BMCook: A Task-agnostic Compression Toolkit for Big Models

Zhengyan Zhang[1], **Baitao Gong**[2], **Yingfa Chen**[1], **Xu Han**[1], **Guoyang Zeng**[2]
Weilin Zhao[1], **Yanxu Chen**[3], **Zhiyuan Liu**[1,4†], **Maosong Sun**[1,4†]
[1] DCST, Institute for AI, BNRIST, Tsinghua University, Beijing, China
[2] Model Best Inc. [3] Zhili College, Tsinghua University, Beijing, China
[4] International Innovation Center of Tsinghua University, Shanghai, China
zy-z19@mails.tsinghua.edu.cn {liuzy,sms}@tsinghua.edu.cn

Abstract

Recently, pre-trained language models (PLMs) have achieved great success on various NLP tasks and have shown a trend of exponential growth in model size. To alleviate the unaffordable computational costs brought by the size growth, model compression has been widely explored. Existing efforts have achieved promising results in compressing medium-sized models for specific tasks, while task-agnostic compression for big models with over billions of parameters is rarely studied. Task-agnostic compression can provide an efficient and versatile big model for both prompting and delta tuning, leading to a more general impact than task-specific compression. Hence, we introduce a task-agnostic compression toolkit BMCook for big models. In BMCook, we implement four representative compression methods, including quantization, pruning, distillation, and MoEfication. Developers can easily combine these methods towards better efficiency. To evaluate BMCook, we apply it to compress T5-3B (a PLM with 3 billion parameters). We achieve nearly 12x efficiency improvement while maintaining over 97% of the original T5-3B performance on three typical NLP benchmarks. Moreover, the final compressed model also significantly outperforms T5-base (a PLM with 220 million parameters), which has a similar computational cost. BMCook is publicly available at https://github.com/OpenBMB/BMCook.

1 Introduction

As the sizes of pre-trained language models (PLMs) increase, especially after reaching 10 billion parameters (Brown et al., 2021; Rae et al., 2021; Zhang et al., 2021a, 2022a; Chowdhery et al., 2022; Black et al., 2022), powerful intelligence capabilities emerge in these big models, supporting PLMs to accomplish tasks that previous smaller models could not do, such as quantitative reasoning (Lewkowycz et al., 2022) and long-form question answering (Nakano et al., 2021). Despite the success of big models, their exponentially growing sizes impose unaffordable computational costs for real-world applications.

	Q	P	D	M
TensorFlow (Abadi et al., 2016)	✓	✓		
PyTorch (Paszke et al., 2019)	✓	✓		
TextPruner (Yang et al., 2022)		✓		
TextBrewer (Yang et al., 2020)		✓	✓	
BMCook (this work)	✓	✓	✓	✓

Table 1: Comparisons between different compression toolkits. "Q", "P", "D", and "M" denote quantization, pruning, distillation, and MoEfication, respectively. Our BMCook is the first compression toolkit to support all four compression techniques.

To improve the efficiency of PLMs, model compression is an essential solution. There are several compression techniques, including model distillation (Hinton et al., 2015), model quantization (Bai et al., 2021), and model pruning (Liang et al., 2021). Based on these techniques, practitioners can conduct task-specific compression during fine-tuning (Sun et al., 2019) and task-agnostic compression during pre-training (Sanh et al., 2019). Previous studies mainly focus on applying task-specific compression for medium-sized PLMs with around one-hundred million parameters, such as $\text{BERT}_{\text{BASE}}$ (Zafrir et al., 2019; Jiao et al., 2020; Hou et al., 2020; Xia et al., 2022), while compressing large-scale PLMs with over billions of parameters is rarely studied.

