Combining Compressions for Multiplicative Size Scaling on Natural Language Tasks

Rajiv Movva[*123], **Jinhao Lei**[*2], **Shayne Longpre**[23], **Ajay Gupta**[2], and **Chris DuBois**[2]

[*]Equal Contribution
[1]Cornell Tech, rm868@cornell.edu
[2]Apple, {jlei2, ajay_gupta2, cdubois}@apple.com
[3]Massachusetts Institute of Technology, slongpre@mit.edu

Abstract

Quantization, knowledge distillation, and magnitude pruning are among the most popular methods for neural network compression in NLP. Independently, these methods reduce model size and can accelerate inference, but their relative benefit and combinatorial interactions have not been rigorously studied. For each of the eight possible subsets of these techniques, we compare accuracy vs. model size tradeoffs across six BERT architecture sizes and eight GLUE tasks. We find that quantization and distillation consistently provide greater benefit than pruning. Surprisingly, except for the pair of pruning and quantization, using multiple methods together rarely yields diminishing returns. Instead, we observe complementary and super-multiplicative reductions to model size. Our work quantitatively demonstrates that combining compression methods can synergistically reduce model size, and that practitioners should prioritize (1) quantization, (2) knowledge distillation, and (3) pruning to maximize accuracy vs. model size tradeoffs.

1 Introduction

As increasingly large models dominate Natural Language Processing (NLP) benchmarks, model compression techniques have grown in popularity (Gupta and Agrawal, 2020; Rogers et al., 2020; Ganesh et al., 2021). For example, quantization (Shen et al., 2020; Zafrir et al., 2019; Jacob et al., 2018) lowers bit precision of network weights to reduce memory usage and accelerate inference (Krashinsky et al., 2020). Knowledge distillation (KD; Hinton et al. (2015)), which trains a student neural network using the logits (or representations) of a teacher network, is used widely to transfer knowledge to smaller models (Sanh et al., 2019; Jiao et al., 2020; Sun et al., 2019, 2020). Pruning identifies weights which can be omitted at test time without significantly degrading performance. Some pruning methods remove individual weights according to magnitudes or other heuristics (Gordon et al., 2020; Chen et al., 2020; Sanh et al., 2020), while others remove structured blocks of weights or entire attention heads (Wang et al., 2020; Hou et al., 2020; Voita et al., 2019; Michel et al., 2019).

Recent work has begun combining these compression methods for improved results. Sanh et al. (2020), Zhang et al. (2020), and Bai et al. (2021) have used knowledge distillation with pruning or low-bit quantization to fine-tune BERT. As practitioners look to combine methods more generally, new research is needed to compare their empirical value and study interactions. This work addresses the questions: (1) Which popular compression methods or combinations of methods are usually most effective? (2) When combining methods, are their benefits complementary or diminishing?

We address these questions by computing accuracy vs. model size tradeoff curves for six pretrained BERT sizes fine-tuned on eight GLUE tasks (Wang et al., 2019b), applying each of eight possible subsets of quantization-aware-training (QAT), knowledge distillation (KD), and magnitude pruning (MP). Our main findings are as follows:

1. When methods are applied independently, QAT yields best accuracy-compression tradeoffs, followed by KD and then MP.
2. Strikingly, we observe no diminishing returns when combining KD with QAT or MP. Instead, KD mitigates the loss in accuracy caused by either method, thereby super-multiplicatively reducing model size.
3. When used together, QAT and MP amplify each other's individual accuracy losses. However, combining all three methods (*i.e.*, also using KD) preserves accuracy, allowing 18x and 11x compression for BERT-LARGE and BASE respectively.

Proceedings of the 29th International Conference on Computational Linguistics, pages 2861–2872
October 12–17, 2022.

2 Methods

In our work, we study three common model compressions: quantization-aware-training, knowledge distillation, and magnitude pruning. We prioritize performant, broadly applicable approaches with accessible implementations, so our findings are most useful to practictioners. Hyperparameters and additional method details are in Appendices A & C.

BERT Architecture Sizes. We test each compression combination across six different BERT architecture sizes, seeing as they may have different compressibilities. These pretrained models are taken from Turc et al. (2019): LARGE (367 million params), BASE (134M), MEDIUM (57M), SMALL (45M), MINI (19M), TINY (8M). Including a range of sizes makes our findings relevant to practitioners or deployment settings without resources for architectures like LARGE or BASE. Additionally, using smaller model sizes is a practical baseline to compare our compression methods against.

Quantization-Aware-Training (QAT). While most neural networks use 32-bit floats for weights and activations, recent work has shown promise for lower precisions. 16-bit floats cause no accuracy loss for most architectures (Das et al., 2018), and Zafrir et al. (2019) show that, with quantization-aware-training (QAT), 8-bit integer (INT8) BERT mostly preserves GLUE accuracy. The INT8 model is nearly 4x smaller, and can achieve 2.4-4.0x inference acceleration with appropriate hardware (Kim et al., 2021a). As lower precisions harm accuracy significantly (Shen et al., 2020), we use an 8-bit BERT with the QAT scheme described by Zafrir et al. (2019), recapped in the Appendix.

Knowledge Distillation (KD). In KD, we fine-tune a small student model by optimizing its weights to mimic the outputs of a teacher model. We use a common, simple variant of KD, emulating Turc et al. (2019): we use a BERT-LARGE fine-tuned for three epochs on the GLUE task as the teacher, and the student is trained to minimize KL-divergence between its predicted probabilities and the teacher's.

