Resource-Efficient Transformer Pruning for Finetuning of Large Models

Fatih Ilhan[1], Gong Su[2], Selim Furkan Tekin[1], Tiansheng Huang[1], Sihao Hu[1], Ling Liu[1]
[1]Georgia Institute of Technology, Atlanta, GA
[2]IBM Research, Yorktown Heights, NY
{filhan, stekin6, thuang, sihaohu}@gatech.edu, gongsu@us.ibm.com, ling.liu@cc.gatech.edu

Abstract

With the recent advances in vision transformers and large language models (LLMs), finetuning costly large models on downstream learning tasks poses significant challenges under limited computational resources. This paper presents a REsource and ComputAtion-efficient Pruning framework (RECAP) for the finetuning of transformer-based large models. RECAP by design bridges the gap between efficiency and performance through an iterative process cycling between pruning, finetuning, and updating stages to explore different chunks of the given large-scale model. At each iteration, we first prune the model with Taylor-approximation-based importance estimation and then only update a subset of the pruned model weights based on the Fisher-information criterion. In this way, RECAP achieves two synergistic and yet conflicting goals: reducing the GPU memory footprint while maintaining model performance, unlike most existing pruning methods that require the model to be finetuned beforehand for better preservation of model performance. We perform extensive experiments with a wide range of large transformer-based architectures on various computer vision and natural language understanding tasks. Compared to recent pruning techniques, we demonstrate that RECAP offers significant improvements in GPU memory efficiency, capable of reducing the footprint by up to 65%.

1. Introduction

Transformer-based neural network architectures have shown remarkable performance in a wide range of areas, e.g., computer vision, natural language understanding (NLU), and multi-modal tasks. Notably, vision transformers (ViTs) [11] and variants such as DINO [3, 30], MaskFormer [5, 6], and CLIP [32] have demonstrated exceptional performance across various tasks from image recognition, semantic segmentation, to object detection. Likewise, BERT [10], GPT [31], and T5 [33] are dominating transformer-based architectures for many popular large language models (LLMs) [2, 36]. These models are usually pre-trained over a combination of very large datasets and serve as a foundation model for a variety of tasks. Before deployment, these pre-trained large models (PLMs) can be finetuned further on a downstream task/dataset to maximize the task-specific model performance.

Problem Statement. Over-parameterization with a large number of parameters and using big datasets in the training of a foundation model is known to result in significantly better generalization performance [29]. The PLMs trained on billions of tokens result in very large models of billions of parameters and can require hundreds of GBs of GPU memory for finetuning and inference. For example, a recent open-source family of LLMs, LLama-2 [36] reports the following statistics for the largest model variant with 70B parameters: 2T tokens for ∼1.7M GPU hours with A100-80GB GPUs, which can cost around ∼$6.7M. Therefore, finetuning such PLMs on downstream tasks instead of training from scratch has become a common practice. However, employing such models may be overindulging when tackling downstream tasks. Even the smallest variant of LLama-2 with 7B parameters requires ∼14GB GPU memory for inference and even more (∼70GB) for finetuning at 16-bit precision. Effectively finetuning a PLM with limited resources while maintaining performance remains an open challenge. This problem can be aggravated when the GPU resources are insufficient to perform finetuning on the original model, making finetuning impossible for many. Furthermore, when users need to finetune a PLM over their proprietary or privacy-sensitive data (e.g., medical data, personal chats, or confidential documents), this process should only be executed in local computing environments without sharing sensitive data.

To reduce the GPU memory footprint of finetuning large transformer-based models, pruning [12, 16] emerges as a promising technique due to its success in convolutional and transformer-based models, particularly to increase inference efficiency. Pruning is based on the argument that not all weights of the pre-trained large model are necessary for a given downstream task. Hence, pruning solutions

estimate the importance of weights for the finetuning task and prune out the least important weights by the system-supplied sparsity ratio. However, representative pruning techniques [12, 16, 27] perform better when they are applied after finetuning (post-finetune pruning) because pre-finetune pruning may yield suboptimal results by removing important parameters for the finetuning task. Therefore, many existing approaches suffer from low resources during finetuning.