In this work, we focus on the task-agnostic compression of big models because it enables developers to utilize the powerful intelligence of big models with fewer computation resources for both prompting (Brown et al., 2021) and delta tuning (aka parameter-efficient tuning) (Houlsby et al., 2019; Ding et al., 2022). Both prompting and delta tuning are the current core approaches to drive big

† Corresponding authors

Proceedings of the The 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 396 - 405

models. There exist two challenges for the task-agnostic compression of big models. First, big models require high compression rates to achieve affordable costs while existing compression toolkits only support one or two techniques as shown in Table 1, which cannot provide enough compression rates. Second, existing compression implementation ignores the memory challenge brought by big models. They are usually based on HuggingFace Transformers (Wolf et al., 2020), which cannot well support the training of large-scale PLMs.

In this work, we introduce BMCook, a task-agnostic compression toolkit for big models. BMCook has three main characteristics: (1) *Zero-redundancy training*. BMCook is developed on an efficient training toolkit, BMTrain[1], which supports the zero-redundancy optimizer with offloading (Rajbhandari et al., 2020; Ren et al., 2021a) to handle the memory challenge. (2) *Flexible combination*. To achieve better efficiency, we make BMCook flexible to support arbitrary combinations of different compression techniques. To this end, we implement four popular compression techniques and distribute each technique to different parts of one unified training life-cycle. (3) *Runtime model modification*. Since some compression techniques require to access the inner hidden states of PLMs, developers have to modify the code of model implementation provided by a third-party package. To make the compression easier to operate, BMCook implements runtime modification by monkey patch to get rid of modifying the source code of PLMs.

We evaluate the effectiveness of BMCook on T5-3B (Raffel et al., 2020), a T5 model with 3 billion parameters. Experimental results show that BMCook achieves nearly 12x compression efficiency by combining all four techniques while maintaining over 97% original performance on three typical NLP benchmarks, including SST-2 (Socher et al., 2013), MNLI (Williams et al., 2018), and SQuAD (Rajpurkar et al., 2016). Besides, the compressed model significantly outperforms T5-base, which has similar computation costs.

BMCook is supported by Open Lab for Big Model Base (OpenBMB)[2]. We hope BMCook can help researchers explore better compression methods for large-scale PLMs in the future and help practitioners to improve their model efficiency in real-world applications.

[1] https://github.com/OpenBMB/BMTrain

[2] https://www.openbmb.org/en/home

2 Design and Implementation

As mentioned in the introduction, we implement three main characteristics in BMCook, zero-redundancy training, flexible combination, and runtime model modification. In this section, we will introduce the design and the implementation details of these three characteristics.

2.1 Zero-redundancy Training

Due to the outrageous model size, big models require large memory to store their parameters and optimizer states, which cannot be maintained in one GPU. Recently, zero-redundancy optimizer has been proposed to solve this problem (Rajbhandari et al., 2020), which distributes the parameters and the optimizer states to multiple GPUs instead of storing all of them in one GPU repetitively. If more GPUs are used, each GPU requires less memory, which can alleviate the memory challenge brought by big models. Since BMCook targets the compression of big models, the training of big models is an important part. Therefore, we implement BMCook based on an efficient training toolkit — BMTrain, which supports zero-redundancy optimizer with parameter checkpointing (Chen et al., 2016) and offloading (Ren et al., 2021b).

2.2 Flexible Combination

Previous work on model compression usually explores one or two specific techniques. Due to the huge model size, we have to combine different techniques to achieve extreme compression. Hence, BMCook explores to build a unified compression framework that can support different techniques. Specifically, BMCook supports model distillation, model pruning, model quantization, and model MoEfication. By better utilizing these techniques, we distribute them into different parts of one unified life-cycle as shown in Figure 1. With this scope, we decouple these techniques in the implementation and support arbitrary combinations. Next, we will show more details about these techniques.