To further improve the utility of KD, we adopt Jiao et al. (2020)'s approach of data augmentation (DA) for GLUE training datasets. This technique helps for all tasks, especially the smaller ones (*e.g.* MRPC, RTE). Each example is copied 10, 20, or 30 times (more copies for smaller tasks), and each copy has some of its words replaced with synonyms (*i.e.* words with closest GLoVe embeddings). Many of the copies have altered meanings, but the teacher is able to adapt by making different predictions. Before running all of our experiments, we ran a few trials (on MRPC and QNLI) to confirm that DA helped with distillation but not without, and so we only used DA for the KD experiments.

Magnitude Pruning (MP). Several pruning methods have been used in NLP (Hoefler et al., 2021). We use unstructured weight pruning, which can achieve higher sparsities than structured pruning (Renda et al., 2020), and has comparatively standard implementations.

Magnitude pruning masks the weights with lowest magnitudes to achieve a target sparsity. As in Sanh et al. (2020), we iteratively prune weights at a linear schedule during training after some warmup steps. Sanh et al. (2020) also propose *movement pruning*, in which weights are pruned according to their gradients during fine-tuning. We found that movement pruning performs worse than magnitude at moderate sparsities (40-60%), when accuracy is retained (corroborated by Sanh et al. (2020)). As we target accuracy-preserving pruning, we use magnitude pruning for the experiments in this work. We prune either 40% or 60% of encoder weights only, as pruning embedding weights significantly damages accuracy (Yu et al., 2020).

3 Results

Experiments For six BERT architecture sizes and eight GLUE tasks[1], we tested every possible subset of compression methods: no compression (Baseline), QAT, KD, MP, QAT+KD, QAT+MP, KD+MP, and QAT+KD+MP. For each of 576 experiment settings, we log the max GLUE development set accuracy across twelve hyperparameter configurations and five repetitions of each configuration. In Figure 1, we plot mean GLUE accuracy across all eight tasks on the y-axis against decreasing model size on the x-axis, for each compression combination. The curves without pruning include six points, one for each architecture size from LARGE to TINY. With pruning, there are twice as many points, as each architecture is pruned to either 40% or 60% encoder sparsity. Results split by task are available in Appendix Figure A1.

Individually, QAT and KD are most effective. For all architectures, QAT (blue) reduces model

[1] We excluded CoLA and WNLI to reduce experimental burden and due to issues with WNLI (Wang et al., 2019a).

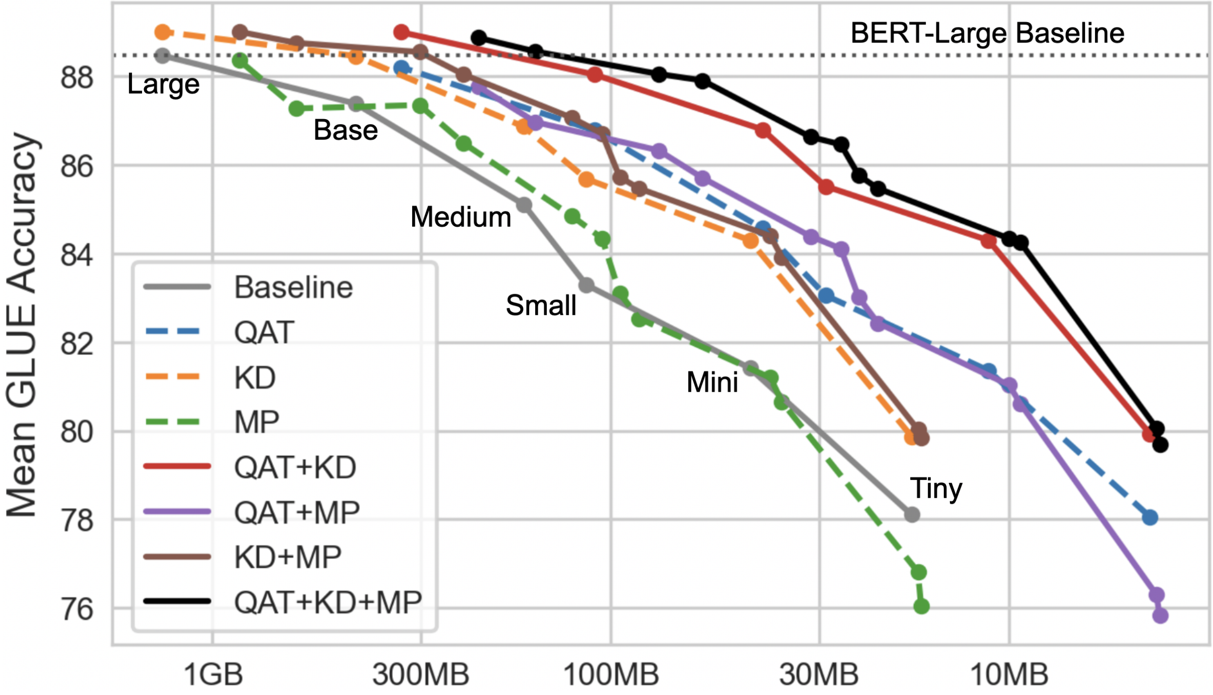

Figure 1: Mean GLUE accuracy vs. **decreasing** model size, with curves plotted for each compression combination. The different points for each curve represent the different BERT architecture sizes, from LARGE down to TINY.

