on (Hoffmann et al., 2022) and adapted (Bansal et al., 2022; Clark et al., 2022; Bahri et al., 2021b), their overall logic appears hard to escape, even if power laws fit observations somewhat less well on small scales.

B. Broader Impact

Overall, we hope that this type of inquiry can aid in the democratization of machine learning. Whereas the current paradigm for most users of models like BERT is to download the existing checkpoint, and tune it for their own purposes, they might now be able to train their own model. The reduced reliance on public checkpoints trained with semi-unknown data distributions is beneficial both for reasons of data provenance, as well as for reasons of model security.

On the other hand, we only provide evidence in this work that these crammed models are similarly useful on existing classical downstream tasks, such as GLUE and superGLUE. It is unclear whether other properties that have been extensively studied in the literature for the existing public BERT checkpoint, such as robustness, out-of-distribution generalization, fairness or inherent biases, carry over to the crammed model. More research is needed here to find out whether there are inopportune trade-offs arising from the modified training procedure.

C. Additional Evaluations

C.1. Implementation Details

We implement everything in PyTorch (Paszke et al., 2017) and to limit our gains from the "software lottery" (Hooker, 2021) we do not use specialized implementations, which would further bias results towards well-established components. We keep everything on the implementation level of the PyTorch framework, allowing only automated operator fusion (Sarofeen et al., 2022) via compilation that can be applied equally to all components. After choosing a final architecture variant, we then re-enable specialized optimizations, such as efficient kernels (Dao et al., 2022). We run all experiments and ablation studies with automated mixed precision (Micikevicius et al., 2018).

Initial Data Setup We start our investigation with a close analogue to the original raw text sources of Devlin et al. (2019), using a recent dump of the English Wikipedia (`20220301.en`) and English `bookcorpus`, noting the commentary of Tan (2019); Bandy & Vincent (2021). We force all text into lower-case, strip accents and non-ascii characters and create an English tokenizer from scratch based only on this data. We choose WordPiece with a vocabulary size of $2^{15} = 32768$ (Wu et al., 2016). We pack tokenized data into randomized sequences of length 128 and separate unrelated fragments by `<sep>`. The shorter sequence length is sufficient for the downstream applications that we are targeting and simplifies attention computations. Packing data into full sequences limits us to simpler sequence losses, but uses the available compute optimally Liu et al. (2019); Izsak et al. (2021). For the targeted compute settings, this sequence length results in micro-batch sizes of 64 to 96 for most variations of the base BERT architecture on the `gtx2080ti`, which we will accumulate into larger batch sizes. With our limited compute budget, this produces enough samples to run single-epoch training (Komatsuzaki, 2019; Hernandez et al., 2022) where no data point is revisited.

C.2. Modifying the Architecture

The most obvious way to efficiently scale down training is by modifying the model architecture; intuitively, it seems likely that lower-capacity models will be optimal in the cramming regime, or that some modification or would provide significant gains when tailored to this setting. In this section, we study the relationship between model type and final efficiency. We find that scaling laws create a strong barrier to scaling down. Per-token efficiency of training depends strongly on model size, but not on transformer shape. Furthermore, smaller models learn less efficiently, and this largely mitigates any throughput gains. Fortunately, the fact that training efficiency is nearly constant across models of the same size means that we can boost performance by finding architecture modifications that speed up gradient computation while keeping the parameter count nearly constant. This makes architecture selection fairly straightforward as we can make design choices based primarily on how they affect computation time for a single gradient step.

Scaling laws are a roadblock to quick wins. A large corpus of research in recent years has developed architectural improvements to speed up the original transformer. Yet, many of these methods have not been found to improve training, see for example meta-studies for T5 architectures in Narang et al. (2021); Tay et al. (2022a). But, in the low compute setting where data throughput is of utmost importance, maybe this is the way forward? Scaling laws hold strongly in the limit as

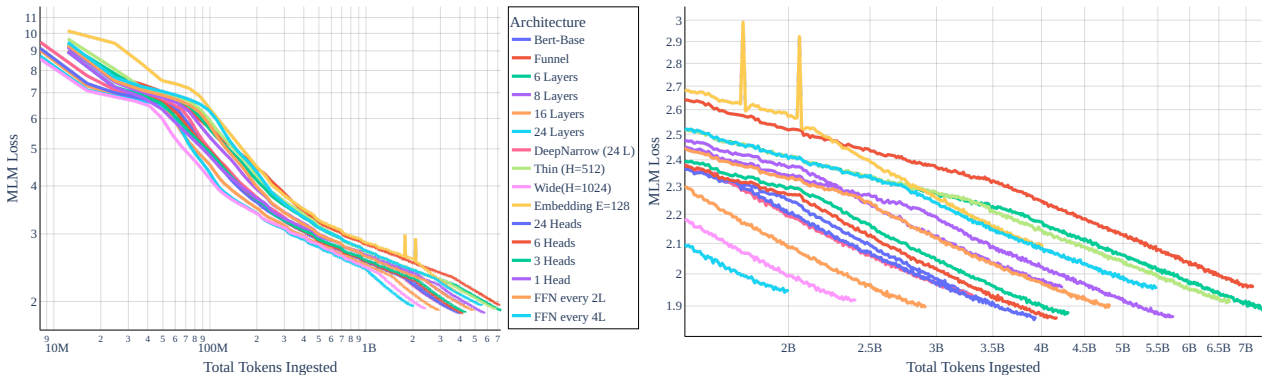

Figure 1. Various Transformer architectures and shapes, showing MLM loss versus number of tokens ingested. Left: Global view. Right: Zoom onto 10e8 or more tokens. All models trained with the same budget. We see that improvements through architectural reshaping are minimal; while there are some fluctuations in loss early in training, the rates of loss decay during most of training differ by a multiplicative constant (horizontal shift due to logarithmic horizontal axis) that depends strongly on the model size and not model shape.

resources grow. However, these laws also hold reliably in the low-resource regime. We extensively test recently published architecture variations, but find that none can escape the conclusion of Kaplan et al. (2020) that model shape does not significantly affect performance.

We exemplify the effect of scaling laws for many transformer variants from the literature in Figure 1, where we train each architecture variant with optimized training hyperparameters as described below in Section 3.2. We apply these architecture variants to a shared baseline model that incorporates Pre-Normalization and rotary embedding. Figure 1 visualizes the progress of masked-language modeling (MLM) loss versus the number of tokens ingested in total and all architectures run with the same time budget.

We observe that varying the transformer *shape* has only minimal impact on the final loss after 24 hours. Models with more parameters learn more efficiently, as their MLM loss decreases faster on a per-gradient basis. However, smaller architectures make up for their slower learning efficiency by higher throughput, and thus process more tokens over the limited budget. Figure 1 shows that different architectures are unpredictable throughout an initial stage of training (the first 1B tokens), after which the per-token efficiencies differ by only a multiplicative constant (a horizontal shift due to the log axis). This constant depends almost entirely on the model size, not model shape, so that all choices reach a MLM loss around 1.9 at the end of training. As in Kaplan et al. (2020) we observe an optimal model *size* for this compute budget (at a budget of around 4B tokens), but gains from model size optimization are small at this compute scale.