Contribution and Scope. We introduce RECAP, a REsource and ComputAtion-efficient Pruning framework to achieve two seemingly conflicting goals: reducing the GPU memory footprint during finetuning while maintaining the finetuned model performance. RECAP jointly utilizes the CPU and GPU to finetune the pre-trained large model in a resource-efficient manner. We use CPU resources for less intensive and low-frequency computations that require access to the full model and resort to GPU for more intensive and high-frequency computations during the finetuning process. *First*, RECAP explores different chunks of the given large-scale model through an iterative CPU-GPU collaboration cycling between pruning, finetuning, and updating stages. At each iteration, we first prune the model with Taylor-approximation-based importance estimation. Then, we upload the pruned model to the GPU, and after finetuning, we transfer the updated weights to the CPU. *Second*, we generate a finetuning mask controlled by Fisher information, which determines the subset of the pruned model that should be updated during finetuning. This prevents early saturation in the exploration process and further reduces the GPU memory footprint of gradients and optimizer states.

By iteratively finetuning various chunks of the full model, while being administrated by the CPU on which part of the model to operate on and which weights to update, RECAP only loads and finetunes a selective portion of the weights on GPU in each iteration. In this way, unlike existing pruning methods that require the model to be finetuned beforehand for better preservation of model accuracy [16, 27], RECAP can achieve high performance with significantly less GPU memory footprint. Extensive experiments are conducted on a range of large transformer-based architectures and various computer vision and natural language understanding tasks. Compared to recent large model pruning techniques, RECAP significantly improves the GPU memory efficiency (reducing the footprint by up to 65%), while maintaining competitive model quality. We also analyze the impact of design components and behavior under various pruning ratios.

2. Related Work

Among the representative efficient finetuning techniques, quantization of neural networks reduces the number of bits used to represent the model weights and enables low-bit-matrix multiplications. A major challenge is the loss of information, especially in the existence of outlier values in model weights [9]. In addition, quantization is not task-aware and the practical improvements are hardware-dependent. Several studies [24, 25] applied quantization for vision transformers but these methods are limited to post-training for reducing the model size for inference-stage deployment. Adapter-based solutions [19] represent another type of efficiency technique, where a set of trainable parameters into the pre-trained models is injected and finetuned while keeping the backbone fixed. Several variants, such as multi-task adapters [26], LoRA [20], and vision transformer adapters [4] have been proposed. Adapters reduce the memory footprint of gradients and optimizer states. However, memory footprint reduction remains limited since the full model weights have to be loaded on GPU during finetuning. Neural architecture search methods on training super networks have also been explored [14], targeting similar problems. Lastly, VPT [21] applies prompt tuning on vision transformers by attaching learnable prefix vectors to inputs at each layer and keeping model weights frozen. Even though this method is parameter-efficient, memory efficiency is limited due to significantly longer input sequences at each layer.

Model pruning reduces the model size by pruning out certain parameters based on an importance criterion, such as weight magnitude [12, 16], weight change [23, 34] or output change [28, 39], until a desired pruning rate is achieved. Recent research [28, 40] reports that pruning yields better results if executed after finetuning (post-finetune) [12, 13, 28] compared to the conventional scenarios where the pruning is applied before finetuning (pre-finetune) [16, 23]. Some studies [39] advocate eliminating the requirement of finetuning after pruning but still require full model finetuning as the first step. Most existing pruning techniques involve finetuning the full model first, which is not suitable for very large transformer models under limited GPU memory. As a result, conventional pruning techniques still require loading and finetuning the full model on GPU for best results, which limits their applicability in practice. Our approach does not have the resource bottleneck required by the finetuning of the full model because RECAP iteratively explores and partially finetunes the smaller subnetworks of the full model, therefore significantly reducing the GPU memory footprint throughout finetuning while preserving model performance.