size by 4× while minimally reducing accuracy: the largest drop is -0.6% for BERT-BASE (supporting Zafrir et al. (2019)), while other architecture sizes are nearly unaffected. KD (orange) does not reduce a given architecture's size, but instead yields a consistent boost to accuracy, especially for smaller architectures. This upward shift on the accuracy-model size curve means that larger models can be downsized more effectively: *e.g.*, LARGE's baseline accuracy is matched by the KD version of BASE, and KD SMALL outperforms baseline MEDIUM. Compared to QAT or KD, MP (green) is only modestly helpful. Typically, 40% of encoder weights can be removed without much impact, but pruning 60% (*i.e.*, the second point in each set of two) degrades accuracy. Also, removing 40% of encoder weights corresponds to a $< 40\%$ model size reduction because we do not prune embedding weights. Therefore, while MP can be helpful for LARGE and BASE, it cannot significantly compress small architectures, which have a higher percentage of their weights in embeddings.

Used together, KD mitigates accuracy losses from both QAT and MP. Moving to combinations of compression methods, we find that the most successful pair combines QAT and KD (red), which yields QAT's 4× memory reduction while retaining the improved accuracy over baseline from KD. Meanwhile, QAT+MP (purple) does poorly: though LARGE and BASE can prune 40% of weights while retaining accuracy, when they are pruned *and* quantized, they have lower accuracy than when they are only quantized. This result suggests that pruning specifically damages accuracy with quantization: practitioners should expect additive (or worse) accuracy degradation when combining QAT and MP. On the other hand, with KD+MP (brown), 40% weights can be pruned while retaining the accuracy boost from KD, for all architectures. Thus, KD *mitigates* accuracy losses from both MP and QAT. This result still holds when we combine all three methods (black). With QAT+KD+MP, 40% of encoder weights for LARGE and 60% for BASE (and smaller) can be removed while matching the accuracy of QAT+KD. In our experiments, KD completely mitigates the compounding losses from QAT+MP and even *improves* accuracy. KD enables deeper compression when practitioners combine methods.

Combining methods yields *super-multiplicative* compression ratios. Building on our qualitative findings, we were interested in quantitative estimates for how much each method allows us to compress each architecture size. So, we compute *compression ratios*, *i.e.*, the maximal size reduction factor possible while preserving accuracy to within 0.5% of baseline. For example, baseline LARGE (1341 MB) yields 92.8% accuracy on QNLI, while

	Size	QAT	KD	MP	QAT+KD	QAT+MP	KD+MP	QAT+KD+MP
LARGE	1341	3.6×	3.4×	1.7×	12.6×	3.9×	5.8×	18.1×
BASE	438	2.6×	1.8×	1.7×	5.9×	3.1×	2.8×	11.1×
MEDIUM	166	2.9×	2.0×	1.2×	8.0×	4.0×	3.5×	14.1×
SMALL	115	3.6×	2.4×	1.2×	9.4×	4.7×	3.6×	14.1×
MINI	45	3.7×	1.2×	1.1×	4.7×	3.4×	1.8×	7.0×
TINY	18	3.4×	1.0×	0.7×	4.0×	2.2×	1.0×	3.9×

Table 1: Ratios, averaged across all GLUE tasks, measuring the maximum possible size reduction factor of a certain architecture while within 0.5% of baseline accuracy. Uncompressed sizes are listed in **megabytes (MB)**.

BASE with QAT, KD, and 40% MP (76 MB) is the smallest model within 0.5% of that, at 92.6% accuracy. Therefore, BERT-LARGE's compression ratio on QNLI is $\frac{1341}{76} = 17.6\times$, and averaging this value across tasks yields a net ratio of 18.1× (top-right, Table 1). We similarly compute ratios for all architectures and compression combinations.

As before, QAT usually retains accuracy and yields a 4× size reduction. However, because there are a few tasks for which QAT causes a $> 0.5\%$ accuracy drop[2], the task-averaged compression ratios for LARGE and BASE of 3.6× and 2.6×. KD also has high mean compression ratios, because it often boosts small architectures' accuracies to match larger baseline architectures. MP yields 1.7× compression (from 40% pruning) for LARGE and BASE, and even less for the smaller architectures.

When combined, we observe synergistic compression between QAT and KD. We might expect strong diminishing returns from combining methods, but even in their absence, we would expect independent compression ratios to multiply. Strikingly, though, we often see *super-multiplicative* model size reductions with QAT+KD: *e.g.* BASE, $5.9 > 2.6 \cdot 1.8 = 4.7$; MEDIUM: $8.0 > 2.9 \cdot 2.0 = 5.8$. For KD+MP, the ratios multiply for LARGE and BASE. Also, while pruning was originally ineffective for MEDIUM and smaller, KD+MP appears to make pruning more effective, again with super-multiplicative compression (e.g. for Medium: $3.5 > 2.0 \cdot 1.2 = 2.4$). These findings show that KD mitigates the drop in accuracy from quantization (Zhang et al., 2020) and pruning (Sanh et al., 2020), supporting qualitative findings from prior literature. This mitigation effect, combined with the accuracy boost that KD provides, yields joint compressions that are more effective than the sum of their parts. Remarkably, with all three compressions, we still see super-multiplicative scaling for BASE and smaller (*e.g.*, BASE: $11.1 > 2.6 \cdot 1.8 \cdot 1.7 = 8.0$; MEDIUM: $14.1 > 2.9 \cdot 2.0 \cdot 1.2 = 7.0$). At 18.1×, LARGE is most compressible, but actually has a sub-multiplicative ratio, perhaps because the individual compressions already work very well.