Exploiting the scaling law. The scaling laws seem to bar us from making large gains via major changes to the transformer size and shape, as per-token performance is tightly coupled to model size. While this principle closes one door for scaling down efficiently, it opens another; Because per-gradient efficiency remains nearly constant for all models of the same size, we can exploit scaling laws by searching for architectural choices that speed up computation while keeping model size roughly constant.

A number of obvious optimizations fall into this category, and we describe them below, in addition to several other tweaks that provide marginal but worthwhile/free gains. We note that, for each modification, we tabulate gains in MLM loss and downstream accuracy in separation in the appendix, and focus on aggregate changes in the main body.

Attention Block: We disable all QKV biases (Dayma et al., 2021). This exploits the scaling law by removing a layer of computation, making the forward and backward pass somewhat faster, while keeping the model size nearly constant. We find that we could decrease gradient costs by reducing the number of attention heads (Merity, 2019; Araabi & Monz, 2020; Liu et al., 2021b; Javaheripi et al., 2022), as this parallelizes better on the GPU and provides a slight performance boost. Yet, reducing the amount of heads also decreases finetuning performance, so we ultimately keep all 12 heads. We further keep the original multi-head self-attention mechanism.

Feedforward Block: We find empirical gains from disabling all linear layer biases (Dayma et al., 2021). Just as for the attention layers, this leverages the scaling law by accelerating gradient computation without noticeable impacts on model size. As a result, we get higher throughput without compromising the rate at which the model improves. We keep the

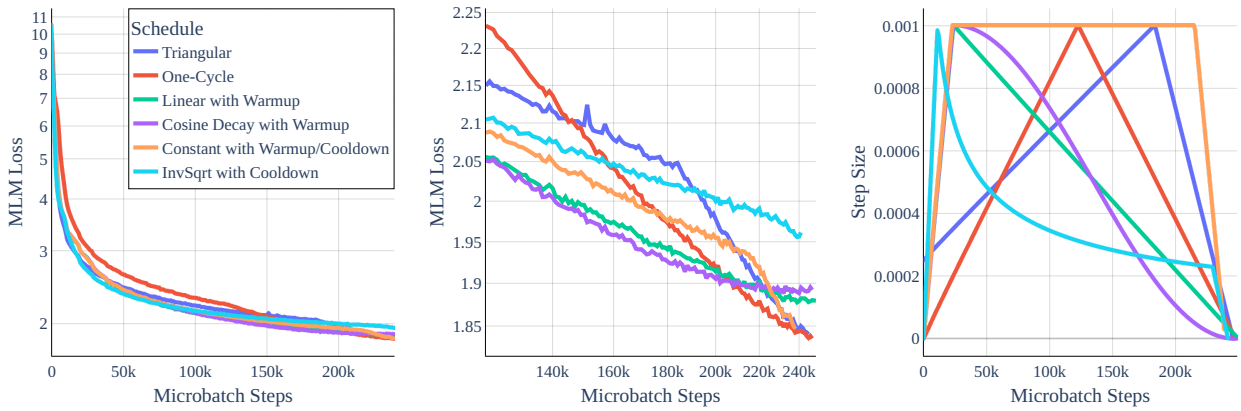

Figure 2. Learning Rate Schedules, for optimal peak step size for each scheduler. Although globally many schedule result in similar behavior, we see in the zoom in the middle, that differences do exist. The right side shows the scheduled step sizes. Triangular- and trapezoidal one-cycle schedules have better end-time behavior, possibly due to the quick annealing and overall greater progress.

original feedforward block largely unchanged.We do see small improvements from re-ordering the block into a gated linear unit (Dauphin et al., 2017). In contrast to other work, e.g. (Black et al., 2022), we do not increase the number of parameters in the FFN block to compensate for the halving of the hidden dimensionality due to gating.

Embedding: We implement scaled sinusoidal positional embeddings as in Hua et al. (2022), finding incremental benefits over learned or unscaled sinusoidal embeddings. We include normalization at the end of the embedding block.

Layer Structure: As observed in many studies, we find that pre-normalization with Layer Norms is beneficial over post Layer Norms (Baevski & Auli, 2018; Xiong et al., 2020). We note that the key effect of pre-normalization is to stabilize training and enable larger learning rates and reduced warmup, pre-normalization by itself has no effect on performance. Further, we find minimal gains by increasing the number of layers to 16.

Head Block: We find that we can remove the nonlinear head without ill effect. We can further drop the decoder bias (Radford et al., 2019) and gain in memory using sparse token prediction (Liu et al., 2019; Izsak et al., 2021). We add a final Layer Norm to stabilize training further.

C.3. Modifying the Training Setup

We study the impact of training hyper-parameters on the BERT-base architecture. The original BERT training recipe understandably results is poor model performance in the cramming setting, and so we revisit a number of choices.

Objective: We train with only masked language modeling on fully packed blocks of tokens with a masking rate of 25% and the original setup of Devlin et al. (2019) where 10% of all masks are filled with random words and 10% unchanged. We find the increased masking rate provides benefits at almost no extra cost, as the original 15% results in inopportune tensor shapes, whereas 25% of micro-batch size and sequence length neatly falls on the next power of 2.

Choice of Optimizer: We keep Adam (Kingma & Ba, 2015) as the optimizer of choice, with weight decay of 0.01 as described in (Loshchilov & Hutter, 2017) (i.e. "AdamW"), $\beta_1 = 0.9, \beta_2 = 0.98$ and $\varepsilon = 10^{-12}$. To stabilize training at no extra cost, we include gradient clipping at 0.5.

Learning Rate Schedule and Peak: Following the advice of Izsak et al. (2021), we re-scale the learning rate schedule so that it is tied to our budget and the learning rate decays as the budget reduces to zero. Interestingly, we observe in Figure 2 that while globally a large number of learning rate shapes lead to similar reductions in loss, we find that we can make some gains through the choice of schedule. We find that a simple one-cycle learning schedules (Smith & Topin, 2018), with a peak learning rate of 10^{-3} lead to minimal pretraining loss within our budget, with the optimum being a triangular shape (denoted "triangular" in Figure 2) that mimics a long warmup period with a quick decay.

Batch Size Schedule: A particularity of our setting is that, due to being limited to a single GPU, the micro-batch size that finds its way onto this GPU (96 for most experiments) is several times smaller than the optimal batch size. We find that

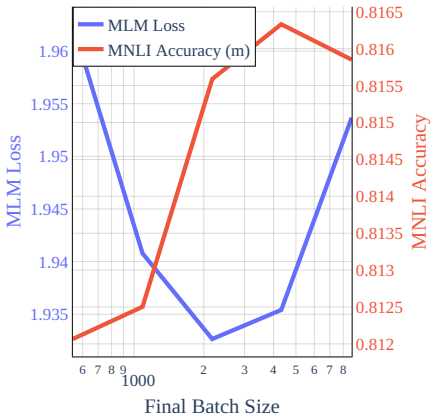

Figure 3. Optimal batch sizes for pretraining and downstream performance (measured on MNLI) are different.