Model quantization aims to represent parameters by low-bit fixed-precision values and reduce both the memory and computational costs. For example, the computation of an 8-bit quantized model is 4 times faster than that of a 32-bit model. There are two main ways to quantize the parameters, post-training quantization and quantization-aware training. Current deep learning frameworks, such as PyTorch (Paszke et al., 2019) and Tensor-

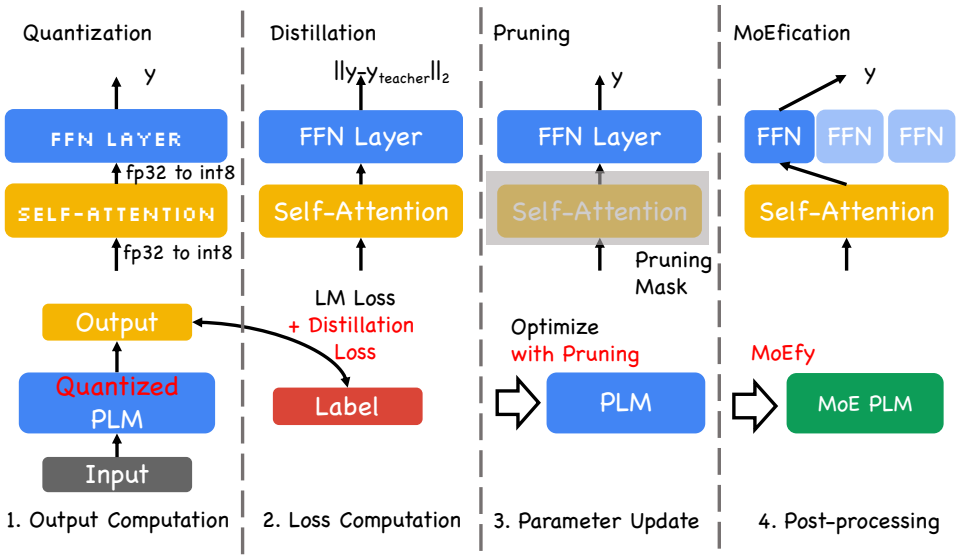

Figure 1: Life-cycle of the training process, including output computation, loss computation, parameter update, and post-processing. Each computation technique is bundled into a specific step. Specifically, quantization influences the output computation, distillation influences the loss computation, pruning influences the parameter update, and MoEfication influences the post-processing.

Flow (Abadi et al., 2016), have already supported post-training quantization. Post-training quantization directly quantizes the parameters of a PLM, which may bring a significant performance degradation. To alleviate the degradation caused by quantization, Stock et al. (2021) propose quantization-aware training (QAT). It simulates the quantization during the training, i.e., the parameters are quantized during the forward propagation, making the parameters adapted to low-bit fixed-precision computation.

Towards better performance, BMCook supports QAT. Specifically, we replace all linear layers in PLMs with quantized linear layers. In quantized linear layers, we simulate the quantized matrix multiplication. Since the linear layers account for more than 90% of the computation in the Transformer (Han et al., 2022), model quantization brings significant efficiency improvement.

Model distillation aims to guide the training of a compressed model by a larger teacher model. Traditional distillation adds the KL divergence between the outputs of teacher models and student models as an extra training objective (Hinton et al., 2015). For PLMs, Sun et al. (2019); Jiao et al. (2020); Liu et al. (2022); Park et al. (2021) find that it is also effective to make the inner computation results of student models close to those of their teachers. For example, they add the MSE loss between the hidden states of student models and teacher models.

Note that the model distillation module in BMCook is only to provide additional training loss instead of reducing the size of the model. Any compression technique requiring further training can be combined with model distillation to improve the performance of compressed models.

Model pruning aims to prune the redundant parameters of a model. There are two typical approaches, structured pruning and unstructured pruning. Structured pruning removes complete modules (e.g., layers) from the model (Fan et al., 2020; Wang et al., 2020; Zhang et al., 2021b; Xia et al., 2022). Instead, unstructured pruning removes individual parameters from the model (Han et al., 2015; Chen et al., 2020; Xu et al., 2021). Both of them change the forward and backward process of the model according to their pruned parameters. To decouple pruning and quantization, we distribute the pruning operations to the optimization step, where we set the pruned parameters to zero after parameter update. Due to this, we keep redundant parameters pruned during the forward and backward processes without directly affecting these processes.