4 Discussion and Future Work

We examine the relative benefits and interactions of three widely-used compression methods in NLP: knowledge distillation, INT8 quantization-aware-training, and magnitude pruning. Though multiple techniques are increasingly used simultaneously, little work has studied the empirical interactions between them.

Across six architecture sizes and eight GLUE tasks, we find that INT8 quantization is most beneficial, and combining quantization with KD mitigates any of its occasional accuracy drops. Quantization and pruning yield diminishing returns, with the two methods exacerbating accuracy losses when used together. However, when all three methods are combined, KD restores accuracy above baseline, yielding 18× and 11× net compression for BERT-LARGE and BASE. We quantitatively confirm that KD's model size improvements are complementary (often super-multiplicative) with quantization and pruning. Given these benefits, using larger, more recent GLUE winners as KD teachers may yield further gains (*e.g.* ERNIE, Sun et al. (2021)). We also hope that our observed benefits inspire authors of new compression techniques to evaluate complementarity with existing methods.

In future work, we hope to measure accuracy vs. inference time tradeoffs, and compare these results to our findings on model size. Though there is work on accelerated inference with INT8 quantization (Kim et al., 2021b) or pruning (Pool et al., 2021), it is not clear how these speedups stack. Profiling compression combinations on specialized hardware would be an informative avenue to explore next.

[2] Those tasks would be considered to have 1× compression.

Broader Impacts

As NLP models dramatically scale in size, they rely increasingly on specialized hardware (*e.g.* GPUs, TPUs) to train and deploy them. The manufacturing and energy consumption involved in the usage of such devices imposes a significant carbon footprint (Strubell et al., 2019; Gupta et al., 2021). Model compression is part of the broader movement towards "Green AI", in which researchers develop more energy-efficient models with similar task accuracies in order to reduce usage of compute-hungry hardware (Schwartz et al., 2020).

By careful empirical study of how to optimally combine compression methods, we believe our work takes further steps towards Green AI. In particular, rather than proposing a single architecture that achieves a specific tradeoff, we equip practitioners with a set of principles to apply depending on their needs, hopefully increasing uptake of at least some subset of compression methods. Applying our insights to widely-used deployment settings (speech-to-text, search, *etc.*) could significantly reduce AI's consumptive footprint.

Though model compression methods can dramatically mitigate deep learning's carbon footprint, they may also create opportunities for harm (Suresh and Guttag, 2021). Specifically, recent work shows that pruning damages accuracy for minority classes in the training dataset (Hooker et al., 2020), and that pruning may change model behavior even when accuracy is preserved (Movva and Zhao, 2020). Such predictive disparities can lead to algorithmic harms: *e.g., representational* harms for language models, or *allocational* harms for certain downstream task predictors (Blodgett et al., 2020). More work is needed to systematically characterize the relationship between compression and algorithmic harms.

References

Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jin Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. 2021. BinaryBERT: Pushing the limit of BERT quantization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4334–4348, Online. Association for Computational Linguistics.

Su Lin Blodgett, Solon Barocas, Hal Daumé III, and Hanna Wallach. 2020. Language (Technology) is Power: A Critical Survey of "Bias" in NLP. *arXiv:2005.14050 [cs]*. ArXiv: 2005.14050.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. 2020. The lottery ticket hypothesis for pre-trained bert networks. In *Advances in Neural Information Processing Systems*, volume 33, pages 15834–15846. Curran Associates, Inc.

Dipankar Das, Naveen Mellempudi, Dheevatsa Mudigere, Dhiraj D. Kalamkar, Sasikanth Avancha, Kunal Banerjee, Srinivas Sridharan, Karthik Vaidyanathan, Bharat Kaul, Evangelos Georganas, Alexander Heinecke, Pradeep Dubey, Jesús Corbal, Nikita Shustrov, Roman Dubtsov, Evarist Fomenko, and Vadim O. Pirogov. 2018. Mixed precision training of convolutional neural networks using integer operations. *CoRR*, abs/1802.00930.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Prakhar Ganesh, Yao Chen, Xin Lou, Mohammad Ali Khan, Yin Yang, Hassan Sajjad, Preslav Nakov, Deming Chen, and Marianne Winslett. 2021. Compressing large-scale transformer-based models: A case study on bert. *Transactions of the Association for Computational Linguistics*, 9:1061–1080.

Mitchell A. Gordon, Kevin Duh, and Nicholas Andrews. 2020. Compressing BERT: Studying the Effects of Weight Pruning on Transfer Learning. *arXiv:2002.08307 [cs]*. ArXiv: 2002.08307.

Manish Gupta and Puneet Agrawal. 2020. Compression of deep learning models for text: A survey. *arXiv preprint arXiv:2008.05221*.