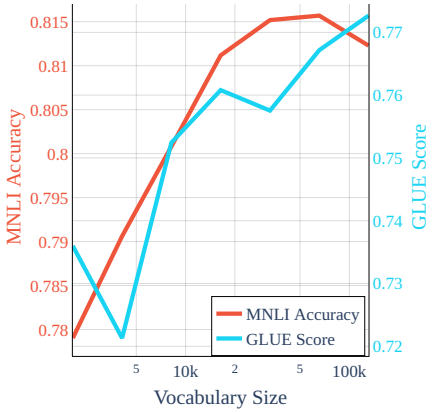

Figure 4. Vocabulary Size versus GLUE Score and MNLI Accuracy for models trained in the cramming regime on `bookcorpus-wikipedia` data.

the optimal batch size in this setting is around 2048 for minimal pretraining loss, but around 8192 for maximal downstream performance, see Figure 3. We accumulate gradients and only perform an update every 85 forward/backward passes. For the larger A4000 and A6000 cards, this corresponds to micro-batch sizes of 128 and 512 for the same batch size of 8192, which we again accumulate.

Fortunately, we can find small speedups by using an aggressive batch size schedule; we increase the number of averaged micro-batches linearly over the course of training, reaching the full batch size of 8192 after 60% of training. This results in more progress earlier in training, and leads to a small benefit to performance.

Dropping Dropout The original BERT model of Devlin et al. (2019) includes dropout as in Vaswani et al. (2017), which prevents overfitting when training data is small relative to total compute budget. While it can be helpful as a regularizer, dropout effectively reduces the number of gradient updates seen by each parameter, as updates do not occur when the associated feature is dropped. At the same time, update runtime is not strongly effected by the presence of dropout, and so dropout results in a net reduction in updates per second. In the cramming setting, training data is large compared to compute. Overfitting is not possible due to the single epoch schedule, and we disable dropout during pretraining (Brown et al., 2020) to maximize the number of parameter updates. We re-enable dropout during downstream fine-tuning with a dropout value of 0.1.

C.4. Optimizing the Dataset

We found above that scaling laws create a barrier to making major gains (beyond computational efficiencies) with architectural modifications. However, scaling laws do not preclude us from training on better data. Once we have exhausted our ability to train on more tokens per second, we should seek to train on better tokens.

We consider two data based pathways to better down-scaling. First, we can filter, process, or sort the existing data

in various ways. Second, we can swap our data source. To this end, we experiment with replacements for the `bookcorpus-wikipedia` dataset. We test several subsets of *The Pile* (Gao et al., 2020). We draw random subset from all sources, denoted `pile` and one containing raw text from only *Gutenberg*, *Books3* and *Wikipedia (en)*, denoted `pile-N`. From these Pile datasets we tokenize the first 4×10^6 entries to generate enough tokens for our single pass. Another popular source of data is C4, the colossal, cleaned version of Common Crawl (Raffel et al., 2020), from which we stream the first 20×10^6 entries. Finally, we also include the 2019 release of the OSCAR dataset (Suárez et al., 2019), denoted by `oscar`. For each data source we regenerate its own WordPiece tokenizer as described in Appendix C.1.

Of these four sources, we find the natural split of The Pile to perform best in terms of downstream GLUE performance out-of-the-box. However, we can further improve performance through additional processing. We evaluate deduplication as described in Lee et al. (2022) via exact substring deduplication (of substrings of length 75), but find this not to reliably help in downstream performance in our case, see Table 2. We then test filtering for incompressible data. We use the tokenizer itself to remove all training sequences from each data source that cannot be compressed well; we simply set a threshold t, here $t = 0.25$, and drop all entries from the dataset where the number of tokens in the entry is larger than t times the number of raw characters. This removes, for example, sequences consisting of hard-to-compress HTML.

Finally we experiment with sorting, where we re-order all tokenized sequences by some metric. We find the optimal metric to be sentence length and we sort so that sequences containing short sentences come first. This is empirically beneficial over e.g. sorting all sequences by their average (unigram) token prevalence.

Overall, wins from post-processing in this manner are noticeable, leading in aggregate to an improvements of 2% over the original dataset and we choose The Pile with both filtering and sorting as our new dataset going forward.

Vocabulary Size We also check whether the original vocabulary size of 30522 described in (Devlin et al., 2019) is optimal in the crammed regime. A priori, this might not hold: The smaller the vocabulary, the fewer, unique tokens and relationships between unique tokens have to be learned during training. On the other hand, increasing the vocabulary size compresses data further (albeit vanishingly after some point), which would allow for more information to be compressed into the fixed number of tokens that can be ingested during the crammed training run. In Figure 4, we find that for `bookcorpus-wikipedia` data, larger vocabulary sizes correlate with larger average GLUE score, although the effect is plateauing for the MNLI task around the original 32768 vocabulary size. Moving forward, we hence keep this vocabulary size.

C.5. Evaluating the Model Further

Where lie the limitations of the crammed model? The crammed model mostly works, even for smaller datasets, such as STSB and MRPC. The largest difference is observed on CoLA, the corpus of linguistic acceptability (Warstadt et al., 2019), see Table 4. This behavior is intriguing and we offer two hypotheses. First, it is conceivable that the chosen global hyperparameters for finetuning are a bad fit for CoLA in particular. CoLa performance can be brittle with respect to hyperparameter, with Jiao et al. (2020) training longer only on CoLA or Joshi et al. (2020) training less only on CoLA. Nevertheless, for BERT, a set of global hyperparameters exists, pointing at a deficiency in the crammed model. As a second hypothesis, the improvements across GPUs imply that these models need to process more text before they memorize enough data to do well on CoLA. This would be in contrast to Liu et al. (2021d), who find that CoLA is learned relatively quickly compared to other downstream tasks when probing intermediate BERT checkpoints. Finally, deficiencies on CoLA in particular are also common in approaches that distill BERT into smaller architectures (Sun et al., 2019; Turc et al., 2019; Mukherjee et al., 2021), which might come with limited capacity for linguistic acceptability.

Ablation - Which Changes Really Mattered? In Table 5 we provide an ablation study summarizing all changes discussed in this work. We group modifications, as in previous sections into the three groups of architecture, training and data and ablate each group by resetting all modifications to the original BERT recipe. Here, we find that we first have to make

Table 4. GLUE-dev performance of baseline BERT to crammed model. T is the number of tokens ingested during training in billions. Avg. Score is all scores excluding CoLA, GLUE is the full avg. over the same tasks as in Devlin et al. (2019).

	T(10^9)	CoLA	Avg. Score	GLUE
Bert-Base	-	**56.5**	**84.0**	**80.9**
Crammed (2080ti)	4.3	48.3	83.1	79.2
Crammed (A4000)	4.5	48.6	83.0	79.2
Crammed (A6000)	8.7	51.8	**84.0**	80.4

Table 5. Ablation study, which improvements were most important? The first group shows an ablation where one component of the final combination of training, architecture, and data modifications (the crammed BERT model) is replaced by the original setup. Here, we find that modifications in training and architecture have to co-occur, as returning to the original training setup or architecture each results in failure. As such we also include a row with *minimal training* modifications (dropout disabled, cosine decay to zero within budget with warmup, fixed batch size of 8192) and a row with *minimal architecture* modifications (Pre-normalization, sparse activations, Layer Norm $\varepsilon = 10^{-6}$). This ablation is for the A6000 variant, see other results in Table 9.