Note that unstructured pruning cannot guarantee efficiency improvement in most cases because parallel processing devices, such as GPUs, usually do not provide optimized sparse computation operations (Zheng et al., 2022). Hence, BMCook implements unstructured pruning with 2:4 sparsity, which is well supported by Sparse Tensor

```
def _forward(module_self, x):
    x = module_self.forward_old(x)
    bmt.inspect.record_tensor(x, "
    hidden_states")
    return x

module.forward_old = module.forward
module.forward = types.MethodType(
    _forward, module)
```

Figure 2: Example of monkey patch. Add a tensor recording step to a forward function.

```
config = ConfigParser(args.config)
# for distillation
Trainer.forward = BMDistill.set_forward(
    model, teacher, Trainer.forward,
    config)
# for pruning
BMPrune.compute_mask(model, config)
BMPrune.set_optim_for_pruning(optimizer)
# for quantization
BMQuant.quantize(model, config)
# for moefication
Trainer.forward = BMMoE.get_hidden(model
    , config, Trainer.forward)
```

Figure 3: Based on the configuration file, practitioners can turn on a specific compression module with one or two lines of code.

Core (Zhou et al., 2021). 2:4 sparsity means that every four continuous parameters have two zeros. In this way, the sparse computation is guaranteed to be twice as fast as the dense computation. Besides, for structured pruning, we implement CoFi (Xia et al., 2022) in BMCook, which adds L0-regularization to the parameters of the model to learn an optimal sparse mask.

MoEfication aims to transform the feedforward networks (FFNs) in Transformers to the equivalent mixture-of-expert (MoE) version (Fedus et al., 2021), which significantly reduces the computational costs of FFNs (Zhang et al., 2022b). Since Transformers (Vaswani et al., 2017) adopt ReLU (Nair and Hinton, 2010) as the activation function of FFNs and there exists an obvious sparse activation phenomenon, we can only involve part of FFNs for a specific input without affecting the model performance. The transformation process does not change the number and the values of model parameters. Therefore, we treat MoEfication as a post-processing technique. It can be applied to any compressed model to achieve better efficiency.

To train routers for expert selection, MoEfication requires the hidden states to simulate the computation process of FFNs. The training of routers is localized to specific FFNs and is dealt with by an external MoEfication package.

In summary, BMCook is the first to contain a series of compression techniques. And, benefiting from the decoupled implementation of compression techniques, practitioners can design their own compression strategies with arbitrary combinations.

2.3 Runtime Model Modification

All of the compression techniques mentioned in the last subsection require modifying the life-cycle of the training process, i.e., the implementation code of PLMs. Taking distillation as an example, to compute the mean squared error between the hidden states of the teacher model and the student model, we have to modify the forward functions to make the hidden states become return values. Existing compression toolkits usually ask developers to modify the codes (Yang et al., 2020, 2022). For example, in the case of distillation, after developers modify the forward functions, the toolkit provides the implementation of the loss calculation.

However, the model implementation is usually provided by a third-party package, such as HuggingFace Transformers, making the manual modification inconvenient. Besides, the modification is simple and similar across different PLMs. Hence, BMCook explores to implement runtime modification in a general way to keep the source code clean and make it easy to compress different PLMs.

Specifically, we utilize the characteristic of monkey patch in Python. Monkey patch is to modify the behavior of an object at runtime. As shown in Figure 2, we first rename the original forward function of the module as `forward_old`, and then define a new forward function containing `forward_old` and a tensor recording step. Finally, we assign the new forward function to the module. The `inspect` function for recording is to store the tensor in a global dictionary. After the whole forward process is finished, we can access the tensor by its name.