Udit Gupta, Young Geun Kim, Sylvia Lee, Jordan Tse, Hsien-Hsin S. Lee, Gu-Yeon Wei, David Brooks, and Carole-Jean Wu. 2021. Chasing Carbon: The Elusive Environmental Footprint of Computing. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 854–867. ISSN: 2378-203X.

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

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks. *arXiv:2102.00554 [cs]*. ArXiv: 2102.00554.

Sara Hooker, Nyalleng Moorosi, Gregory Clark, Samy Bengio, and Emily Denton. 2020. Characterising Bias in Compressed Models. *arXiv:2010.03058 [cs]*. ArXiv: 2010.03058.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2020. Dynabert: Dynamic bert with adaptive width and depth. In *Advances in Neural Information Processing Systems*, volume 33, pages 9782–9793. Curran Associates, Inc.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2017. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *arXiv:1712.05877 [cs, stat]*. ArXiv: 1712.05877.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2704–2713. IEEE Computer Society.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. Tinybert: Distilling bert for natural language understanding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 4163–4174.

Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. 2021a. I-BERT: Integer-only BERT Quantization. *arXiv:2101.01321 [cs]*. ArXiv: 2101.01321.

Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2021b. I-bert: Integer-only bert quantization. *arXiv preprint arXiv:2101.01321*.

Ronny Krashinsky, Oliver Giroux, Stephen Jones, Nick Stam, and Sridhar Ramaswamy. 2020. NVIDIA Ampere Architecture In-Depth.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are Sixteen Heads Really Better than One? *arXiv:1905.10650 [cs]*. ArXiv: 1905.10650.

Rajiv Movva and Jason Y. Zhao. 2020. Dissecting Lottery Ticket Transformers: Structural and Behavioral Study of Sparse Neural Machine Translation. *arXiv:2009.13270 [cs, stat]*. ArXiv: 2009.13270.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Jeff Pool, Abhishek Sawarkar, and Jay Rodge. 2021. Accelerating inference with sparsity using the nvidia ampere architecture and nvidia tensorrt. *NVIDIA Developer Blog*.

Alex Renda, Jonathan Frankle, and Michael Carbin. 2020. Comparing Rewinding and Fine-tuning in Neural Network Pruning. *arXiv:2003.02389 [cs, stat]*. ArXiv: 2003.02389.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in bertology: What we know about how bert works. *Transactions of the Association for Computational Linguistics*, 8:842–866.

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

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

Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. 2020. Green AI. *Communications of the ACM*, 63(12):54–63.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and Policy Considerations for Deep Learning in NLP. *arXiv:1906.02243 [cs]*. ArXiv: 1906.02243.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. 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)*, pages 4323–4332.

Yu Sun, Shuohuan Wang, Shikun Feng, Siyu Ding, Chao Pang, Junyuan Shang, Jiaxiang Liu, Xuyi Chen, Yanbin Zhao, Yuxiang Lu, Weixin Liu, Zhihua Wu, Weibao Gong, Jianzhong Liang, Zhizhou Shang, Peng Sun, Wei Liu, Xuan Ouyang, Dianhai Yu, Hao Tian, Hua Wu, and Haifeng Wang. 2021. ERNIE 3.0: Large-scale Knowledge Enhanced Pre-training for Language Understanding and Generation. *arXiv:2107.02137 [cs]*. ArXiv: 2107.02137.

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

Harini Suresh and John Guttag. 2021. A Framework for Understanding Sources of Harm throughout the Machine Learning Life Cycle. In *Equity and Access in Algorithms, Mechanisms, and Optimization*, EAAMO '21, pages 1–9, New York, NY, USA. Association for Computing Machinery.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-Read Students Learn Better: On the Importance of Pre-training Compact Models. *arXiv:1908.08962 [cs]*. ArXiv: 1908.08962.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned. *arXiv:1905.09418 [cs]*. ArXiv: 1905.09418.

Alex Wang, Jan Hula, Patrick Xia, Raghavendra Pappagari, R. Thomas McCoy, Roma Patel, Najoung Kim, Ian Tenney, Yinghui Huang, Katherin Yu, Shuning Jin, Berlin Chen, Benjamin Van Durme, Edouard Grave, Ellie Pavlick, and Samuel R. Bowman. 2019a. Can you tell me how to get past sesame street? sentence-level pretraining beyond language modeling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4465–4476, Florence, Italy. Association for Computational Linguistics.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019b. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In the Proceedings of ICLR.

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. 2020. Structured Pruning of Large Language Models. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6151–6162. ArXiv: 1910.04732.

Haonan Yu, Sergey Edunov, Yuandong Tian, and Ari S. Morcos. 2020. Playing the lottery with rewards and multiple languages: lottery tickets in RL and NLP. *arXiv:1906.02768 [cs, stat]*. ArXiv: 1906.02768.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. *arXiv preprint arXiv:1910.06188*.

Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020. Ternarybert: Distillation-aware ultra-low bit bert. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 509–521.

A Experiment Details

A.1 Experiment Breakdown

In total, we tested 576 experimental conditions, each of which involves fine-tuning a model on a GLUE task. We used eight GLUE tasks: SST-2, MRPC, STS-B, QQP, MNLI, MNLI-MM, QNLI, and RTE. We excluded CoLA and WNLI from the pruning experiments to reduce the computational burden (more on the compute budget below), and because there are some known issues with WNLI that make it difficult for a fair evaluation (Wang et al., 2019a).