	MNLI	SST-2	STSB	RTE	QNLI	QQP	MRPC	CoLA	GLUE
crammed BERT	**84.0/84.4**	**92.3**	**87.0**	**57.4**	90.0	**87.7**	**89.0**	**51.8**	**80.4**
+ original data	82.6/83.2	91.7	86.0	55.6	**90.1**	87.0	85.6	45.4	78.6
+ original train	50.9/49.9	82.2	15.6	49.8	58.6	66.4	73.6	7.5	50.5
+ original arch.	-/-	-	-	-	-	-	-	-	-
+ minimal train mod.	80.8/81.6	90.6	86.6	54.5	88.2	86.6	88.6	44.3	78.0
+ minimal arch. mod.	83.1/83.6	91.6	85.3	57.0	89.9	87.0	85.7	41.8	78.3

minimal modifications in any case, as modifications to architecture, such as PreNorm layer structures also in turn allow the more aggressive learning rate described in the training setup - without both, training fails when going back to either the original architecture or results in a model close to random performance when going back to the original training.

Taking this into account, we also include an ablation with *minimal training* modifications (dropout disabled, cosine decay to zero within budget with warmup, fixed batch size of 8192) and with *minimal architecture* modifications (Pre-normalization, sparse activations, Layer Norm $\varepsilon = 10^{-6}$). Comparing these variants, we ultimately find about two percentage points gained in average GLUE score through either architectural changes, data changes, or training modifications.

D. Negative Results

This section collects negative results for each paragraph in the main body. The overall paragraph structure follows the main body directly.

D.1. Implementation Details

We run all experiments and ablation studies with the same setup of automated mixed precision (Micikevicius et al., 2018) for standard 16- and 32-bit floating point precision over full 32-bit float, scaled 16-bit (Rasley et al., 2020) and pure `bfloat16` (Wang & Kanwar, 2019). We find no benefit from offloading (Ren et al., 2021; Rasley et al., 2020) in our setting.

Initial Data Setup The used language is English (Bender, 2019). We found no significant change in performance with BPE (Sennrich et al., 2016) or SentencePiece with Unigrams (Kudo, 2018; Kudo & Richardson, 2019). Smaller vocabulary sizes ($2^{12}, 2^{13}, 2^{14}$) resulted in worse performance, while larger vocabulary sizes (2^{16}) we not reliably better. We pack tokenized data into randomized sequences of length 128 and separate unrelated fragments by `<sep>`.The performance impact from dropping this separator was minimal. No impact was observed from including a `<cls>` token in pretraining.

D.2. Modifying the Architecture

Exploiting the scaling law. We find no improvements when using a funnel-transformer architecture (Dai et al., 2020; Nawrot et al., 2022), when dropping FFN layers (Sridhar et al., 2022), or when using recurrent layers (Lan et al., 2019), even when trained with BPTT as in Schwarzschild (2021). Rescaling architectures to be deep-narrow (Tay et al., 2021; Wies et al., 2021) provides no gains.

Attention Block: We find no benefits from replacements to the softmax operation (Richter & Wattenhofer, 2020). We further keep the original multi-head self-attention mechanism. A large amount of work has been focused on efficient attention (Sukhbaatar et al., 2019; Beltagy et al., 2020; Wang et al., 2020a; Liu et al., 2021c) and studies of efficient attention (Tay et al., 2020a;b). But, because we set the maximal sequence length to 128, attention complexity is less of a concern in our setting. To verify this, we implement the recently proposed FLASH mechanism (Hua et al., 2022), but find no benefits. We further experiment with Fourier attention as proposed in Lee-Thorp et al. (2021), but find no improvements. We find rotary embeddings (Su et al., 2021; Black et al., 2022), to provide small benefits, but these are evened out by the drop in speed, so we ultimately decide against these.

Feedforward Block: We keep the original feedforward block largely unchanged, finding no benefits from changing to another activation than GELU.

Embedding: We see no improvements from decoupling the input and output embeddings (Chung et al., 2020). The suggestion from Lan et al. (2019) to factorize the input embedding provides no gains in our setting.

Layer Structure: We see no additional benefit from other variants of this modification, such as (Liu et al., 2020b; Shleifer et al., 2021). Further, replacing Layer Normalization with RMS Normalization provides no gains (Zhang & Sennrich, 2019).

D.3. Modifying the Training Setup

Objective: We see no improvement from masking at larger rates, e.g. at 40% as proposed in (Wettig et al., 2022), see Appendix. We see no difference enabling or disabling the mentioned 20% rule. We evaluate other functions for the masked-language objective, such as mean-squared error (Hui & Belkin, 2021) or L1 loss, but find no benefits.

Choice of Optimizer: We find no noticeable change in varying these parameters in reasonable amounts, e.g. $\varepsilon = 10^{-6}$, $\beta_1 = 0.9, \beta_2 = 0.999$. We test other first-order adaptive optimizers (Shazeer & Stern, 2018; Liu et al., 2020a) but find no advantages in our setting. We further find no advantages using higher-order optimizers (Yadav, 2020; Anil et al., 2021), but note that especially for higher-order optimizers there is a greater amount of variability in implementation.

Batch Size Schedule: We also experiment with automatic and adaptive batching rules (De et al., 2017; Bollapragada et al., 2018a;b), but find that the best results from these adaptive schedules resemble the fixed linear schedule. For simplicity we just stick to the simpler linear schedule.

Dropping Dropout Further, we experiment with length curricula (Li et al., 2022) (see appendix) and token dropping (Hou et al., 2022), but find no gains in our setting.

E. Limitations

In this work, we limited our investigation to transformer-based architectures trained with MLM objectives. However, we do think that the general task of cramming posed in Section 2 is interesting even when relaxing these constraints. There have been a number of modifications proposed to the objective in particular (Joshi et al., 2020; Bao et al., 2020; Bajaj et al., 2022; Tay et al., 2022b). While Artetxe et al. (2022) and Wang et al. (2022) find MLM still to hold up well as a pretraining objective, other suggestions such as ELECTRA (Clark et al., 2019; 2020; He et al., 2021) could be employed which might be beneficial for crammed models. Also, the optimal architecture might not be transformer-based (Merity, 2019; Fusco et al., 2022; Peng, 2021; Peng et al., 2023).

Other Modifications A few recent developments not included in this study are Roy et al. (2022), Shen et al. (2022), and Mindermann et al. (2022). Modifications further not included in this study are more involved initialization (Zhu et al., 2021), additional objective modifications (Müller et al., 2019), progressive growth (Gu et al., 2021; Shen et al., 2022), convolutional variants (Iandola et al., 2020; Chelombiev et al., 2021; So et al., 2021), sequence recurrence (Lei et al., 2022) and TUPE embeddings (Ke et al., 2020).

