MosaicBERT: How to Train BERT with a Lunch Money Budget

Jacob Portes [*] [1] **Alex Trott** [*] [1] **Sam Havens** [1] **Daniel King** [1] **Abhinav Venigala** [1] **Moin Nadeem** [2]
Nikhil Sardana [1] **Daya Khudia** [1] **Jonathan Frankle** [1]

Abstract

Although BERT-style encoder models are heavily used in NLP research, many researchers do not pretrain their own BERTs from scratch due to the high cost of training. In the past half-decade since BERT first rose to prominence, many advances have been made with other transformer architectures and training configurations that have yet to be systematically incorporated into BERT. Here, we introduce MosaicBERT, a BERT-style encoder architecture and training recipe that is empirically optimized for fast pretraining. This efficient architecture incorporates FlashAttention, Attention with Linear Biases (ALiBi), Gated Linear Units (GLU), a module to dynamically remove padded tokens, and low precision LayerNorm into the classic transformer encoder block. The training recipe includes a 30% masking ratio for the Masked Language Modeling (MLM) objective, bfloat16 precision, and vocabulary size optimized for GPU throughput, in addition to best-practices from RoBERTa and other encoder models. When pretrained from scratch on the C4 dataset, this base model achieves the downstream average GLUE (dev) score of 79.6 in 1.13 hours on 8 A100 80 GB GPUs at a cost of roughly \$20. We plot extensive accuracy vs. pretraining speed Pareto curves and show that MosaicBERT base and large are consistently Pareto optimal when compared to a competitive BERT base and large. This empirical speed up in pretraining enables researchers and engineers to pretrain custom BERT-style models at low cost instead of finetune on existing generic models. We open source our model weights and code.[1]

[*]Equal contribution [1]MosaicML [2]Work done while at MosaicML. Correspondence to: Jacob Portes <jacob@mosaicml.com>.

Work presented at the ES-FoMo: Efficient Systems for Foundation Models Workshop at ICML 2023, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

[1]Code can be found at github.com/mosaicml/examples, and model weights can be found at huggingface.co/mosaicml. An earlier version of this work appeared as a blogpost "*MosaicBERT: Pretraining BERT from Scratch for \$20*" (mosaicml.com/blog/mosaicbert).

1. Introduction

BERT has been the workhorse of modern natural language processing (NLP) since its introduction in 2018 (Devlin et al., 2018). Even in the era of large language models (LLMs), BERT-style encoder models are still quite relevant; for example, encoder models are used for vector database embeddings and retrieval augmented generation in tandem with LLMs (Karpukhin et al., 2020; Lewis et al., 2020; Izacard et al., 2022; Shi et al., 2023). In the past half-decade since BERT first rose to prominence, however, many advances have been made with other transformer architectures and training configurations that have yet to be systematically incorporated into BERT (Dauphin et al., 2017; Press et al., 2021; Dao et al., 2022). In this study we empirically show that these speed optimizations can successfully be incorporated into the classic BERT architecture and training recipe.

BERT-style models are typically trained in two stages: an initial self-supervised pretraining phase that builds general representations of language, and a subsequent supervised finetuning phase that uses those representations to address a specific task. The pretraining stage for BERT models has historically been computationally expensive; in the original BERT study, for example, the authors trained their models for 4 full days on 16 Google TPUs. In recent years, however, the time and cost to train BERT models has dropped significantly. One widely cited paper from 2021 successfully reduced the training time of BERT-Large to 24 hours on 8 Titan V-12 GPUs (Izsak et al., 2021), and another very recent paper trained a competitive BERT-Base model on a single consumer GPU in only 24 hours (Geiping and Goldstein, 2022). Our work builds on these trends.

In this study, we introduce our optimized MosaicBERT architecture and show that certain architectural choices for BERT-style encoders lead to accuracy vs. time Pareto improvements in pretraining. We do this by empirically comparing MosaicBERT with an optimal BERT baseline that does *not* incorporate our architectural changes but *does* have non-architectural optimizations such as fast data streaming

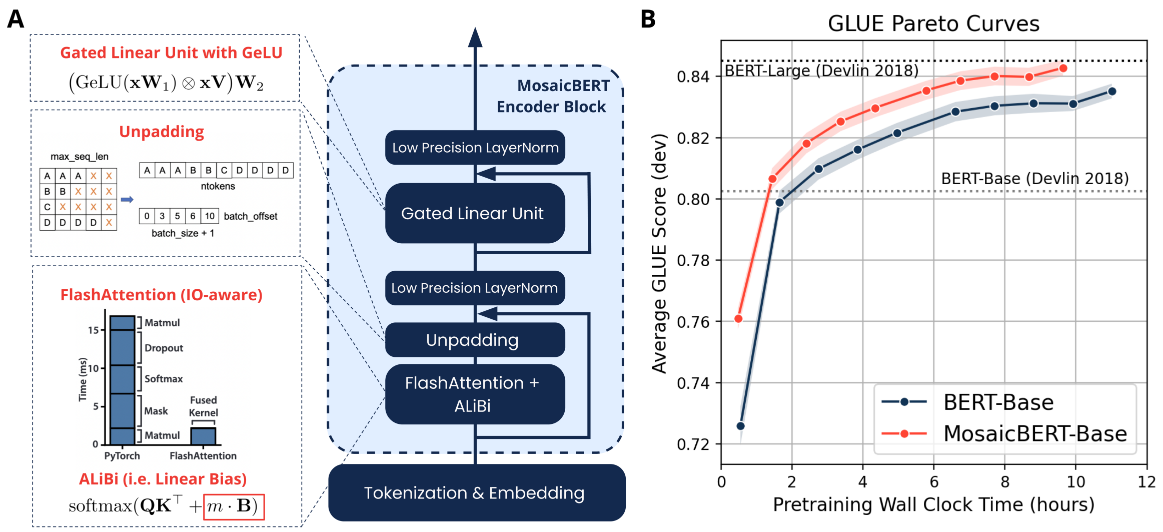

Figure 1. (A) Schematic of MosaicBERT architecture (B) Pareto curves of average GLUE (dev) scores for MosaicBERT-Base and the standard BERT-Base. Error bars indicate 95% confidence interval over n=5 pretraining seeds. All training was on $8\times$A100-80GB GPUs. FlashAttention schematic adapted from (Dao et al., 2022), and unpadding schematic adapted from (Zeng et al., 2022)).

and optimal floating point precision. We then evaluate on the classic GLUE benchmark (Wang et al., 2018).

The contributions of this work are as follows: (1) We implement a new BERT-style encoder architecture that optimizes pretraining speed and accuracy. This architecture combines FlashAttention (Dao et al., 2022), ALiBi (Press et al., 2021), Gated Linear Units (Dauphin et al., 2017; Shazeer, 2020), a dynamic unpadding module (Zeng et al., 2022), and low precision LayerNorm. (2) We show that MosaicBERT base achieves the downstream average GLUE (dev) score of 79.6 in 1.13 hours on $8\times$A100 80 GB GPUs at a cost of roughly \$20 on a standard cloud provider. (3) We characterize the accuracy vs. pretraining time Pareto frontier for MosaicBERT Base and Large, and empirically show that the performance of MosaicBERT Base and Large is Pareto optimal relative to BERT Base and Large. (4) We characterize the relative throughput properties of each of MosaicBERT's architecture and design choices via ablations. (5) We characterize the tradeoff between model size and training duration, and show that BERT Large performance only surpasses BERT Base performance after extensive training. We open-source our model weights, benchmarking data, and code.