Both knowledge distillation and MoEfication require accessing the hidden states of PLMs. Considering that different modules have different foward functions, e.g., attention modules take hidden states and attention masks as input, we choose to access the hidden states of layer normalization modules and provide a general interface to add tensor recording to their forward functions. The inputs of layer normalization modules are only hidden states and are widely used before or after other modules. Hence, based on layer normalization modules, we

```
{
    "distillation": {
        "ce_scale": 0,
        "mse_hidn_scale": 1,
        "mse_hidn_module": ["[post]encoder.output_layernorm:[post]encoder.output_layernorm", "[post]decoder.output_layernorm:[post]decoder.output_layernorm"],
        "mse_hidn_proj": false
    },
    "pruning": {
        "is_pruning": true, "pruning_mask_path": "prune_mask.bin",
        "pruned_module": ["ffn.ffn.w_in.w.weight", "ffn.ffn.w_out.weight", "input_embedding"],
        "mask_method": "m4n2_1d"
    },
    "quantization": { "is_quant": true},
    "MoEfication": {
        "is_moefy": false,
        "first_FFN_module": ["ffn.layernorm_before_ffn"]
    }
}
```

Figure 4: Example of the configuration file.

can access nearly all hidden states of PLMs.

Similarly, we also modify the linear layers and optimizer by monkey patching for quantization and pruning. For quantization, we replace the matrix multiplication in the `forward` functions of linear layers with a quantized one. For pruning, we modify the behavior of the optimizer's `step` function. We keep the original operation and add a pruning step after the parameter update.

In summary, BMCook utilizes runtime modification to keep the source code clean and provides general interfaces to compress different PLMs.

2.4 Usage and Configuration

Since different compression modules are decoupled in BMCook, we implement each module independently, where each module is usually a Python file and provides one or two general interfaces. Benefiting from the general interfaces, BMCook can be applied to a PLM with only a few lines of code as shown in Figure 3. The details of compression are mainly determined by a configuration file, which will be used by different compression modules. In practice, users can easily reuse the code of pre-training for compression by adding a few lines of code to import compression modules and then setting the configuration file. Note that BMCook supports the PLMs implemented based on BMTrain and ModelCneter[3] has provided BMTrain-based implementations of almost all mainstream PLMs.

As shown in Figure 4, the configuration file is a JSON file. The keys are the names of the compression modules. The values are the configurations of the compression modules. Note that the module names used in the configuration file are corresponding to the names provided by PyTorch. Therefore, BMCook can access the modules by their names.

The key of knowledge distillation is `distillation`. Currently, BMCook supports two kinds of distillation objectives, KL divergence between output distributions (turn on when `ce_scale`>0) and mean squared error (MSE) between hidden states (turn on when `mse_hidn_scale`>0). Practitioners need to specify the hidden states used for MSE by `mse_hidn_module`. Meanwhile, the dimensions of the hidden states may be different between teacher and student models. Therefore, the hidden states of the teacher model need to be projected to the same dimension as those of the student model. Practitioners can turn on `mse_hidn_proj` for simple linear projection.

The key of model pruning is `pruning`. Practitioners can turn on pruning by `is_pruning`. The pruning mask is stored in `pruning_mask_path`. The pruned modules are specified by `pruned_module`. To simplify the list, practitioners can only provide the suffix of the modules. The mask method `mask_method` is to choose the algorithm for the computation of the pruning mask.

The key of quantization is `quantization`. Practitioners can turn on quantization by `is_quant`, which will replace all linear layers with quantized

[3] https://github.com/OpenBMB/ModelCenter

	Model	Activated Model Size	SST-2 Acc	MNLI-m Acc	SQuAD 1.0 EM	SQuAD 1.0 F1
Original Model	T5-Base	0.34GB	0.9278	0.8626	0.8076	0.8890
	T5-Large	0.60GB	0.9461	0.8938	0.8474	0.9193
	T5-3B	2.42GB	0.9621	0.9087	0.8754	0.9379
Single Module	Structured Pruning	1.21GB	0.9014	0.8472	0.8072	0.8877
	Unstructured Pruning	1.21GB	0.9576	0.8946	0.8592	0.9262
	Quantization	0.60GB	0.9598	0.9075	0.8746	0.9374
	MoEfication	1.61GB	0.9529	0.8961	0.8502	0.9260
Combination	Quant + Pruning	0.30GB	0.9518	0.8902	0.8628	0.9289
	Quant + Pruning + MoE	0.20GB	0.9518	0.8819	0.8316	0.9110

Table 2: Evaluation of original models and compressed models. In the combination experiments, we use unstructured pruning due to its superior performance in the single module experiments. The size of adapters keeps the same for all PLMs. Activated model size is used to measure the compression rate because the computational cost, i.e., FLOPS, is linear to the model size.

linear layers. BMCook provides the simulation of 8-bit quantization.