Relationship between MLM Loss and Downstream Performance Improvements in pretraining loss do not have to correspond to improved downstream performance (Tay et al., 2021; Wang et al., 2022). In the presentation of this work, we have chosen not to make this a focus of our discussion. We show results with improved pretraining loss, e.g. Figure 1 along-side results for downstream performance, e.g. Figure 4, and only comment on the discrepancy between MLM pretraining loss and downstream GLUE performance in a few locations.

Nevertheless, we have cross-checked most improvements discussed in this work for their utility in downstream applications, even when only results for pretraining loss are shown. Additional results, showing both pretraining loss and downstream performance on MNLI can be found in Table 11 for architectural changes and Table 12 for training modifications. For data modifications, changes in pretraining loss not very meaningless, so we always compare data modifications in terms of their effect on downstream performance. In Figure 5 we show a scatter plot of downstream versus pretraining performance, summarizing both tables.

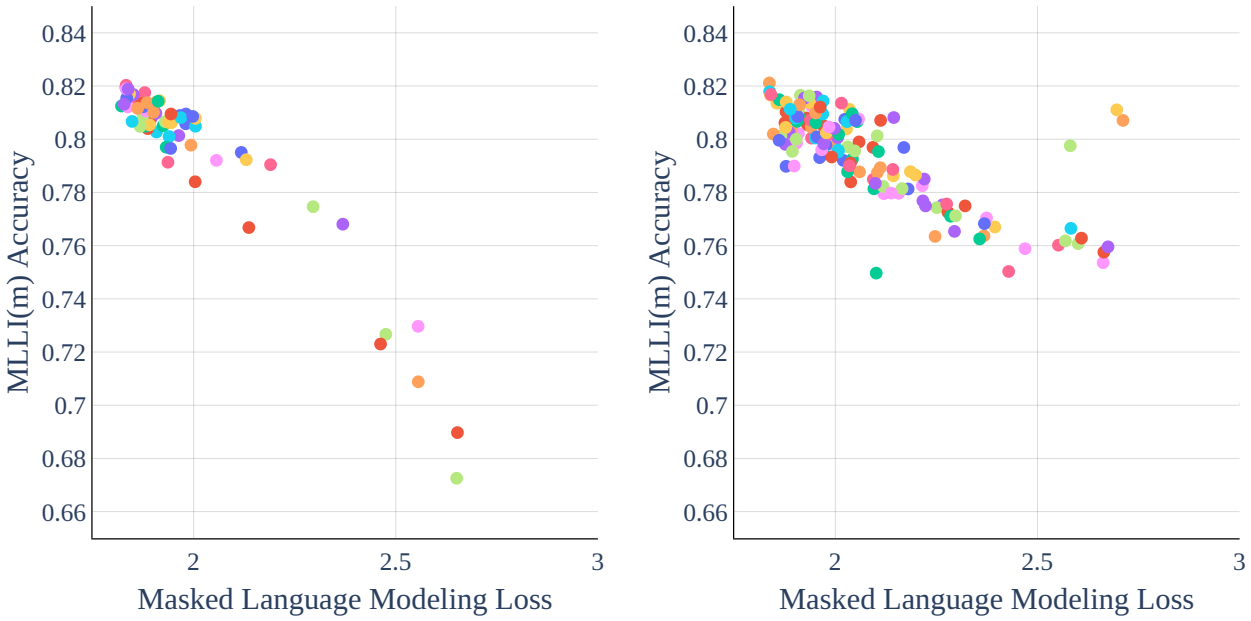

Figure 5. Relationship between pretraining loss and downstream performance. **Left:** for all architecture variations considered (see Table 9 for details). **Right:** for all training variations considered (see Table 12 for details).

F. SuperGLUE Comparisons

Do these results hold only for the GLUE benchmark? To investigate this question, we also compare against the superGLUE benchmark (Wang et al., 2019). We evaluate downstream performance on this new set of tasks with the same (global) hyperparameters used for GLUE, and within the same downstream epoch constraints. Tasks are solved as sequence classification problems and other tasks that require feature engineering are skipped. We do this for both the original BERT model and the crammed model, although noting that BERT numbers are accordingly sub-optimal, compared to numbers achieved after hyperparameter-tuning and feature engineering in Wang et al. (2019). Nevertheless these results show that the crammed model is similarly performant on new tasks as the original BERT model, when similar effort is applied. Both models could be used as the starting point for an investigation of a new task.

Table 6. SuperGLUE results for the crammed model (A6000 variant) and the base BERT model from Devlin et al. (2019). Downstream finetuning for both models uses the same hyperparameters as for GLUE without tuning. These hyperparameters follow the rules laid out in Section 2 and not the usual, extensive tuning for superGLUE. Tasks are solved as sequence classification problems and other tasks that require feature engineering are skipped.

	AX-b	AX-g	CB (f1)	CB (Acc)	COPA	MultiRC	RTE	WiC	WSC	BoolQ	Avg.
Bert-Base (Fully trained)	**10.8**	50.3	**60.8**	**75.0**	49.0	60.5	**58.8**	**64.7**	**63.5**	72.4	**56.6**
crammed BERT	10.4	**50.6**	49.3	70.5	**51.5**	**61.5**	57.4	61.8	55.3	**74.3**	54.2

G. Additional Information

Additional results concerning architecture modifications can be found in Table 10 and Table 11. Additional results for training modifications can be found in Table 12. An extended version of the comparison of different data sources and processing options, separated into scores for each GLUE task can be found in Table 7. Other, miscellaneous, variations for data can be found in Table 8.

Not all results remarked on in Appendix D are accompanied by raw results in this appendix, but can be computed using the provided implementation. Note that baseline settings and implementation details change between tables, making each result only comparable to other results within the same table.

Table 7. Variations of data source and data processing. Shown are GLUE scores for all tasks and in aggregate. `bw` denotes `bookcorpus-wikipedia`, `c4` is C4 (colossal-cleaned-common-crawl) (Raffel et al., 2020), `oscar` is the 2019 release of the OSCAR dataset (Suárez et al., 2019), (`https://oscar-project.org/`) and `owt` is the opens-source replication of the open-webtext corpus (`https://huggingface.co/datasets/openwebtext`). `pile` is a random subset of The Pile (Gao et al., 2020). `pile-N` is the subset drawn only from the natural sources in the Pile (Gutenberg, books3, wikipedia). `roots` is a subset of the English portions of the ROOTS dataset (Laurençon et al., 2023). Filtering denotes a removal of hard-to-tokenize sequences at $t = 0.25$ as described in the main body. Sorting denotes sorting by number of sentences per sequence. DD is deduplication as in Lee et al. (2022) with a substring length of 75. All values are based on runs on A6000 GPUs.