We used six architecture sizes (details about the architectures are in Table 2), and eight subsets of compression methods (including the baseline of no compression). Note that, when we combine compression methods, there is no concept of order, because all methods function simultaneously and independently: QAT simply adds additional operations after each Linear layer, KD only modifies the loss, and MP gradually masks more weights. Therefore, these eight subsets are exhaustive.

There were twice as many pruning experiments as non-pruning experiments, since each pruning experiment tested two different sparsity levels[3]. So, there were 4 compression subsets without pruning, 4 compression subsets with 40% pruning, and 4 compression subsets with 60% pruning. Overall, there were $6 \cdot 8 \cdot (3 \cdot 4) = 576$ experimental conditions.

A.2 Training & Hyperparameters

For each experiment, we started by initializing the BERT architecture with pretrained weights from Turc et al. (2019). We fine-tuned for 3 epochs (on either the base or augmented GLUE training set, depending on whether we were performing a distillation experiment). As is standard for BERT fine-tuning on GLUE, batch size and LR can have a significant effect on the results (Devlin et al., 2018; Turc et al., 2019). For each experimental condition, we tested three batch sizes ({8, 16, 32}) and four learning rates ({1e-5, 2e-5, 3e-5, 4e-5} for Large and Base; {3e-5, 5e-5, 0.0001, 0.0003} for Medium/Small/Mini/Tiny). For each hyperparameter combination, we performed five repetitions, so there were $3 \cdot 4 \cdot 5 = 60$ total training runs per experimental condition. We report max accuracy across these 60 runs rather than taking an average, as the BERT training on GLUE can be unstable and lead to poor results a high fraction of the time (Devlin et al., 2018). We use the public GLUE development sets rather than the official test sets, since it wouldn't have been feasible to make thousands of submissions to the GLUE testing portal.

Overall, then, we performed $576 \cdot 60 = 34560$ fine-tuning experiments. This was feasible because the smaller architectures could be fine-tuned quickly (from an hour for Medium to a few minutes for Tiny, on the largest GLUE tasks). We performed all experiments on NVIDIA V100 GPUs, and all told, we would estimate approximately 75K GPU hours were necessary for our experiments. As we set a rough budget of 100K GPU hours, this was the reason why we had to make decisions like excluding two GLUE tasks (CoLA, WNLI), not performing data augmentation for non-distillation experiments, and not performing pretraining distillation; any of these decisions would have ballooned our experimental burden. We recognize the privilege of having had access to as much GPU time as we did, and hope that other researchers can benefit from this thorough empirical analysis.

B Task-Specific Results

In Figure A1, we plot similar curves to Figure 1, split by each task. The data from Figure 1 are replicated in the top-left plot. Tasks are ordered by size (reading left-to-right and then top-to-bottom). Most tasks have concordant trends with the curves for average GLUE performance as discussed in the main text, but there are a couple exceptions.

First, for magnitude pruning, we find that some tasks experience *worse* accuracy-size tradeoffs than baseline when pruned to 40% of weights remaining; this was typically not the case when all task accuracies are averaged. Specifically, for some tasks (*e.g.* SST-2, STS-B, MRPC), pruning significantly degrades accuracy below baseline. Smaller BERT architectures are especially harmed, since they are already under-parameterized compared to BERT-Large and Base. This finding agrees with Chen et al. (2020), who find that these particular GLUE tasks are least prunable while preserving accuracy (they also use magnitude pruning, but

[3] Actually, we tested three sparsities (also including 80%), but we only show experiments from 40% and 60% in the main text. This was because 80% sparse models generally performed poorly, and fell outside the accuracy-model size frontier, so they did not affect our results – which focused on the best possible tradeoffs for each method. We show these additional results in Appendix C.3.

Architecture	# Layers	Hidden Dim.	Params (Millions)	Size (MB)	Avg GLUE
LARGE	24	1024	367	1341	88.47
BASE	12	768	134	438	87.39
MEDIUM	8	512	57	166	85.11
SMALL	4	512	45	115	83.29
MINI	4	256	19	45	81.41
TINY	2	128	8	18	78.10

Table 2: Information on the different BERT architecture sizes we use in our experiments, with pretrained versions of each size downloaded from (Turc et al., 2019) (in accordance with their license). The "Avg GLUE" column is the mean GLUE accuracy across the eight tasks included in our experiments.