3. Methodology

The design of RECAP is motivated by the limitations of pre-finetune pruning and post-finetune pruning. The former first prunes the full model, and then finetunes the pruned model on GPU. The main limitation is the inability to incorporate the critical information related to the downstream task during pruning, and thus suboptimal pruning of important pa-

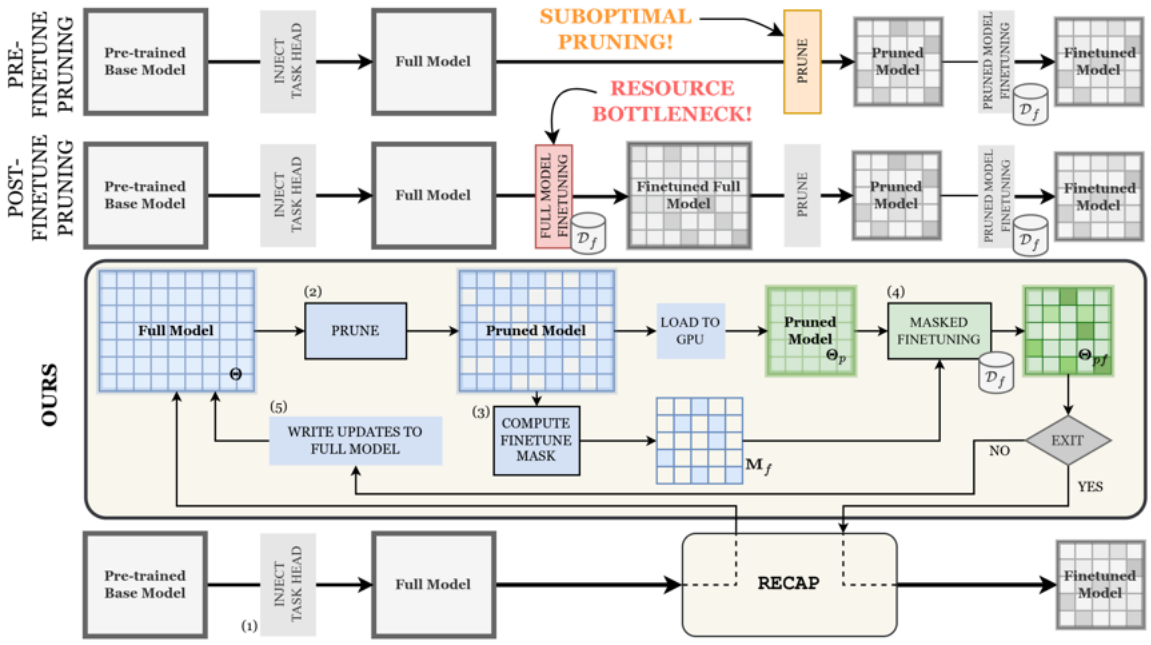

Figure 1. System architecture of RECAP. Components in blue are done on CPU, and those in green are done on GPU.

rameters of the model. In contrast, the post-finetune pruning methods conduct finetuning both before and after pruning, i.e., the full model is first finetuned on GPU followed by pruning to obtain the pruned model, and then the pruned model is finetuned again on GPU. This method can produce a high-quality model at the cost of loading and finetuning the full model on GPU. Unlike pre-finetune pruning and post-finetune pruning, RECAP introduces a new approach to enable efficient pruning during finetuning. In RECAP, we iteratively finetune various chunks of the full model to maintain a high quality in the finetuned pruned model without requiring loading and operating on the full model on GPU. To this end, RECAP leverages the CPU to determine which part of the model to operate on and which weights to update at each iteration, which reduces the GPU memory footprint caused by model weights, optimizer states, activations and gradients. Figure 1 provides the RECAP system architecture and the illustrative comparison with pre-finetune pruning and post-finetune pruning scenarios.