Source	Filt.	Sort.	DD	MNLI (m/mm)	SST-2	STSB	RTE	QNLI	QQP	MRPC	CoLA	GLUE
bw	✗	✗	✗	82.8/82.9	91.8	85.4	53.8	90.0	86.9	85.3	43.9	78.1
bw	✓	✗	✗	82.9/83.4	91.6	86.0	51.8	90.0	87.2	86.3	49.3	78.7
bw	✓	✓	✗	82.7/83.3	91.7	85.5	52.0	90.2	87.0	87.8	48.6	78.8
c4	✗	✗	✗	83.8/84.1	92.7	87.0	53.8	90.1	87.6	87.5	16.0	75.9
c4	✓	✗	✗	84.1/**84.7**	91.7	86.7	54.9	90.4	87.6	88.6	44.9	79.3
c4	✓	✓	✗	83.7/84.5	92.3	87.0	54.9	90.1	87.4	88.8	42.3	79.0
oscar	✗	✗	✗	83.9/84.2	92.4	87.3	54.9	89.8	**87.8**	88.0	43.9	79.1
oscar	✓	✗	✗	83.9/84.3	92.7	87.0	54.5	90.0	87.7	88.5	44.6	79.2
oscar	✓	✓	✗	83.7/84.1	92.5	86.6	**56.5**	89.9	87.4	89.3	42.5	79.2
oscar	✓	✓	✓	84.0/84.2	92.5	87.2	56.7	89.7	87.5	88.1	50.6	**80.1**
owt	✗	✗	✗	84.2/84.3	**92.8**	87.2	54.9	90.2	87.6	88.9	48.2	79.8
owt	✓	✗	✗	84.4/**84.7**	**92.8**	86.7	55.6	**90.6**	87.5	88.7	45.9	79.7
owt	✓	✓	✗	84.3/84.6	92.2	**87.4**	55.2	90.0	87.4	88.8	46.9	79.6
owt	✓	✓	✓	**84.6**/84.6	92.3	86.0	55.2	90.5	87.5	87.9	51.0	80.0
pile	✗	✗	✗	83.0/83.1	91.6	85.2	54.9	89.3	87.3	87.5	41.9	78.2
pile	✓	✗	✗	84.1/84.5	92.1	86.1	56.0	90.2	87.7	88.5	44.7	79.3
pile	✓	✓	✗	84.2/84.5	92.5	87.0	53.6	90.0	87.6	89.9	52.1	**80.1**
pile	✓	✓	✓	83.9/84.5	92.0	87.2	56.3	89.4	87.6	89.4	49.9	80.0
pile-N	✗	✗	✗	83.3/83.5	92.2	84.6	56.3	89.9	87.3	86.2	49.9	79.2
pile-N	✓	✗	✗	83.9/84.3	92.5	85.9	54.5	90.5	87.5	88.2	50.9	79.8
pile-N	✓	✓	✗	83.8/83.9	92.1	86.5	53.4	90.2	87.2	**89.1**	**54.8**	**80.1**
roots	✗	✗	✗	83.8/84.1	92.1	86.3	54.3	90.2	87.5	88.6	34.8	78.0

Table 8. Miscellaneous Variations of the Data Processing setup, using `pile` data, filtered and sorted, as a baseline. All values are from runs on RTXA6000 cards.

	MNLI (m/mm)	SST-2	STSB	RTE	QNLI	QQP	MRPC	CoLA	GLUE Score
Baseline	83.8/84.3	92.3	87.0	55.2	89.8	87.6	88.7	47.3	79.6
65336 tokens in vocab.	83.6/84.1	91.6	87.2	56.0	89.8	87.6	88.9	44.1	79.2
Include `[CLS]` in pretrain	84.4/84.8	92.3	86.3	54.5	90.5	87.7	86.8	44.5	79.1
SentencePieceBPE	83.4/83.7	91.5	84.8	53.1	89.7	87.5	86.8	47.3	78.6
Sort by num. chunks in seq.	83.9/84.4	92.0	84.7	54.2	90.1	87.7	87.7	43.3	78.7
Sort by unigram prob.	83.2/83.7	91.9	86.2	53.8	88.7	87.6	87.5	44.9	78.6

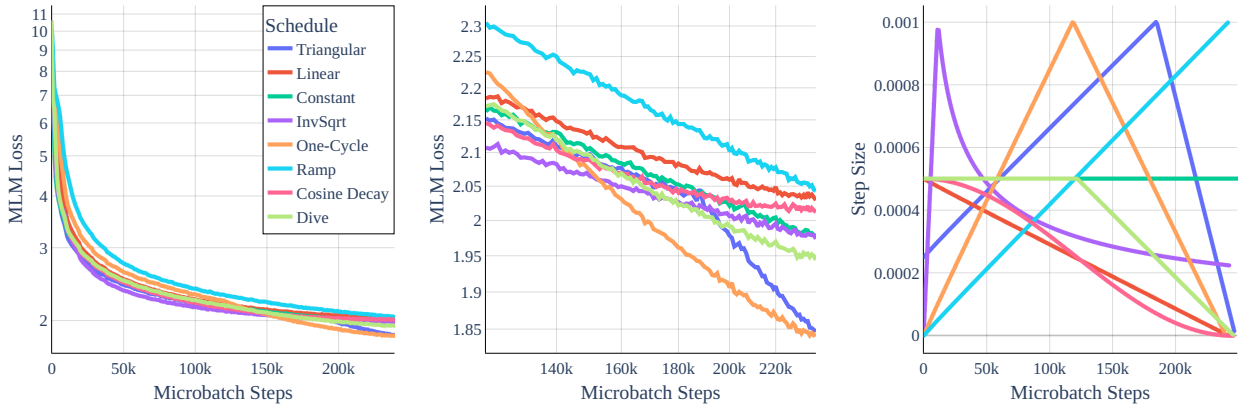

Figure 6. Extended version of Figure 2, including additional learning rate schedules.

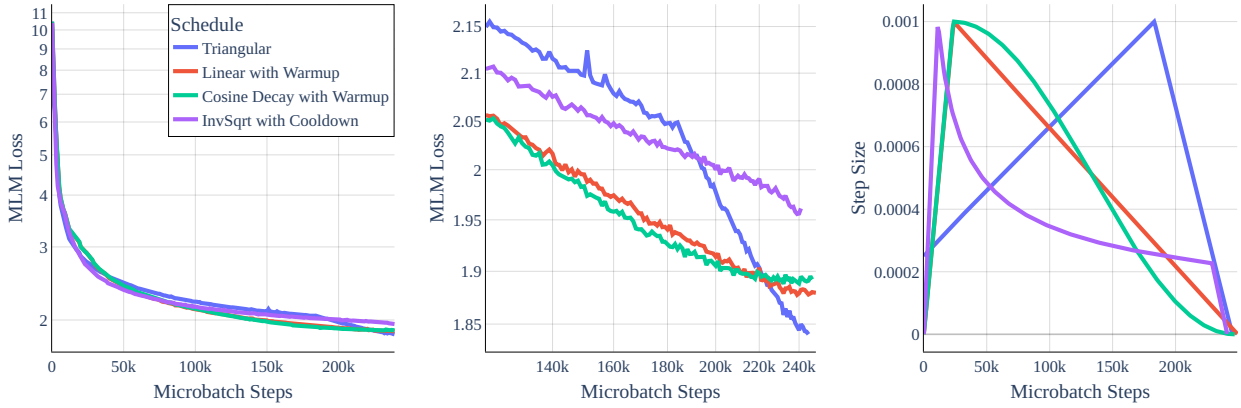

Figure 7. Variant version of Figure 2, including additional learning rate schedules with warmup and cooldown.