2. Methods

In order to build MosaicBERT, we incorporated architectural choices from the recent transformer literature. These include FlashAttention (Dao et al., 2022), ALiBi (Press et al., 2021), training with dynamic unpadding (Zeng et al., 2022), low precision LayerNorm, and Gated Linear Units (Dauphin et al., 2017; Shazeer, 2020). Before describing these modifications in detail, we first review the classic BERT architecture and how we chose a strong baseline.

Since our goal here is to show relative improvements in training time and final accuracy, we do not attempt to beat state of the art models for finetuning benchmarks such as GLUE (Wang et al., 2018). These SOTA models are often trained for much longer (e.g. (Liu et al., 2019)) and are larger than the models we explore in this study (Clark et al., 2020; He et al., 2021)

Choosing a Strong BERT Baseline The basic transformer block used in BERT models consists of (1) the attention mechanism and (2) the feed forward layers. This block is then repeated depending on the model size; BERT-Base has 12 repeated transformer blocks, while BERT-Large has 24.

For our baseline BERT-Base, we used the exact architecture of BERT from (Devlin et al., 2018);[2] this includes a hidden size of 768, an intermediate size of 3072, 12 attention heads and 12 hidden layers, as well as the GeLU activation function, and learned positional embeddings. For our baseline BERT-Large, we used the exact architecture of BERT-Large from (Devlin et al., 2018), which has a hidden size of 1024, an intermediate size of 4096, 16 attention heads, and 24 hidden layers.

While MosaicBERT Base (i.e. 12 hidden layers) and Large (i.e. 24 hidden layers) stay true to this general structure, we introduce modifications that affect both the attention

[2]The Hugging Face `bert-base-uncased` model has been downloaded 62 million times, more than any other model on the Hugging Face hub.

mechanism and the feedforward layers.

2.1. MosaicBERT Architecture Modifications for Fast Pretraining

FlashAttention: The recently proposed FlashAttention layer reduces the number of read/write operations between the GPU HBM (high bandwidth memory, i.e. long-term memory) and the GPU SRAM (i.e. short-term memory) (Dao et al., 2022). We modified the FlashAttention module built by Hazy Research with OpenAI's triton library in order to flexibly incorporate ALiBi.[3]

Attention with Linear Biases (ALiBi): In most BERT models, the positions of tokens in a sequence are encoded using learned position embeddings, which are added to the learned token embeddings at the start of the forward pass. ALiBi eliminates position embeddings and instead encodes position information directly through the attention operation (Press et al., 2021). It adds a negative bias to the attention score between each token pair, which grows linearly with the relative distance between the tokens. Intuitively, this biases attention to nearby tokens and allows for extrapolation to context lengths longer than those used for training (Press et al., 2021). See Appendix for more details.

Gated Linear Units (GLU): We used Gated Linear Units for the feedforward sublayer of a transformer. GLUs were first proposed in 2016 (Dauphin et al., 2017), and incorporate an extra learnable matrix that "gates" the outputs of the feedforward layer (Figure 1A). More recent work has shown that GLUs can improve performance quality in transformers (Shazeer, 2020; Narang et al., 2021). We used the GeLU (Gaussian-error Linear Unit)[4] activation function with GLU, which is sometimes referred to as GeGLU. The module can be described by the following equation: $\big(\mathrm{GeLU}(\mathbf{x}\mathbf{W}_1) \otimes \mathbf{x}\mathbf{V}\big)\mathbf{W}_2$, where $\mathbf{x}$ is the input to the feedfoward layer and the matrix $\mathbf{V}$ gates the output of the GeLU activation function. The extra gating matrix in a GLU model potentially adds additional parameters to a model; we chose to augment our MosaicBERT-Base model with additional parameters due to GLU modules, as it leads to a Pareto improvement across all timescales. While BERT-Base has 110 million parameters, MosaicBERT-Base has 137 million parameters. Note that MosaicBERT-Base reaches higher accuracy faster than BERT-Base *despite having more parameters* (Figure 1B). Similarly, BERT-Large has 340 million parameters, and MosaicBERT-Large has 430 million parameters (see Appendix D).

[3]github.com/HazyResearch/flash-attention. Note that while this research was being completed, PyTorch 2.0 was released with support for FlashAttention. However, FlashAttention in PyTorch 2.0 does not currently support ALiBi integration.

[4]GeLU is a fully differentiable approximation to ReLU, and was used in the original BERT study (Devlin et al., 2018).

Low Precision LayerNorm: LayerNorm is a bandwidth-bound operation, which means that its speed depends on how quickly data can be loaded from memory to the compute units for element-wise operations. Typically the LayerNorm operation is set to `float32` precision, which requires 4-bytes per-element. In MosaicBERT, we modify LayerNorm modules to run in `bfloat16` precision instead of `float32`. This reduces the amount of data that needs to be loaded from memory, as only 2-bytes are required per-element. PyTorch's automatic-mixed-precision package does not, by default, run LayerNorm in lower precision because this can lead to numerical instabilities for certain models. However, our experimental results show that MosaicBERT does not experience any numerical instabilities with `bfloat16` precision LayerNorm.

Unpadding: Standard NLP practice is to combine text samples of different lengths into a batch, and pad the sequences with empty tokens so that all sequence lengths are the same (Figure 1A). During training, however, this leads to many wasted operations on the padding tokens. In MosaicBERT, we take a different approach and instead concatenate all the examples from a minibatch into a single sequence of batch size 1. Results from NVIDIA and others have shown that this approach leads to speed improvements during training, since operations are not performed on padding tokens (Zeng et al., 2022).

MLM Masking Ratio and Dropout We used the standard Masked Language Modeling (MLM) pretraining objective. While the original BERT paper also included a Next Sentence Prediction (NSP) task in the pretraining objective, subsequent papers have shown this to be unnecessary (Liu et al., 2019; Izsak et al., 2021). For our BERT baselines, we used the standard 15% masking ratio. However, we found that a 30% masking ratio led to slight accuracy improvements in both pretraining MLM and downstream GLUE performance. We therefore included this simple change as part of our MosaicBERT training recipe. Recent studies have also found that this can lead to downstream improvements (Wettig et al., 2022; Ankner et al., 2023).

For the baseline BERT, we applied the standard 0.1 dropout to both the attention and feedforward layers of the transformer block. For MosaicBERT, however, we applied 0.1 dropout to the feedforward layers but did not apply dropout to the FlashAttention module, as this was not possible with the OpenAI triton implementation.[5]

3. Results

In our first set of experiments, we pretrained BERT-Base and MosaicBERT-Base for 70,000 steps of batch size 4096, which roughly corresponds to 78% of English C4. We

[5]https://github.com/openai/triton

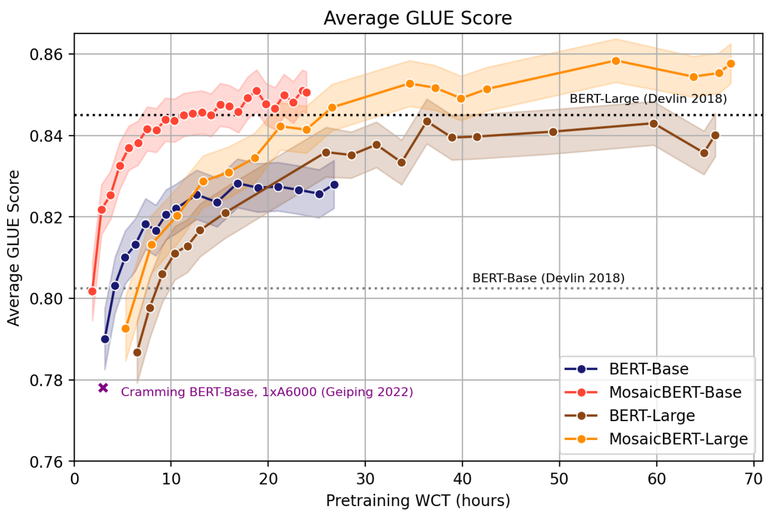

Figure 2. Average GLUE (dev) score Pareto curves for MosaicBERT-Base and Large trained for 2 epochs of C4 (178,000 steps with batch size 4096). MosaicBERT-Base and Large are Pareto optimal relative to BERT-Base and Large. All pretraining is done on 8×A100 80GB devices (n=2-3 pretraining seeds). MosaicBERT-Base took much less time to train than MosaicBERT-Large.

ran experiments with $n = 5$ pretraining seeds for each model class. We then finetuned these models on the GLUE benchmark suite using identical finetuning parameters for all models and experiments (see Appendix).