Concretely, we first inject task-specific heads into the pre-trained base model (Step 1). Then, at each iteration before transferring the pruned model to the GPU for finetuning, we compute importance metrics for all model weights at the CPU (Step 2). To this end, we first sample a tiny subset of the dataset and approximate how much the objective loss changes over it after removing each model weight using a first-order Taylor expansion. These importance metric values are utilized to determine which weights will be pruned out such that the weights with a high impact on the output will be preserved. To further improve the pruning efficiency, we determine which weights within the pruned model should be updated during finetuning based on their empirical Fisher such that the weights with lower gradient values will not be updated (Step 3). Consequently, their optimizer states will not be needed during finetuning, reducing the GPU memory footprint caused by optimizer states and gradients. After loading the pruned model and finetuning masks to GPU, we perform finetuning on the downstream task/dataset (Step 4). At each optimization step, we only update the weights determined by the pre-computed finetuning mask. After finetuning, we update the full model weights at CPU (Step 5). We continue the main loop from Step 2, and repeat for K rounds or until the performance on the validation dataset converges. Lastly, if more compression is necessary, our approach can be combined with quantization [9] to boost the efficiency and benefit from the orthogonal advantages.

3.1. Pruning Stage

In this subsection, we provide the details of the pruning stage operations. Given a model with weights $\boldsymbol{\theta}$ and finetuning dataset $\mathcal{D}_f$, our goal is obtaining a pruned model with $r_p\%$ pruning ratio (i.e. $r_p\%$ of the original model parameters will be pruned out).

3.1.1 Grouping Model Weights

There are mainly two pruning approaches depending on how the weights are grouped. In unstructured pruning, the pruning decision is made for each model weight separately,

resulting in unstructured sparsity in weight matrices after pruning. In the structured pruning approach, model weights are grouped (e.g. channels, filters for CNNs, hidden dimensions, heads for transformers, etc.) and the same pruning decision is made for the weights within each group together. After structured pruning, instead of being a masked version of the original model with sparse weights, the pruned model has weight matrices of smaller size, less number of heads, etc. In our method, we consider structured pruning since it is more hardware-friendly and the improvements in latency and memory are better reflected in practice.

In our case for transformers, we have two main types of model weights, respectively for attention modules and feedforward layers. For the attention module weights, each group contains the model weights (query, key, value, output weight matrices) corresponding to a head. For feedforward layers, each group contains the model weights corresponding to a hidden dimension. We consider the coupled structure of model weights within the computation graph to eliminate inconsistencies after pruning, as illustrated in Figure 2. Let us denote the weight groups for pruning as $\mathcal{G}_m^{(p)}$, where m is the group index.

3.1.2 Importance Estimation for Pruning

After grouping model weights, the next step of pruning is computing the importance of each weight group within the model. Several approaches have been proposed to define the importance criterion based on weight magnitude, activations, gradients etc [15, 18, 39]. Following the success of the Taylor expansion-based weight importance estimation approach proposed for convolutional neural networks [27], we also define the importance of each weight θ_k by the estimation of how much the loss would change if that weight were pruned. Performing separate computations for each weight over the full dataset is computationally infeasible. Therefore, we can consider the second-order Taylor expansion over a randomly sampled tiny subset $\mathcal{D}_s \subset \mathcal{D}_f$ with $|\mathcal{D}_s| \ll |\mathcal{D}_f|$ to approximate the loss change induced by the removal of each weight and then define the importance $\mathcal{I}_{\theta_k}$ as follows:

$$\mathcal{I}_{\theta_k} = |\mathcal{L}(\mathcal{D}_s) - \mathcal{L}_{\theta_k=0}(\mathcal{D}_s)| \approx \left(\frac{\partial \mathcal{L}(\mathcal{D}_s)}{\partial \theta_k}\theta_k - \frac{1}{2}\theta_k \mathbf{H}_k \boldsymbol{\theta}\right)^2,$$

where $\frac{\partial \mathcal{L}(\mathcal{D}_s)}{\partial \theta_k}$ is the mean derivative of the observed loss with respect to the weight θ_k over $\mathcal{D}_s$. $\mathbf{H} = [\frac{\partial^2 \mathcal{L}(\mathcal{D}_s)}{\partial \theta_i \partial \theta_j}]_{ij}$ is the Hessian matrix and $\mathbf{H}_k$ is the kth row. Since the computation of the Hessian matrix can be expensive due to a high number of model parameters, we use the first-order approximation with the following form for the computation of the pruning importance of each weight group $\mathcal{G}_m^{(p)}$:

$$\mathcal{I}_m^{(p)} = \sum_{\theta_k \in \mathcal{G}_m^{(p)}} \left(\frac{\partial \mathcal{L}(\mathcal{D}_s)}{\partial \theta_k}\theta_k\right)^2, \tag{1}$$

where we sum the importance of each weight within the group. Other pruning importance criteria can also be utilized in our framework as well and we investigate the effect of these on the final model performance in our experiments.