The key of MoEfication is `MoEfication`. Practitioners can turn on MoEfication by `is_moefy`. The hidden states used for router training are specified by `first_FFN_module`, which is the nearest layer normalization module before each FFN. Providing the suffix of the modules is also sufficient.

3 Evaluation

To validate the effectiveness of BMCook, we study task-agnostic compression on T5-3B (Raffel et al., 2020), which has 3 billion parameters. Since task-agnostic compression would benefit various downstream tasks, we evaluate the performance of adapter tuning (Houlsby et al., 2019) of T5-3B and its compressed variants. We also study T5-Base and T5-Large, which have 220 million and 770 million parameters, respectively.

Training and evaluation data. We use the Pile dataset (Gao et al., 2020) for task-agnostic compression training, which is a large-scale corpus for pre-training language models. The training objective is masked language modeling used by T5. Note that we turn on distillation during the compression training in all experiments because we find knowledge distillation with MSE loss can improve model performance in our pilot experiments. Besides, we choose three downstream datasets for evaluation: SST-2 (Socher et al., 2013), a representative single-sentence classification dataset, MNLI (Williams et al., 2018), a representative sentence-pair classification dataset, SQuAD v1.1 (Rajpurkar et al., 2016), a representative question-answer dataset. For the first two datasets, we use accuracy as the evaluation metric. For the third dataset, we use both exact match and F1 score as the evaluation metrics. We evaluate model performance on their development sets. We adopt the same task templates and label words of the original T5 paper (Raffel et al., 2020).

Hyper-parameters. The learning rate of task-agnostic compression training is 1e-4 while that of adapter tuning ranges from 1e-6 to 1e-5. The batch size of task-agnostic compression training and adapter tuning is 32. We use 4 NVIDIA A100 GPUs in the experiments. The training step of task-agnostic compression training ranges from 10K to 50K according to the compression methods. The training epoch of adapter tuning ranges from 3 to 5.

To fairly compare the efficiency of T5-3B and its variants, we define a new metric, named *activated model size*, because Brown et al. (2021) mentioned that the computation of Transformer is linear in the model size, which excludes the embedding layer. Hence, we consider the parameters of self-attention networks and FFNs. For the original model, the activated model size is equal to its original model size. Although it is intuitive to directly compare the speedup of compressed models, there is no inference toolkit supporting all the compressed methods. Hence, we focus on the theoretical computational cost in this work.

In the evaluation, we set the pruning sparsity to 50%, i.e., we prune 50% of the parameters and reduce half of the activated model size. Besides, we quantize the parameters to 8 bits, which reduces three-fourths of the activated model size compared to the floating-point version. For MoEfication, we dynamically involve 50% of parameters in FFNs for specific input. Therefore, the activated model

size of the modified FFNs is about half of the original FFNs. Note that Transformer consists of both attention layers and FFNs and the model size of FFNs are about 70%. The final activated model size of the modified Transformer is about 66% of the original one. If we combine all three techniques, we can achieve a compressed model with about one-twelfth of the original activated size.

We report the evaluation results in Table 2. From this table, we have three observations: (1) In the experiment of single modules, quantization achieves the best efficiency and performance. Unstructured pruning achieves the second-best efficiency and performance, and significantly outperforms structured pruning. It suggests that directly removing layers may bring significant performance degradation. Besides, as a post-processing method, which does not require further pre-training, MoEfication maintains over 98% original performance while reducing 33% of the activation model size. (2) Different compression techniques can be combined to archive better efficiency while maintaining most of performance. For example, combining quantization, unstructured pruning, and MoEfication achieves a compressed model with about one-twelfth of the original activated size and maintains over 97% original performance. (3) Compressing big models can get better small models. For example, Quant+Pruning+MoE is smaller than T5-base while this model significantly outperforms T5-base.