MosaicBERT-Base Achieves 79.6 Average GLUE (dev) Score in 1.13 Hours MosaicBERT-Base achieves the downstream average GLUE (dev) score of 79.6 in 1.13 hours on 8×A100 80 GB GPUs at a cost of roughly \$20 on a standard cloud provider. More details on cost estimates are included in the Appendix. The baseline BERT-Base reached an average GLUE (dev) score of 83.2% in 11.5 hours (A100-80GB), while MosaicBERT reached the same accuracy in roughly 4.6 hours on the same hardware, which is roughly a 2.38× speedup (Table S1 and Figure 1B).

MosaicBERT-Base is Pareto Optimal As can be seen Figure 1B, MosaicBERT-Base consistently achieves higher average GLUE accuracy more quickly than the standard BERT-Base across all training durations. The performance of MosaicBERT on individual GLUE finetuning tasks can be seen in Figure S1. MosaicBERT-Base outperforms BERT-Base in four out of eight GLUE tasks across pretraining durations.

MosaicBERT-Large is Pareto Optimal While BERT-Base is one of the most popular BERT models, BERT-Large comes in a close second. All of our model development was done on MosaicBERT-Base; we were therefore curious whether our architecture and pretraining choices also generalized to a larger model.

In a second set of experiments, we pretrained MosaicBERT-Base and Large as well as BERT-Base and Large for two epochs of the C4 dataset. The training duration is 178,000 steps with batch size 4096, and is more than twice as long as the duration of the models in Figures 1 and S1.

BERT-Large has 24 repeated transformer layers (BERT-Base has 12), and as a result consumes much more memory and takes longer to train. We found that MosaicBERT-Large reached an average GLUE score of 83.2 in 15.85 hours, while BERT-Large took 23.35 hours (Figure 2). MosaicBERT-Large therefore had a 1.47× speedup over BERT-Large in this training regime.

While MosaicBERT-Base is optimal for a constrained budget, the MosaicBERT-Large average GLUE score eventually surpasses MosaicBERT-Base after 25 hours of training on a 8×A100-80GB node, and reaches an average score of 85.5 in roughly 50 hours. In our experiments, the MosaicBERT-Large architecture and pretraining was the same as MosaicBERT-Base, outside of the number of attention heads, number of hidden layers, and intermediate size of the feedforward units.

A striking result from Figures 2 and S2 is that MosaicBERT-Base is Pareto optimal relative to BERT-large in this training regime, and is Pareto optimal to MosaicBERT-Large during the first half of training. MosaicBERT-Base takes only 25 hours to complete 2 epochs, while MosaicBERT-Large takes close to 70. A potential takeaway from this is that MosaicBERT-Large only surpasses Base performance in the large data regime. For certain tasks such as QQP, SST-2, and MRPC MosaicBERT-Base achieves a maximum accuracy on par with the maximum accuracy of MosaicBERT-Large, for far fewer pretraining hours. When building encoders for specific domains and tasks, bigger is not always better.

References

Z. Ankner, N. Saphra, D. Blalock, J. Frankle, and M. L. Leavitt. Dynamic masking rate schedules for mlm pretraining. *arXiv preprint arXiv:2305.15096*, 2023.

L. Bentivogli, P. Clark, I. Dagan, and D. Giampiccolo. The fifth pascal recognizing textual entailment challenge. In *TAC*. Citeseer, 2009.

C. Blakeney, J. Z. Forde, J. Frankle, Z. Zong, and M. L. Leavitt. Reduce, reuse, recycle: Improving training efficiency with distillation. *arXiv preprint arXiv:2211.00683*, 2022.

D. Cer, M. Diab, E. Agirre, I. Lopez-Gazpio, and L. Specia. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*, 2017.

A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

I. Dagan, O. Glickman, and B. Magnini. The pascal recognising textual entailment challenge. In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment: First PASCAL Machine Learning Challenges Workshop, MLCW 2005, Southampton, UK, April 11-13, 2005, Revised Selected Papers*, pages 177–190. Springer, 2006.

T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR, 2017.

M. Dehghani, J. Djolonga, B. Mustafa, P. Padlewski, J. Heek, J. Gilmer, A. Steiner, M. Caron, R. Geirhos, I. Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. *arXiv preprint arXiv:2302.05442*, 2023.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

B. Dolan and C. Brockett. Automatically constructing a corpus of sentential paraphrases. In *Third International Workshop on Paraphrasing (IWP2005)*, 2005.

J. Geiping and T. Goldstein. Cramming: Training a language model on a single gpu in one day. *arXiv preprint arXiv:2212.14034*, 2022.

D. Giampiccolo, B. Magnini, I. Dagan, and W. B. Dolan. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9, 2007.

P. He, J. Gao, and W. Chen. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*, 2021.

A. Henry, P. R. Dachapally, S. Pawar, and Y. Chen. Query-key normalization for transformers. *arXiv preprint arXiv:2010.04245*, 2020.

S. Iyer, N. Dandekar, and K. Csernai. First quora dataset release: Question pairs, 2017. URL https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs.

G. Izacard, M. Caron, L. Hosseini, S. Riedel, P. Bojanowski, A. Joulin, and E. Grave. Unsupervised dense information retrieval with contrastive learning. 2022.

P. Izsak, M. Berchansky, and O. Levy. How to train bert with an academic budget. *arXiv preprint arXiv:2104.07705*, 2021.

V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020.

V. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro. Reducing activation recomputation in large transformer models. *arXiv preprint arXiv:2205.05198*, 2022.

H. Levesque, E. Davis, and L. Morgenstern. The winograd schema challenge. In *Thirteenth international conference on the principles of knowledge representation and reasoning*, 2012.

P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

H. Li, P. Keung, D. Cheng, J. Kasai, and N. A. Smith. Narrowbert: Accelerating masked language model pretraining and inference. *arXiv preprint arXiv:2301.04761*, 2023.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

S. Narang, H. W. Chung, Y. Tay, W. Fedus, T. Fevry, M. Matena, K. Malkan, N. Fiedel, N. Shazeer, Z. Lan, et al. Do transformer modifications transfer across implementations and applications? *arXiv preprint arXiv:2102.11972*, 2021.

O. Press, N. A. Smith, and M. Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

N. Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

W. Shi, S. Min, M. Yasunaga, M. Seo, R. James, M. Lewis, L. Zettlemoyer, and W.-t. Yih. Replug: Retrieval-augmented black-box language models. *arXiv preprint arXiv:2301.12652*, 2023.

M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

A. Warstadt, A. Singh, and S. R. Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019.

A. Wettig, T. Gao, Z. Zhong, and D. Chen. Should you mask 15% in masked language modeling? *arXiv preprint arXiv:2202.08005*, 2022.