After the computation of (1) for every weight group, we keep the weight groups with the highest importance values. We sort the weight groups based on their importance values and prune out the groups until $v_p\%$ of the model parameters remain. The pruned model ends up with the weights $\boldsymbol{\theta}^{(p)} = \{\mathcal{G}_m^{(p)} | \mathbf{M}^{(p)}[m] = 1\}$, where $\mathbf{M}^{(p)}$ holds one for the preserved group indices and zero for the pruned group indices). We iteratively repeat the computation of (1) and the pruning operation for N_p steps until the desired cost reduction condition is satisfied such that $1 - v_p^{N_p} \geq r_p$. Gradually pruning the model provides a better approximation of the importance values and empirically yields higher performance compared to the one-shot approach. Here, the final desired pruning ratio r_p can be set such that the finetuning cost of the pruned model (GPU memory requirement) will be less than the available resources.

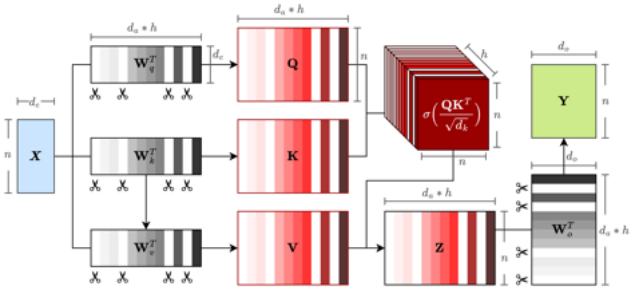

Figure 2. Illustration of the operations at a multi-head attention module with $h = 12$ heads. We group the attention module weights as represented with colors and perform structured pruning at the head level for attention module weights.

3.2. Finetuning Stage

During the finetuning of the pruned model $\boldsymbol{\theta}^{(p)}$ on GPU, we perform masked weight updates, which provides two crucial advantages. First, we can further reduce the GPU memory footprint incurred by the optimizer states and gradients by only storing those for the weights that are being updated. Another motivation for masked updates during finetuning is related to the convergence of the pruned model structure over iterations. Finetuning all weights of the pruned model inherently causes an increase in the importance of these weights compared to the weights that were pruned out. Thus, the iterations may saturate very quickly and cause the pruning of the same weights at each iteration

from the very beginning of the process, which can be sub-optimal as in pre-finetune pruning.

3.2.1 Importance Estimation for Finetuning

Due to the above motivations, instead of finetuning all weights, we only finetune a predefined portion of the pruned model weights. To decide which weight to update, after obtaining the pruned model following the procedure described in Section 3.1, we perform an intermediate step of computation to determine the subset of weights that are most important to update within the pruned model. We would like to note that although this step is similar to the operations in the pruning stage, here instead of determining the weights that have the *most impact on model output*, our goal is finding the weights that would cause the *highest change in model predictions when we update them*.

We measure how much the pruned model output $p_{\theta_k}(y|x)$ changes when we change the weight θ_k to $\theta_k + \delta$, where δ is a small weight perturbation, by approximating the KL divergence $D_{KL}(p_{\theta_k}(y|x)||p_{\theta_k+\delta}(y|x))$ between two output distributions [35]:

$$\mathbb{E}_x D_{KL}(p_{\theta_k}(y|x)||p_{\theta_k+\delta}(y|x)) \approx \delta^2 F_{\theta_k} + O(\delta^3), \quad (2)$$

where F_{θ_k} is the kth diagonal value of the Fisher information matrix and is defined as:

$$F_{\theta_k} = \mathbb{E}_{x\sim p(x)}\left[\mathbb{E}_{y\sim p(y|x)}\left(\frac{\partial \log p_{\theta_k}(y|x)}{\partial \theta_k}\right)^2\right]. \quad (3)$$