4 Conclusion and Future Work

In this paper, we introduce a task-agnostic compression toolkit for big models, named BMCook. This toolkit contains four popular techniques and is designed to be flexible to support arbitrary combinations. Users can easily compress a PLM by adding several lines to its pre-training code and specifying the strategy in a configuration file.

In the future, there are three directions to further improve BMCook. First, we will enrich the options of existing compression techniques, such as knowledge distillation on attention matrices (Jiao et al., 2020) and extreme low-bit quantization (Bai et al., 2021). Second, there are some other compression techniques that are not covered by BMCook, such as weight sharing (Lan et al., 2020) and low-rank decomposition (Chen et al., 2021). Third, we will explore the automatic search for better compression strategies or configurations. Given a specific computation budget, we want to find the compression strategy that achieves the best model performance, which is similar to neural architecture search (Elsken et al., 2019).

Meanwhile, we will also plan to enrich the inference toolkits to support different compression techniques. Although compression techniques have been fast developed, the inference toolkits are still lagging behind. Recently, there are some efforts to support compressed models in inference, such as BMInf (Han et al., 2022) and DeepSpeed-MoE (Rajbhandari et al., 2022), while they are still limited to specific compression techniques.

Acknowledgments

This work is supported by the National Key R&D Program of China (No. 2020AAA0106502), Institute Guo Qiang at Tsinghua University and NExT++ project from the National Research Foundation, Prime Minister's Office, Singapore under its IRC@Singapore Funding Initiative.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. Tensorflow: A system for large-scale machine learning. In *Proceedings of the OSDI*, pages 265–283.

Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jin Jin, Xin Jiang, Qun Liu, Michael R. Lyu, and Irwin King. 2021. Binarybert: Pushing the limit of BERT quantization. In *Proceedings of ACL/IJCNLP*, pages 4334–4348.

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. 2022. Gpt-neox-20b: An open-source autoregressive language model. *arxiv preprint arXiv:2204.06745*.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, et al. 2021. Language models are Few-Shot learners. In *Proceedings of NeurIPS*, pages 1877–1901.

Patrick H. Chen, Hsiang-Fu Yu, Inderjit S. Dhillon, and Cho-Jui Hsieh. 2021. DRONE: data-aware low-rank compression for large NLP models. In *Proceedings of NeurIPS*, pages 29321–29334.

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 *Proceedings of NeurIPS*.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, et al. 2022. PaLM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. 2022. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20:55:1–55:21.

Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. In *Proceedings of ICLR*.

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

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2020. The Pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Song Han, Jeff Pool, John Tran, and William J Dally. 2015. Learning both weights and connections for efficient neural networks. In *Proceedings of NeurIPS*, pages 1135–1143.

Xu Han, Guoyang Zeng, Weilin Zhao, Zhiyuan Liu, Zhengyan Zhang, Jie Zhou, Jun Zhang, Jia Chao, and Maosong Sun. 2022. BMInf: An efficient toolkit for big model inference and tuning. In *Proceedings of ACL Demonstration*, pages 224–230.

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

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2020. Dynabert: Dynamic BERT with adaptive width and depth. In *Proceedings of NeurIPS*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of ICML*, pages 2790–2799.

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 *Findings of EMNLP*, pages 4163–4174.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *Proceedings of ICLR*.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay V. Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving quantitative reasoning problems with language models. *arXiv preprint arXiv:2206.14858*.

Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. 2021. Pruning and quantization for deep neural network acceleration: A survey. *arXiv preprint 2101.09671*.