A. Williams, N. Nangia, and S. R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

J. Zeng, M. Li, Z. Wu, J. Liu, Y. Liu, D. Yu, and Y. Ma. Boosting distributed training performance of the unpadded bert model. *arXiv preprint arXiv:2208.08124*, 2022.

B. Zhang and R. Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.

A. Accuracy vs. Pretraining Time Pareto Curves

It can often be difficult to understand the effects of model architecture modifications and training hyperparameter choices by simply reporting a few numbers in a table, as is traditionally done in ML papers. A few numbers in a table can obscure differences in the training data, objective function, batch size, training duration, learning rate schedule and many other details. Since our main goal in this study is to show how combinations of architecture choices lead to improvements in both *accuracy* and *training time*, we make the choice to plot accuracy vs. training time Pareto curves for all models. Certain architecture changes might lead to an increase in throughput, but a decrease in accuracy (e.g. changes to the floating point precision); other changes might lead to an increase in accuracy but take a much longer time to converge (e.g. increasing the model size). Accuracy vs. training time Pareto curves allow us to adequately asses all these changes.

B. Further ALiBi Details

Following the notation in (Press et al., 2021), the attention block computes the attention scores between the *i*th query $q_i \in \mathbb{R}^d$ and keys $\mathbf{K} \in \mathbb{R}^{L\times d}$ where d is the head dimension and L is the sequence length. ALiBi adds a fixed bias with m as a head-specific slope controlling how the bias grows with absolute distance between tokens, yielding attention weights as

$$\texttt{softmax}\big(q_i\mathbf{K}^\top - m\cdot\texttt{abs}([i-1, i-2, ..., i-L])\big). \quad (1)$$

The slopes m follow a geometric sequence such that for n heads, each head has a ratio of $2^{-8/n}$ (Press et al., 2021). During finetuning or inference, the static bias can be increased to accommodated longer sequence lengths. For example, a model pretrained using ALiBi with a maximum sequence length of 128 tokens can then extrapolate to a task with 256 tokens with little to no decrease in zero-shot performance. Since pretraining a model with a maximum sequence length of 128 has much higher throughput than pretraining a model with a sequence length of 256, ALiBi can be considered an indirect speedup method.

C. Pretraining Optimizations for Both MosaicBERT and the BERT baseline

Data Pretraining data is an important factor when comparing BERT models; while the original BERT study trained on English Wikipedia and the Books Corpus (Zhu et al., 2015), subsequent models such as RoBERTa trained on much larger datasets (e.g. (Liu et al., 2019) trained on 160 GBs of text while (Devlin et al., 2018) only trained on 16GB of text). Here we chose to train all models on the more modern Colossal Cleaned Common Crawl (C4) corpus (Raffel et al., 2020). For all experiments, we used a maximum sequence length of 128 tokens; a larger maximum sequence length naturally leads to a decrease in throughput (see Figure S5).

Streaming Dataset As part of our efficiency pipeline, we converted the C4 dataset to the `StreamingDataset` format[6] and used this for both MosaicBERT-Base and the baseline BERT-Base. This ensured that our wall clock time measurements were not hindered by data streaming issues.

Bfloat16 Precision We use `bfloat16` mixed precision training for all the models. `bfloat16` is a custom 16-bit floating point format for machine learning that has one sign bit, eight exponent bits, and seven mantissa bits, and has the dynamic range of `float32`. For mixed precision training, a matrix multiplication layer uses `bfloat16` for the multiplication and 32-bit IEEE floating point (`float32`) for gradient accumulation. We found this to be more stable than using `float16` mixed precision.

Vocab Size as a Multiple of 64 We increased the vocab size to be a multiple of 64 (i.e. from 30,522 to 30,528). This small constraint is something established in the MEGATRON work by Shoeybi et al. (2019), and leads to a nontrivial throughput speedup. Note that in the original BERT study, the vocabulary size was 32,768 (i.e. 2^{16}) (Devlin et al., 2018). Across all experiments, we use the standard BERT-Base and Large tokenizers.

Hyperparameters For all models, we use a global batch size of 4096, and microbatch size of 128. We set the maximum sequence length during pretraining to 128, and we used the standard embedding dimension of 768. These hyperparameters were the same for MosaicBERT-Base and the baseline BERT-Base. Full hyperparameter details are included in the github.com/mosaicml/examples repository.

The space of NLP benchmarks and tasks has exploded in recent years; we include more information on the individual tasks in the classic GLUE benchmark later in the Appendix. MNLI, QNLI and QQP are the largest datasets in the GLUE benchmark, with 100k-400k training examples, and MosaicBERT-Base is strictly Pareto-optimal for these tasks relative to BERT-Base (Figure S1). We interpret this to mean that the architectural changes and training recipe we chose resulted in an optimized, efficient BERT-Base model.

The quality of both models on smaller datasets (3k-67k training samples) is much more variable, as shown by the large error bars (standard deviation across n=5 pretraining seeds) for SST-2, MRPC and STSB. Regardless of this variation, MosaicBERT-Base performs equivalently to BERT-Base on these tasks across training duration.

[6]github.com/mosaicml/streaming. It is a drop in replacement for PyTorch's `IterableDataset` that allows for fast streaming from cloud storage.

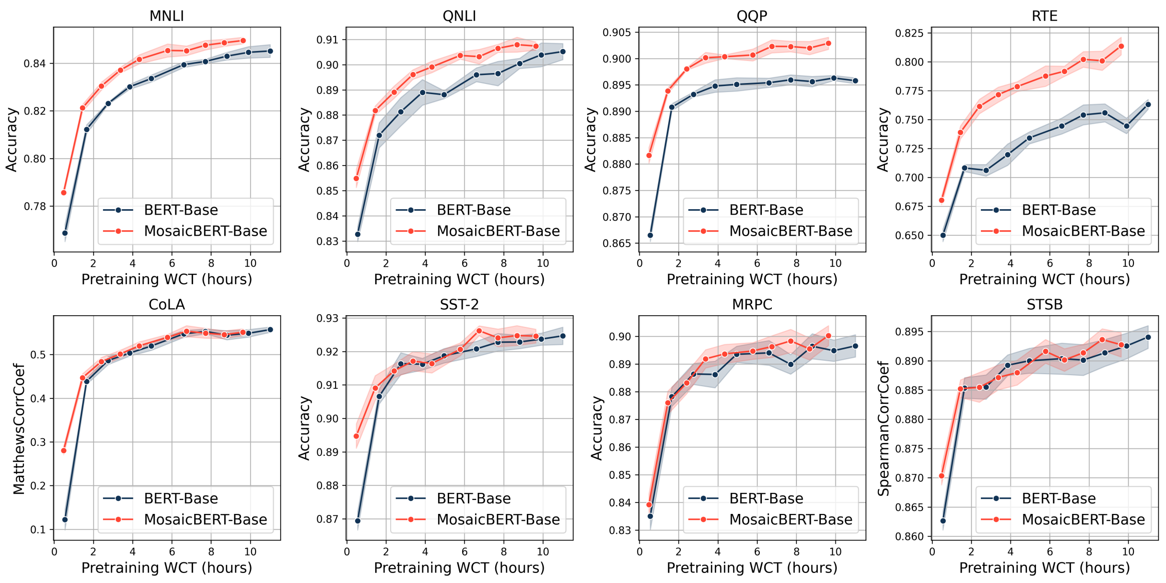

Figure S1. Performance on individual GLUE (dev) finetuning tasks. Our MosaicBERT-Base consistently outperforms BERT-Base on MNLI-m, QNLI, QQP and RTE, and has comparable performance on CoLA, SST-2, MRPC and STSB. Wall clock time is for 8×A100-80GB GPUs, and does not include finetuning time. Error bars are plotted with 95% confidence interval across $n = 5$ pretraining seeds, and all models are trained for 70,000 steps with batch size 4096.