G.1. References for Table 1

The maximal floating point operations referenced in Table 1 are based on the following published numbers.

- For TPU specs, according to `cloud.google.com/tpu/docs/system-architecture-tpu-vm` we find 275 TFLOP/s in `bfloat16` precision for the TPUv4 and 123 TFLOP/s for the TPUv3, each per chip.
- The V100 peak performance is given as 125 TFLOP/s in `images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf` in "TFLOPS of mixed precision". Some NVIDIA datasheets also reference TFLOP/s with sparsity, which are not applicable in the context of this work.
- The Titan RTX comes out at 130.5 TFLOP/s, in "Peak FP16 Tensor TFLOPS with FP32 Accumulate" as described in `images.nvidia.com/aem-dam/en-zz/Solutions/geforce/ampere/pdf/NVIDIA-ampere-GA102-GPU-Architecture-Whitepaper-V1.pdf`.
- For the A6000, we find 154.8 "Peak BF16 Tensor TFLOPS with FP32 Accumulate" also in `nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.pdf`.
- A4000 performance is actually less clear. Its datasheet at `nvidia.com/content/dam/en-zz/Solutions/gtcs21/rtx-a4000/nvidia-rtx-a4000-datasheet.pdf` only describes 153.4 TFLOPS "using the new sparsity feature" - which is not applicable in our context and does not reflect the card's actual performance. We estimate its actual numbers to be 88.45 TFLOP/s, based on it containing 192 tensor cores, compared to 336 for the A6000.

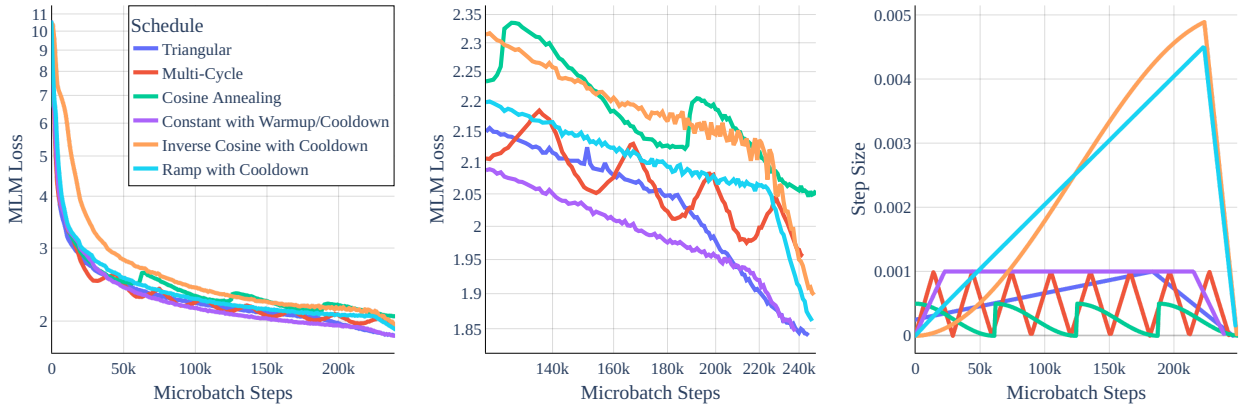

Figure 8. Variant version of Figure 2, including periodical learning rate schedules, as well as "ramping" schedules with cooldown.

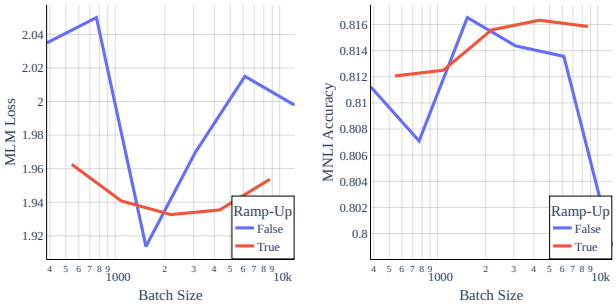

Figure 9. Partial variations of batch sizes with and without linear ramp-up in extension of Figure 3. All experiments run with the training setup described in Section 3.2 for a day on a single GPU with mixed precision. Batch size is 4036 and dataset is `bookcorpus-wikipedia`. Downstream evaluation as described in Section 4. All values for pretraining on an A4000. Note the discrepancy between optimal pretraining batch size and optimal batch size for evaluation on MNLI when ramp-up is used, but also note that differences are overall barely significant.

- For the RTX2080ti, the whitepaper at `images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf` reports 53.8 "peak FP16 Tensor TFLOPS with FP32 Accumulate" for the reference edition.

All total exaFLOP numbers are then computed based on these TFLOP/s numbers over the training time period described in each work.

Table 9. Extension of Table 5, including results on the other GPU types.

	MNLI	SST-2	STSB	RTE	QNLI	QQP	MRPC	CoLA	GLUE
Trained for 1 day on a **2080ti**									
crammed BERT	82.5/82.8	91.6	86.2	56.7	89.1	87.1	88.3	48.3	79.2
+with original data	81.5/82.1	91.2	85.6	53.1	88.5	86.8	87.5	44.7	77.9
+with original train	51.6/50.7	82.5	10.5	52.0	58.7	66.1	72.2	3.8	49.8
+with original arch.	32.7/33.0	50.9	-0.0	50.0	49.5	0.0	81.2	0.0	33.0
+with minimal train mod.	79.2/79.4	89.9	84.5	54.7	85.9	85.5	87.3	41.1	76.4
+with minimal arch. mod.	82.1/82.4	91.1	85.2	55.8	88.5	86.9	87.6	43.8	78.2
Trained for 1 day on an **A4000**									
crammed BERT	82.6/83.2	91.9	86.6	56.7	88.9	87.1	88.9	46.4	79.1
+with original data	81.6/82.1	91.2	85.2	53.1	88.6	86.7	86.9	44.3	77.8
+with original train	50.9/50.0	81.9	17.5	50.5	58.8	64.7	74.2	8.5	50.8
+with original arch.	32.3/32.4	50.9	-4.1	47.3	50.3	0.0	81.2	0.0	32.3
+with minimal train mod.	79.4/79.2	89.3	84.4	55.2	85.5	85.1	87.0	40.8	76.2
+with minimal arch. mod.	81.8/82.5	91.6	84.7	55.8	88.9	86.7	87.8	42.2	78.0
Trained for 1 day on an **A6000**									
crammed BERT	**84.0/84.4**	**92.3**	**87.0**	**57.4**	90.0	**87.7**	**89.0**	**51.8**	**80.4**
+ original data	82.6/83.2	91.7	86.0	55.6	**90.1**	87.0	85.6	45.4	78.6
+ original train	50.9/49.9	82.2	15.6	49.8	58.6	66.4	73.6	7.5	50.5
+ original arch.	-/-	-	-	-	-	-	-	-	-
+ minimal train mod.	80.8/81.6	90.6	86.6	54.5	88.2	86.6	88.6	44.3	78.0
+ minimal arch. mod.	83.1/83.6	91.6	85.3	57.0	89.9	87.0	85.7	41.8	78.3

Table 10. Additional raw results for experiments considered in the main body. This table contains architecture variants for a prelimary architecture setup which contained 4 heads in the attention block, 12 layers and included rotary embeddings. First two blocks: Architectural variants as discussed in Section 3.1 (but for this preliminary variant). Third block: Ablation study of this model. All experiments run with the training setup described in Section 3.2 for a day on a single GPU with mixed precision. Batch size is 4032 and dataset is `bookcorpus-wikipedia`. Downstream evaluation as described in Section 4. All values for pretraining on an A4000.