Chang Liu, Chongyang Tao, Jiazhan Feng, and Dongyan Zhao. 2022. Multi-granularity structural knowledge distillation for language model compression. In *Proceedings of ACL*, pages 1001–1011.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of ICML*, pages 807–814.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2021. WebGPT: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Geondo Park, Gyeongman Kim, and Eunho Yang. 2021. Distilling linguistic context for language model compression. In *Proceedings of EMNLP*, pages 364–378.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch:

An imperative style, high-performance deep learning library. In *Proceedings of NeurIPS*.

Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, H. Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, et al. 2021. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified Text-to-Text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. 2022. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation AI scale. In *Proceedings of ICML*, volume 162, pages 18332–18346.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: memory optimizations toward training trillion parameter models. In *Proceedings of SC*, page 20.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of EMNLP*, pages 2383–2392.

Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. 2021a. Zero-offload: Democratizing billion-scale model training. In *Proceedings of ATC*, pages 551–564.

Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. 2021b. Zero-offload: Democratizing billion-scale model training. In *Proceedings of USENIX ATC*, pages 551–564.

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

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*, pages 1631–1642.

Pierre Stock, Angela Fan, Benjamin Graham, Edouard Grave, Rémi Gribonval, Hervé Jégou, and Armand Joulin. 2021. Training with quantization noise for extreme model compression. In *Proceedings of ICLR*.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for BERT model compression. In *Proceedings of EMNLP*, pages 4323–4332.

Ashish Vaswani, Noam Shazeer, Niki Parmar, and Jakob Uszkoreit. 2017. Attention is all you need. In *Proceedings of NeurIPS*, pages 5998–6008.

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. 2020. Structured pruning of large language models. In *Proceedings of EMNLP*, pages 6151–6162.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of NAACL-HLT*, pages 1112–1122.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of EMNLP Demonstration*, pages 38–45.

Mengzhou Xia, Zexuan Zhong, and Danqi Chen. 2022. Structured pruning learns compact and accurate models. In *Proceedings of ACL*, pages 1513–1528.

Dongkuan Xu, Ian En-Hsu Yen, Jinxi Zhao, and Zhibin Xiao. 2021. Rethinking network pruning – under the pre-train and fine-tune paradigm. In *Proceedings of NAACL-HLT*, pages 2376–2382.

Ziqing Yang, Yiming Cui, and Zhigang Chen. 2022. Textpruner: A model pruning toolkit for pre-trained language models. In *Proceedings of ACL*, pages 35–43.

Ziqing Yang, Yiming Cui, Zhipeng Chen, Wanxiang Che, Ting Liu, Shijin Wang, and Guoping Hu. 2020. Textbrewer: An open-source knowledge distillation toolkit for natural language processing. In *Proceedings of ACL*, pages 9–16.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8BERT: Quantized 8bit BERT. *arXiv preprint 1910.06188*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022a. OPT: open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Zhengyan Zhang, Yuxian Gu, Xu Han, Shengqi Chen, Chaojun Xiao, Zhenbo Sun, Yuan Yao, Fanchao Qi, Jian Guan, Pei Ke, Yanzheng Cai, Guoyang Zeng, Zhixing Tan, Zhiyuan Liu, Minlie Huang,

Wentao Han, Yang Liu, Xiaoyan Zhu, and Maosong Sun. 2021a. CPM-2: large-scale cost-effective pre-trained language models. *AI Open*, 2:216–224.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2022b. MoEfication: Transformer feed-forward layers are mixtures of experts. In *Findings of ACL 2022*.

Zhengyan Zhang, Fanchao Qi, Zhiyuan Liu, Qun Liu, and Maosong Sun. 2021b. Know what you don't need: Single-shot meta-pruning for attention heads. *AI Open*, 2:36–42.

Ningxin Zheng, Bin Lin, Quanlu Zhang, Lingxiao Ma, Yuqing Yang, Fan Yang, Yang Wang, Mao Yang, and Lidong Zhou. 2022. SparTA: Deep-Learning model sparsity via Tensor-with-Sparsity-Attribute. In *Proceedings of OSDI*, pages 213–232.

Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. 2021. Learning n:m fine-grained structured sparse neural networks from scratch. In *Proceedings of ICLR*.