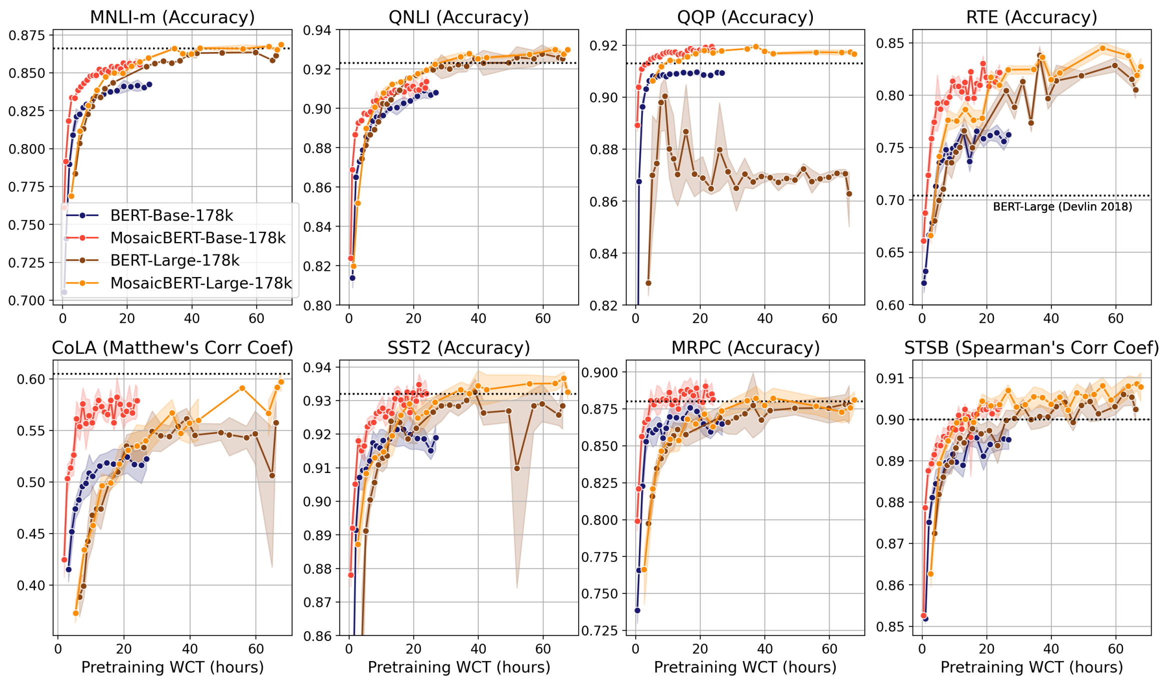

Figure S2. MosaicBERT-Base and Large accuracy (dev) vs. pretraining speed Pareto curves for individual GLUE benchmarks. All models were trained on 2 epochs of C4 (178,000 steps with batch size 4096). Error bars represent 95% confidence interval for n=2-3 pretraining seeds. Dashed black line represents BERT-Large accuracy on the GLUE (dev) data (Devlin et al., 2018; Liu et al., 2019).

Model Architecture	Average GLUE (dev)	Training (hours)	Hardware	Pretraining Corpus	# Params
MosaicBERT-Base	79.6	1.13	8×A100-80	C4	137M
MosaicBERT-Base	82.2	2.81	8×A100-80	C4	137M
MosaicBERT-Base	83.2	4.6	8×A100-80	C4	137M
MosaicBERT-Base	83.4	5.27	8×A100-80	C4	137M
BERT-Base (benchmark)	83.2	11.5	8×A100-80	C4	110M
BERT-Base (Devlin et al., 2018)	79.6	96	16×TPU	Wiki+Books	110M
BERT-Base (Geiping and Goldstein, 2022)	80.3	24	1 RTX A6000	Wiki+Books	110M
CrammingBERT-Base (Geiping and Goldstein, 2022)	77.8	24	1 RTX A6000	Wiki+Books	110M
BERT-Large (Devlin et al., 2018)	84.1	96	64×TPU	Wiki+Books	340M

Table S1. Average GLUE (dev) score across various efficient BERT implementations. Average includes all 8 GLUE tasks.

D. Ablation Throughput Experiments

How do each of the architecture modifications affect throughput during training? We ran a series of experiments looking at the individual and combinatorial effects of low precision LayerNorm, ALiBi and GLU on the baseline BERT-Base. As can be seen from Figure S3A, ALiBi has a marginal effect on the throughput as measured by samples per second, while GLU leads to a decrease in throughput. Low precision LayerNorm alone causes an *increase* in throughput, due to the reduced floating point precision. When low precision LayerNorm, ALiBi and GLU are combined together on top of the baseline BERT, the throughput gains of low precision LayerNorm are cancelled out by the slowdown due to GLU.

In Figure S3B, we show the throughput of the "complete" MosaicBERT-Base with differently-sized GLU matrices of intermediate size 2048 and 3072. While MosaicBERT-Base with GLU-2048 has the highest throughput, we found that the added expressivity of GLU-3072 due to the increase in parameters compensated for the relative decrease in throughput, and therefore used this configuration for the modeling in Figures 1-S2. Therefore, while BERT-Base has 110 million parameters, MosaicBERT-Base has 137 million parameters due to the added GLU parameters. Similarly, while BERT-Large has 340 million parameters, MosaicBERT-Large has 430 million parameters.

Since Dao et al. (2022) showed that FlashAttention leads to an increase in throughput without any decrease in accuracy, and Zeng et al. (2022) showed that removing padding leads to an increase in throughput, we do not include those ablation experiments here.

E. Related Work

Various studies rightly focus on a single method or modification that improves throughput, such as FlashAttention (Dao et al., 2022) and unpadding (Zeng et al., 2022). In this study, we incorporate many of these techniques to investigate whether they combine advantageously.

RoBERTa ("Robustly optimized BERT approach") is the most influential work in this regard (Liu et al., 2019). In this study, they kept the exact BERT architecture but changed the training recipe by removing the next sentence prediction objective, training for longer on much larger datasets, and changing the batch size, among other things. They showed that the original BERT was significantly undertrained - while the original BERT trained on 16GB worth of text, the top accuracy RoBERTa (Large) was trained on 160GB of data for 500,000 steps with batch size 8192. Many of the training choices in RoBERTa have become standard practice; our training recipe therefore more closely resembles RoBERTa than the original BERT.

Improvements in transformer architecture and GPU hardware have caused the cost of pretraining BERT models to decline precipitously. The very recent paper "Cramming: Training a Language Model on a Single GPU in One Day" (Geiping and Goldstein, 2022) exemplifies this trend. The goal of this study was to train the best BERT in 24 hours on a single GPU. Similar to us, they tweaked the BERT architecture to incorporate FlashAttention and Gated Linear Units (but without increasing the dimensionality of the hidden block). Unlike MosaicBERT, they used scaled sinusoidal positional embeddings (Vaswani et al., 2017) as well as Pre-LayerNorm (applying LayerNorm before the attention and feedforward layers) and did not change the pretraining masking rate. With this setup, they were able to train their modified BERT-Base to an average GLUE score of 78.6 in 24 hours on a single A6000 GPU (i.e. 24 GPU hours). Our study is similar in spirit, but asks what is the fastest architecture for pretraining, and expands on this in greater detail by showing that MosaicBERT is Pareto optimal relative to BERT.

While we believe that our training optimization choices for MosaicBERT go a long way to improve BERT training efficiency, there are still exciting optimizations to pursue.