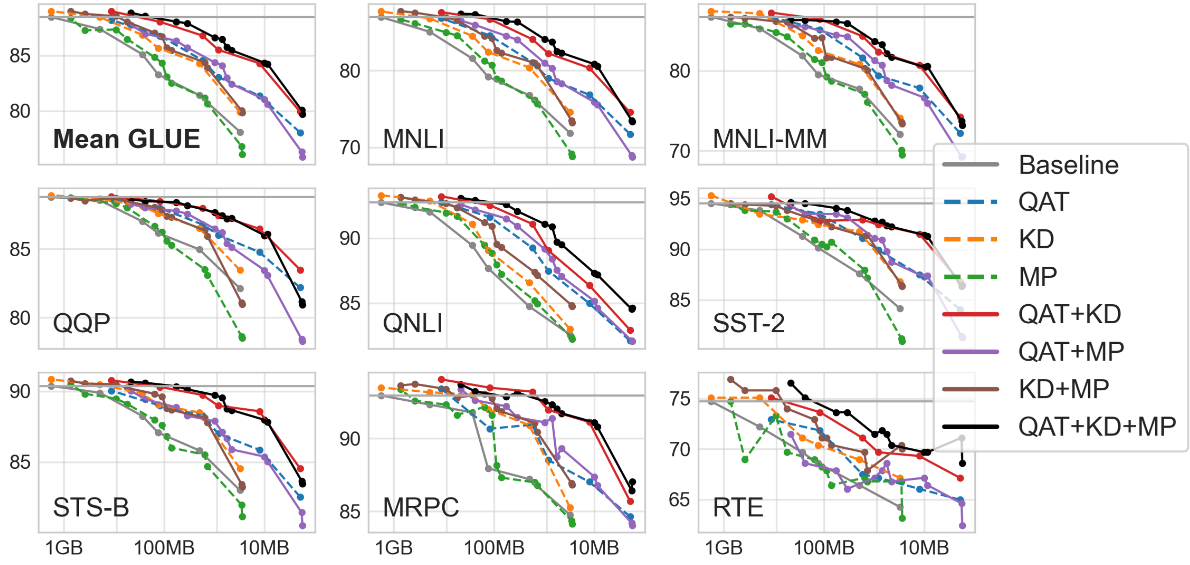

Figure A1: For each task, dev set accuracy vs. decreasing model size (from >1GB down to 10MB), with curves plotted for each compression combination. Tasks are ordered by training dataset size (left-to-right). The grey horizontal line in each plot is the baseline accuracy for BERT-LARGE.

a more compute-intensive version, allowing for slightly higher sparsities than we report).

Second, we find that for MRPC and RTE, the curves appear noisier, making some trends hard to discern. This is for two reasons: one, the tasks have the smallest training dataset, so they tend to degrade more in response to pruning (Chen et al., 2020). Second, we empirically found that these tasks varied more in their accuracy from run-to-run than other tasks (perhaps also because of their smaller training datasets). Thus, the true trends for different compression methods may be obscured by lower-confidence accuracy metrics.

C Implementation Details

C.1 Quantization-Aware-Training

In this work, we use quantization-aware-training (QAT) rather than naive post-training quantization (PTQ). PTQ quantizes weights after training and can significantly increase error due to the loss of precision. Recently, QAT has been more common, in which the effects of weight quantization are simulated during training with fake quantization operations (Jacob et al., 2017). Therefore, at inference time, the model's weights are better tailored to accommodate a reduced precision.

We specifically quantize the embedding and linear modules in our BERT architecture to use INT8 weights, following the symmetric linear quantization scheme from Q8BERT (Zafrir et al., 2019). The following quantization operation is applied to weights and activations, with scaling factor S and max value M, to quantize a value x:

$$\mathrm{Quantize}(x \mid S, M) = \mathrm{Clamp}(\lfloor x \cdot S \rceil, -M, M),$$

where $\lfloor \cdot \rceil$ is the integer rounding function, and

$\text{Clamp}(\cdot, -M, M)$ maps out-of-range values to $-M$ or M. M is determined by the number of bits; with 8 bits, for example, we have up to 256 possible quantization levels, so $M = 127$. Following Zafrir et al. (2019), the scaling factor S is set so that the largest possible value for a weight or activation matrix gets quantized to M. Thus, for a weight matrix W, the scale S^W is given by

$$S^W = \frac{M}{\max |W|}.$$

For activations x from a given layer L, the scale factor S^x is computed as an exponential moving average of the max activation value during training,

$$S^x = \frac{M}{\text{EMA}(\max_L |x|)}.$$

For quantization-aware-training, we add fake quantization ops to the model's weight matrices and activations during the training forward pass, therefore simulating the effect of quantization on each layer's output. However, $\text{Quantize}(\cdot)$ is not differentiable due to the rounding operation, so the backward pass simply ignores the quantization op using the straight-through-estimator: $\partial x^q / \partial x = \vec{\mathbf{1}}$. We model our fake quantization ops off the implementation in Intel's `nlp-architect`[4] repository, authored by Zafrir et al. (2019) and others.

C.2 Knowledge Distillation

Knowledge Distillation (KD) aims to transfer the knowledge from a large teacher model into a smaller student model: ideally, our student's predictions will emulate the teacher's, but with reduced compute cost. Formally, models trained with KD learn to minimize $\mathcal{L}_{\text{KD}}$, the difference between the teacher's and student's functions f^T and f^S, across the training set $\mathcal{X}$:

$$\mathcal{L}_{\text{KD}} = \sum_{x \in \mathcal{X}} L\left(f^T(x), f^S(x)\right).$$

The functions $f^T(\cdot)$ and $f^S(\cdot)$ include the final output probabilities, and $L(\cdot)$ measures the cross entropy between the student and teacher predicted probabilities (Sanh et al., 2019).

While some approaches perform distillation during BERT pretraining, we only distill during task fine-tuning, which is also common. Focusing on fine-tuning was necessary to make our experimental search space tractable, since BERT pretraining can take multiple orders of magnitude more compute than fine-tuning. Task-specific distillation is also more critical to preserving accuracy than pretraining distillation (Jiao et al., 2020). We follow Jiao et al. (2020) in augmenting the GLUE datasets by copying examples and replacing words with synonyms. By running an ablation, they find that augmentation is useful for all tasks, and especially ones with less data (*i.e.,* CoLA and MRPC benefit much more than MNLI). They use different augmentation factors for each dataset, either scaling up the size by 10, 20, or 30 times. We use the same values in our work, copied here: {MNLI: 10, QQP: 10, QNLI: 20, SST-2: 20, STS-B: 30, MRPC: 30, RTE: 30}.