Name	MLM Loss	MNLI-m	MNLI-mm	Tokens/Second
Modified Transformer	1.89	81.02	81.35	50946
DeepNarrow (12 Layers)	1.94	80.90	80.97	78396
DeepNarrow (24 Layers)	1.98	80.78	81.14	41289
$E = 128$	2.14	76.68	77.62	53267
FFN every 2 blocks	1.93	80.43	80.97	64774
FFN every 3 blocks	1.97	80.44	80.93	71634
FFN every 4 blocks	2.00	80.03	79.67	73319
$H = 512$	1.93	80.61	80.93	83718
$H = 1024$	1.95	80.07	80.68	32004
4 Layers	2.00	78.45	79.00	137127
6 Layers	1.93	79.49	79.82	96156
8 Layers	1.89	81.11	81.08	74248
10 Layers	1.89	81.02	81.21	61431
16 Layers	1.92	81.39	82.10	39406
24 Layers	2.01	80.64	80.97	26927
Recurrent (1-12)	2.40	77.46	77.81	52405
Recurrent (2-6)	2.04	80.45	80.73	53148
Recurrent (3-4)	2.00	80.78	81.33	51634
Recurrent (4-3)	1.98	80.95	81.26	51952
BERT-tiny	3.30	56.71	57.21	914694
BERT-mini	2.49	72.22	73.21	429593
BERT-Large (Izsak variant)	2.38	76.93	77.47	13448
Original BERT	7.54	35.45	35.22	41978
With decoder bias	1.89	80.97	81.20	51155
With $\varepsilon = 10^{-6}$ in Layer Norm	1.90	80.49	81.35	51728
Learned Embedding	1.88	80.51	81.03	52601
No Norm after Embedding	1.94	79.65	80.34	52175
No Final Norm	1.89	80.40	80.89	51207
No Skip of Head Transform	1.88	80.49	81.19	51728
No Rotational Embedding	1.88	80.91	81.52	53526
Post-LN	7.54	31.82	31.82	52270
With QKV bias	1.89	80.70	80.88	51112
With bias in Linear Layers	1.89	80.64	81.49	50584
12 Heads	1.88	81.75	81.99	47967

Table 11. Additional raw results for experiments considered in the main body for the final architecture variant. First two blocks: Architectural variants as discussed in Section 3.1. Third block: Ablation study of finally adopted model. All experiments run with the training setup described in Section 3.2 for a day on a single GPU with mixed precision. Batch size is 4096 and dataset is `bookcorpus-wikipedia`. Downstream evaluation as described in Section 4. All values for pretraining on an A4000.

Name	MLM Loss	MNLI	MNLI-mm	Tokens/Second
Modified Transformer	1.84	81.79	82.14	46431
DeepNarrow (12 Layers)	1.91	80.97	81.30	99717
DeepNarrow (24 Layers)	1.91	81.39	81.61	52558
$E = 128$	2.04	-	-	48468
FFN every 2 blocks	1.84	81.40	81.65	62134
FFN every 3 blocks	1.87	80.90	81.53	70685
FFN every 4 blocks	1.88	81.10	81.42	75163
$H = 512$	1.87	81.34	82.20	79116
$H = 1024$	1.94	80.63	80.97	28511
4 Layers	1.94	79.13	79.51	161034
6 Layers	1.87	80.48	80.84	115037
8 Layers	1.84	81.22	81.62	88652
10 Layers	1.82	81.25	82.31	71414
12 Layers	1.85	81.68	82.18	59346
18 Layers	1.90	81.02	81.82	40577
24 Layers	1.97	80.81	81.26	30455
Recurrent (1-12)	2.13	79.23	79.78	62318
Recurrent (2-6)	2.00	80.86	81.24	62677
Recurrent (3-4)	1.94	80.95	81.48	61772
Recurrent (4-3)	1.91	81.43	81.84	61596
BERT-Tiny Variant	3.51	56.10	56.60	1018443
BERT-Mini Variant	2.46	72.30	73.47	523061
BERT-Large Variant	2.12	79.50	79.84	17688
BERT-Large (Izsak variant)	2.37	76.81	77.56	13522
Original BERT	7.53	35.45	35.22	41362
With decoder bias	1.84	81.71	81.91	45996
With $\varepsilon = 10^{-6}$ in Layer Norm	1.83	81.55	82.13	45841
Learned Embedding	1.83	81.31	81.79	46608
No Norm after Embedding	1.89	81.38	81.15	46267
No Final Norm	1.85	80.67	80.87	46598
No Skip of Head Transform	1.83	82.03	82.19	46324
With QKV Bias	1.83	81.89	82.28	46469
With bias in Linear Layers	1.84	81.88	82.16	45629
4 Heads	1.88	81.22	81.77	40551
With Rotary Embedding	1.86	81.16	81.94	42257
Post-LN	7.54	35.21	35.17	46324
Fourier Attention	2.65	68.97	69.06	46634
GELU	1.83	81.94	82.17	47779

Table 12. Additional raw results for experiments considered in the main body for the final training variant, not otherwise mentioned. Batch size is 4096 and dataset is `bookcorpus-wikipedia`. Downstream evaluation as described in Section 4. All values for pretraining on an A4000.

Name	MLM	MNLi-m	MNLI-mm	Tokens/Second
Original training recipe	7.28	60.65	60.31	49264
With Izsak Training recipe	2.06	79.90	80.30	46869
Minimal Modifications	2.03	78.78	79.36	47346
+Larger LR	1.99	80.25	80.50	46524
+One Cycle, +Larger LR	1.84	82.12	82.55	46843
+One Cycle, +Larger LR, +Clipping	1.84	81.79	82.14	46303
Sequence Curriculum (10%,20%,30%,50%,75%)	3.02	70.06	70.77	29359
Sequence Curriculum (+unfolding)	1.87	80.13	80.04	46014
Sequence Curriculum (20%,35%,50%,65%,85%)	1.90	79.86	79.80	45804
Adafactor	1.86	81.36	82.22	45997
Adam (classic WD formulation)	7.44	32.28	32.39	49598
SGD	7.46	59.30	58.02	47678
RADAM	7.50	32.74	32.95	48812
With Dropout activated	1.97	80.95	80.98	45198
With MLM masking 20%	2.06	80.76	81.48	45944
With MLM masking 40%	2.70	81.11	81.30	43467
With MLM masking 60%	3.41	80.62	80.88	40756