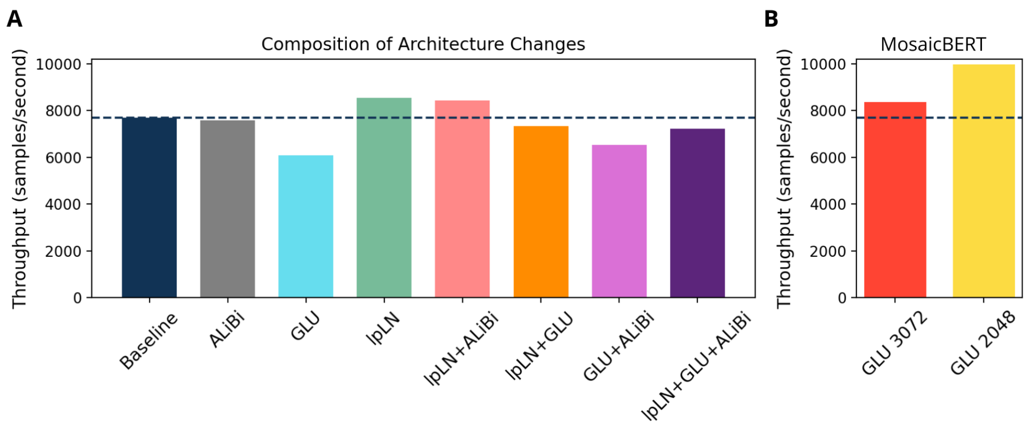

Figure S3. Effect of architectural modification on throughput (samples/second) for BERT-Base. (A) ALiBi, GLU and low precision LayerNorm (lpLN) each affect throughput. In combination, ALiBi, GLU and low precision LayerNorm have lower throughput than the baseline BERT-Base. (B) Throughput of the "complete" MosaicBERT-Base with different intermediate sizes for GLU.

"NarrowBERT" (Li et al., 2023) suggested a clever change to the way encoder blocks are stacked so that computation is not wasted on unmasked tokens. Another recent study showed that dynamically masking the masking ratio during pretraining leads to downstream accuracy gains in BERT (Ankner et al., 2023). There are likely further modifications that could lead to an increase in accuracy with no effect on throughput, such as replacing LayerNorm with RMSNorm (Zhang and Sennrich, 2019) and GeGLU with SwiGLU (Shazeer, 2020; Narang et al., 2021).

Approaches such as knowledge distillation (Blakeney et al., 2022) might additionally push the Pareto frontier of BERT models during pretraining and finetuning. As the field is learning how to optimize stable model training at the billion parameter scale (Chowdhery et al., 2022; Dehghani et al., 2023), we expect some of these innovations to cycle back to smaller models such as BERT-Base. For example, it has been hypothesized that incorporating more LayerNorm modules at the QK matrix output (i.e. QK-LayerNorm (Henry et al., 2020)) can lead to improved stability during training; combining this with a more aggressive learning rate schedule could lead to faster convergence.

F. GLUE Benchmark Details

The GLUE benchmark consists of 8 (originally 9) tasks (Wang et al., 2018). Since there has been a Cambrian explosion of benchmarks since the halcyon days of GLUE, we elaborate on the individual GLUE benchmarks for reference:

F.1. Large Finetuning Datasets

MNLI (Multi-Genre Natural Language Inference) [392,702 train, 19,643 test] is a large crowd-sourced entailment classification task (Williams et al., 2017). The model is given two sentences and has to predict whether the second sentence is entailed by, contradicts, or is neutral with respect to the first one. For example:

- Premise: "Buffet and a la carte available."
- Hypothesis: "It has a buffet."
- Label: 0 (entailment)

QNLI [104,743 train, 5,463 test] this Stanford Question Answering dataset consists of question-paragraph pairs drawn from Wikipedia (Rajpurkar et al., 2016).

QQP (Quora Question Pairs 2) [363,846 train, 390,965 test]. The task is to determine whether two sentences are semantically equivalent (Iyer et al., 2017).

F.2. Small Finetuning Datasets

RTE (Recognizing Textual Entailment) [2,490 train, 3,000 test] Given two sentences, the model has to predict whether the second sentence is or is not entailed by the first sentence (Dagan et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009). Note that in our work we use a checkpoint from the MNLI finetuning to finetune on RTE.

CoLA (Corpus of Linguistic Acceptability) [8,551 train, 1,063 test] (Warstadt et al., 2019) is a benchmark with sentences that are either linguistically acceptable or grammatically incorrect. For example:

- "The higher the stakes, the lower his expectations are." Label: 1 (acceptable)
- "Mickey looked up it." Label: 0 (unacceptable)

SST-2 (Stanford Sentiment Treebank) [67,349 train, 1,821 test] consists of sentences from movie reviews. The task is to classify the sentiment as either positive or negative (Socher et al., 2013).

MRPC (Microsoft Research Paraphrase Corpus)[3,668 train, 1,725 test] (Dolan and Brockett, 2005) The dataset consists of sentence pairs extracted from online news sources. The task is to classify whether the sentences in the pair are semantically equivalent.

STSB (Semantic Textual Similarity Benchmark) [5,749 train, 1,379 test] This dataset contains sentence pairs that are given similarity scores from 0 to 5 (Cer et al., 2017).

Note that we excluded finetuning on the 9th GLUE task WNLI (Winograd NLI) (Levesque et al., 2012), as in the original BERT study (it is a very small dataset [634 train, 146 test] with a high number of adversarial examples). Finetuning on RTE, MRPC and STSB starts from a checkpoint already finetuned on MNLI (following the example of (Izsak et al., 2021) and other studies). This is done because all the above tasks deal with sentence pairs, and this staged finetuning leads to consistent empirical improvement.

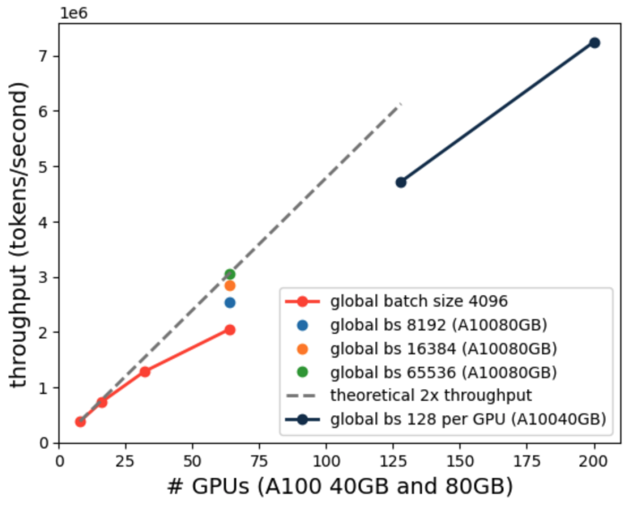

Figure S4. MosaicBERT-Large (430M) multinode throughput scaling

G. Finetuning Hyperparameters

We used the hyperparameters in Table S2 for finetuning all BERT and MosaicBERT models. All finetuning datasets used a max sequence length of 256 tokens. We found that these values worked well across all tasks for BERT-Base, MosaicBERT-Base, and MosaicBERT-Large; BERT-Large however was somewhat under-performant on QQP for some pretraining seeds (Figure S2).

H. MosaicBERT-Large Multinode Throughput Scaling

The experiments in the main section of this paper were all performed on a single node with $8\times$ A100 GPUs. How well do our innovations to the BERT architecture maximize throughput at the multinode scale?

We measured the throughput of MosaicBERT-Large (430M) during training on 8, 16, 32, 64, 128 and 200 GPUs, and plotted the tokens per second for various global batch sizes. Global batch size is an important factor in the throughput measurements; in general, cranking up the batch size increases the GPU utilization and raw throughput. As the number of nodes increases, the global batch size needs to be increased as well in order to maintain high throughput.