We directly used their script, `data_augmentation.py`, from the `TinyBERT`[5] Github repository. For each GLUE training dataset, this script generates an expanded file in the same format, except with multiple words replaced with synonyms (*i.e.*, L2 nearest neighbors from GLoVe embeddings (Pennington et al., 2014)). We then take our teacher model (fine-tuned BERT-Large on the same GLUE task) and generate predicted probabilities for every example in the augmented dataset, which includes the original sentences and their synonym-replacements. Note that, when using multiple synonyms, some of the sentences change meaning, leading to a substantially different prediction than the original sentence. This is another reason (in addition to little observable change in performance and our compute budget) that we did not use augmentation for the experiments which did not use distillation.

C.3 Pruning

We use the magnitude pruning setup from the `nn_pruning` Github repository[6] (Sanh et al., 2020). Importantly, because the `nn_pruning` implementation of a "pruned" model stores the weight values along with a dictionary of weights to be masked, the real model sizes on disk are not smaller. **So, the pruned model sizes that we report are theoretical rather than actual.** That said,

[4] https://github.com/IntelLabs/nlp-architect

[5] https://github.com/yinmingjun/TinyBERT; no license visible on Github.

[6] https://github.com/huggingface/nn_pruning; license allows commercial and private use.

it would be easy to attain a true size reduction if, for example, the weights were changed to zeroes and the model file was gzipped.

As in the movement pruning approach, we prune gradually throughout training. We prune to three possible sparsity levels: 40%, 60%, or 80% sparsity. Specifically, weights are masked on a linear schedule after 5000 warmup steps, meaning that a constant number of weights are masked at each step in order to reach the target sparsity by the end of training.

For most architectures, the 80% pruning results were very poor: they caused significant accuracy degradation, so they did not yield accuracy-model size tradeoffs that were on the optimal frontier. These results did not affect our conclusions, so we removed the 80% sparse points from Figure 1. We display these results here in the Appendix (Figure A2), by showing the same plots as Figure A1, but with the 80% sparse points added to the curves. There is often a steep accuracy dropoff from 60% to 80% sparsity, especially for the smaller tasks (STS-B, MRPC, RTE).

We acknowledge that there are some marginal improvements on magnitude pruning or other forms of weight pruning that may yield better results for some architectures or tasks. For example, Chen et al. (2020) use iterative magnitude pruning with multiple rounds of full training to achieve higher than 40% sparsities without accuracy loss. However, the goal of our work is not necessarily to achieve the largest model size reductions possible, but rather to understand how methods interact; therefore, we think our conclusions on magnitude pruning would hold even with slight modifications to the method.

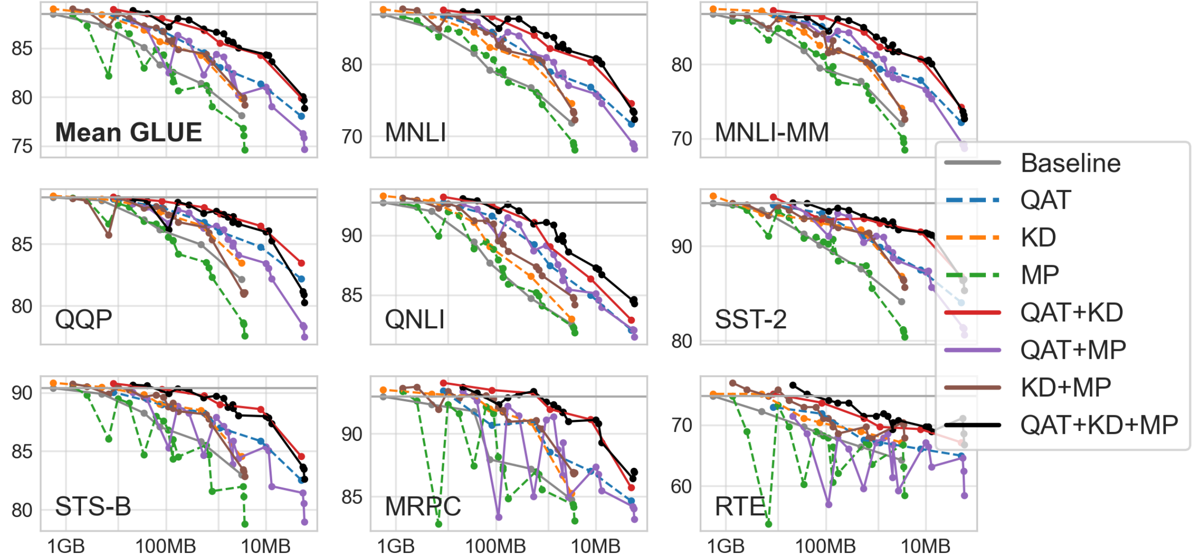

Figure A2: Accuracy vs. decreasing model size; same as Figure A1, but with the 80% pruning experiments also included (*i.e.*, 20% of weights remaining). There is a large dropoff when 80% of weights are pruned compared to 60%, especially for smaller tasks.