If the global batch size is kept constant while increasing the number of nodes, the throughput does not increase linearly. This can be seen in Figure S4; a global batch size of 4096 spread across 64 GPUs using Distributed Data Parallelism (DDP) means that each GPU will only apply matmul operations on matrices with a dimension of 64, which leads to suboptimal throughput. If the global batch size is increased to 65,536 across 64 GPUs, this roughly means that each GPU will apply matmul operations on matrices with a dimension of 1024, leading to higher throughput. However, such a large global batch size might not lead to the best downstream accuracy; this is a question that we were not able to address in this study due to resource and time constraints.

I. MosaicBERT-Base Model FLOPs Utilization (MFU)

Model FLOPs Utilization (MFU) is an estimate of what percentage of the hardware's FLOPs are being used during training. The estimate is based on the measured throughput and the known FLOPs of the computation.

MFU calculates the utilization from the floating point operations required for a single forward/backwards pass of the model, and does not account for the additional compute required for other implementation details such as activation checkpointing. Thus, MFU is independent of implementation and hardware. For more details, see (Korthikanti et al., 2022). All FLOP calculations exclude the operations required for normalization, activation, and residuals.

Following the notation in the PaLM paper (Chowdhery et al., 2022), Model FLOPs Utilization (MFU) is approximated as:

GLUE Task	learning rate	beta	epsilon	weight decay	epochs
MNLI	5e-5	[0.9, 0.98]	1e-6	5e-6	3
QNLI	1e-5	[0.9, 0.98]	1e-6	1e-6	10
QQP	3e-5	[0.9,0.988]	1e-6	3e-6	5
RTE	1e-5	[0.9, 0.98]	1e-6	1e-5	3
CoLA	5e-5	[0.9, 0.98]	1e-6	5e-6	10
SST-2	3e-5	[0.9,0.988]	1e-6	3e-6	3
MRPC	8e-5	[0.9, 0.98]	1e-6	8e-6	10

Table S2. Finetuning hyperparameters for BERT and MosaicBERT across Base and Large.

$$\text{MFU} = \frac{(6 \cdot n_{parameters} \cdot T_{observed})}{n_{gpus} \cdot T_{theoretical}} \tag{2}$$

where $T_{observed}$ is the observed throughput and $T_{theoretical}$ is the theoretical peak throughput.

In the numerator, the number of learnable parameters in the model is multiplied by a factor of 6 to estimate the matmul FLOPs per token seen ($2\times$ for the forward pass and $4\times$ for the backward pass). This is then multiplied by the number of tokens seen per second. As a first-order approximation, we exclude the extra FLOPs per token due to dense self-attention.

In the denominator, the theoretical peak throughput is provided in the GPU hardware specs. For A100 GPUs using `bfloat16`, this theoretical peak throughput is 312 teraFLOPs.

J. GPU Pricing

As of this writing, A100 GPU pricing ranges from \$4.10 (40 GB) for on demand cloud compute on AWS, to \$2.46 (40 GB) / \$5.00 (80 GB) per GPU on GCP to \$1.10 (40 GB) / \$1.50 (80 GB) per GPU using Lambda labs. At an intermediate price of \$2.50 an hour per A100 80 GB GPU, training to 79.6 GLUE average score takes 1.13 hours and costs roughly \$22.60.[7] Some example costs are calculated in Table S4.

K. Throughput as a Function of Sequence Length

In Figure S5, we plot the pretraining throughput of MosaicBERT-Base with various context windows. As the sequence length doubles, the pretraining throughput halves. We note that for all of the pretraining in the main text, we use a maximum sequence length of 128.

[7]See for example "Cloud GPU instances with the largest VRAM 2022" (https://medium.com/@aleixlopez/cloud-gpu-instances-to-solve-out-of-memory-error-2022-d5012883a272?)

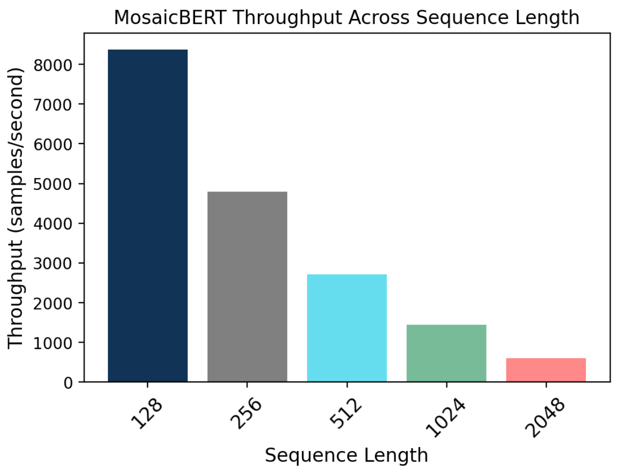

Figure S5. Throughput for Various Sequence Lengths

L. Gated Linear Units (GLU) Optimizations

GLU adds elementwise multiplication of two linear projections and it leads to qualitative improvements over the standard Transformer block. There are multiple ways to implement GLUs and we experimented with two implementations. Figure S6 shows standard feedforward transformer block (A) and two implementations of GLUs (B-C). "Fused GLU" in (C) fuses the two matrix multiplications into one and is expected to perform better in some domains.

Figure S7 shows the performance impact of the two GLU over standard feedforward transformer block (which would be 0% slowdown) for a single GPU. This figure only shows the performance of the forward pass, and the backward pass is expected to behave similarly. We can draw two conclusions from this chart: 1) For smaller batch sizes, both GLU implementations add significant overhead over the standard block 2) For batch sizes < 128, Fused GLU implementation is better than regular GLU implementation and beyond 128 it's slightly worse. The implementation used in the main text is the "Fused GLU" implementation (C) with batch size global 4096. Since the profiling in Figure S7 is per GPU, this is in the regime of $4096/8 = 512$.

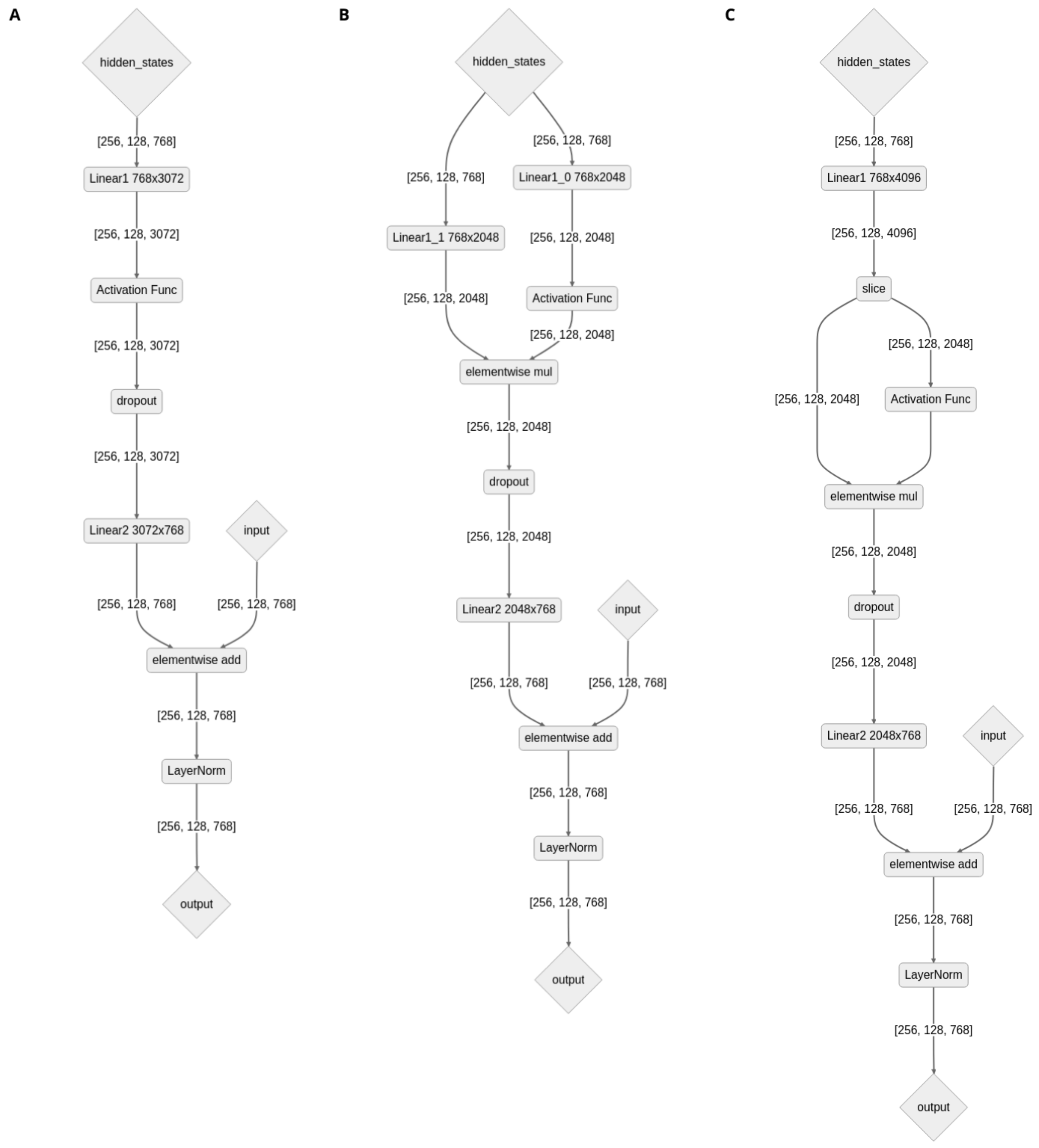

Figure S6. Standard FeedForward Transformer Block and Gated Linear Unit Modifications. Each edge shows the tensor dimension with a batch size of 256, sequence length of 128 and a hidden dimension of 768. (A): A standard transformer feedforward block. (B): Naive implementation of a Gated Linear Unit. The number of parameters in this are the same as in (A). (C): Fused implementation of a Gated Linear Unit where the two matrix multiplications (`Linear1_0` and `Linear1_1`) from (B) are fused into one (`Linear1`) with 2× the parameters. Here the output is sliced, and is functionally equivalent to (B).

Model	Throughput (tokens /sec)	MFU	Hardware	Time to Av. GLUE (dev) 79.6	Batch Size	Micro-batch Size
BERT-Base	0.4e6	10.4%	8× A100 80	110.4 minutes (1.84 hours)	4096	512
MosaicBERT-Base	1.1e6	39.97%	8× A100 80	67.8 minutes (1.13 hours)	4096	512
MosaicBERT-Base	0.938e6	30.9%	8× A100 40	76.8 minutes	4096	128
MosaicBERT-Base	1.88e6	31.0%	16× A100 40	38.5 minutes	4096	128
MosaicBERT-Base	3.15e6	25.9%	32× A100 40	23.1 minutes	4096	128
MosaicBERT-Base	4.77e6	19.6%	64× A100 40	15.7 minutes	4096	64

Table S3. Multinode Throughput scaling for MosaicBERT-Base

MosaicBERT-Base Ave. GLUE Score	8×A100 80GB hours	8×A100 80GB cost ($2.50 GPU/hr)	8×A100 40GB hours	8×A100 40GB cost ($2 GPU/hr)
79.6	1.13	$22.60	1.28	$20.00
82.2	2.81	$56.20	3.19	$51.00
83.4	5.27	$105.40	5.99	$95.78

Table S4. MosaicBERT-Base GLUE (dev) scores, time and cost comparison

The main reason for slowness of GLUs over standard block is extra elementwise multiplication in GLU layers. As for why fused implementation is slower, profiling analysis shows that the Linear layer ends up calling different CUDA kernels for matrix-multiplications and their relative performance varies for different sizes. While the MosaicBERT architecture in this work uses the fused GLU implementation, the analysis here indicates that it would be slightly more efficient to use the standard GLU implementation instead.

M. Limitations and Broader Impact

M.1. Limitations

While we trained two different model sizes, we have not pretrained a MosaicBERT model in the >1B parameter range. In this regime, it is possible there will be training stability issues; this is an area of future work.

We also only trained models for 70,000 steps and 178,000 steps of batch size 4096. It is possible that some of the Pareto properties change in the longer regime, although we suspect that this is unlikely.

M.2. Broader Impact

BERT models are highly used for NLP tasks. By open-sourcing this work, we hope that our code and models will be used by the wider research community. We recognize however that models like BERT and MosaicBERT are tools that can be used for nefarious purposes, and that biases inherent in the training data can be reflected in the final model artefacts.

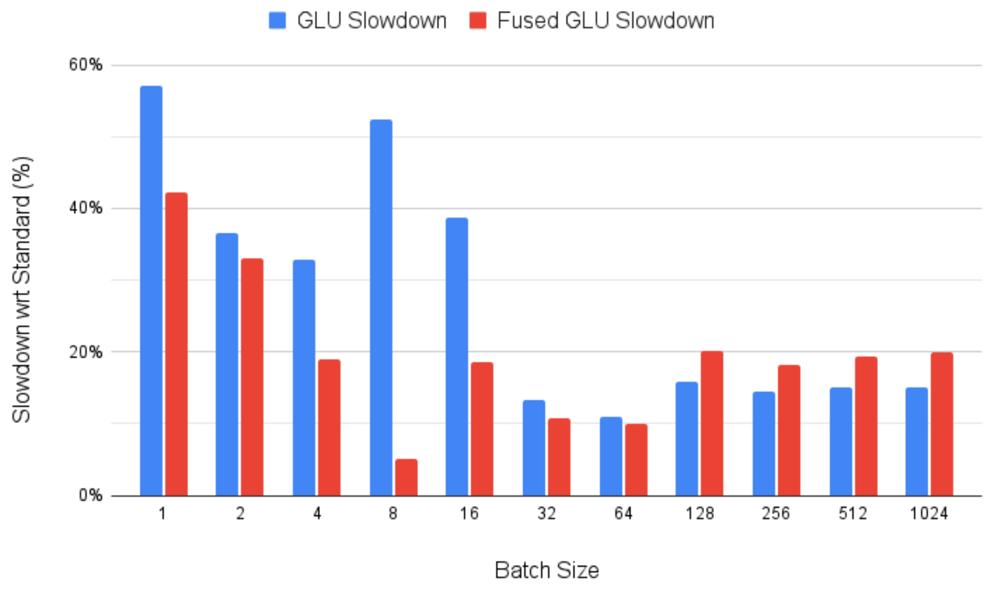

Figure S7. Slowdown of different implementations of the Gated Linear Unit (GLU). This slowdown is with respect to the standard feedforward transformer block. The number of parameters between the standard feedforward transformer block and the two GLU implementations are the same (whereas in the main text, the number of parameters is larger for MosaicBERT corresponding to an intermediate size of 3072 for the GLU matrix; see Appendix D). The profiling here is per GPU on an 8-GPU cluster. For global batch size 4096, the corresponding per-GPU batch size on the x-axis is 512.