As seen from (2) and (3), the Fisher information is related to the impact of weight updates on the model output. To reduce the computation cost, we consider the following approximation for the expectation (also referred as empirical Fisher in recent studies [35]) and define the finetuning importance of each weight group as below:

$$\mathcal{I}_n^{(f)} = \sum_{\theta_k \in \mathcal{G}_n^{(f)}} \left(\frac{\partial \log p_{\theta_k}(\mathcal{D}_s)}{\partial \theta_k}\right)^2, \quad (4)$$

where $\mathcal{G}_n^{(f)}$ is the nth weight group and $\frac{\partial \log p_{\theta_k}(\mathcal{D}_s)}{\partial \theta_k}$ is the mean derivative of the pruned model output with respect to θ_k over $\mathcal{D}_s$. In other words, to minimize the misalignment between optimizing all weights and only a portion of weights of the pruned model, we can update the weights with the highest gradient magnitudes. During finetuning, each weight group $\mathcal{G}_n^{(f)}$ contains the coupled weights corresponding to the same hidden dimension.

Then, as in 3.1, we determine which weight groups to finetune by sorting the weight groups based on their finetuning importance values. We finetune $v_f\%$ of the pruned model parameters belonging to the groups with the highest importance values. Let $\mathbf{M}^{(f)}$ denote the finetuning mask such that it holds one for the group indices that will be updated and zero for the rest. While loading the pruned model at GPU, we also load this finetuning mask. Next, we start the finetuning procedure on GPU over the pruned model with sparsified updates for N_f steps. During finetuning, we only load the optimizer states and store the gradients for the subset of weights $\boldsymbol{\theta}^{(f)} = \{\mathcal{G}_n^{(f)} | \mathbf{M}^{(f)}[n] = 1\}$.

3.3. Updating Stage

After each finetuning stage, we transfer the updates to the CPU and overwrite the full model weights that have been updated. In addition, we update the optimizer states corresponding to the whole weights since re-initializing optimizer states at the beginning of each iteration and not carrying over the momentum values to the next iteration for the weights that are being updated can cause slower convergence. This also enables us to prevent any instability that might occur in case certain weights have not been updated for a number of past iterations start to be updated. For instance, for SGD with a momentum value of β, we have the following update step at the end of each finetuning stage:

$$V_k \leftarrow \begin{cases} V_k & \text{if } \theta_k \in \cup_{\mathcal{G}\in\boldsymbol{\theta}^{(f)}}\mathcal{G} \\ \beta V_k & \text{ow,} \end{cases} \quad (5)$$

$$\theta_k \leftarrow \begin{cases} \theta_k + \Delta\theta_k & \text{if } \theta_k \in \cup_{\mathcal{G}\in\boldsymbol{\theta}^{(f)}}\mathcal{G} \\ \theta_k & \text{ow,} \end{cases} \quad (6)$$

where $\Delta\theta_k$ is the weight update for θ_k computed in the finetuning stage. We would like to note that (5) can be modified depending on the preferred optimizer type. We repeat the pruning, finetuning and updating stages for K iterations or until the validation performance converges.

4. Experiments

In this section, we report the results of experiments on four vision benchmarks: CIFAR100 [22], TinyImageNet for image classification and Cityscapes [7], KITTI [1] for semantic segmentation, and six natural language understanding tasks from the GLUE benchmark. We show that RECAP effectively balances the tradeoff between final model accuracy after pruning and the GPU memory footprint of the finetuning process. We also analyze the impact of masking on the performance and memory footprint of our system. We provide experimental setup details (datasets, preprocessing, implementation and measurements) with further analysis in the Appendix. Our code is available at: https://github.com/git-disl/recap.

4.1. Accuracy vs Memory-Efficiency Analysis

In this subsection, we report the performance of RECAP by analyzing the relation between the finetuned model performance, and the GPU memory footprint of the process.

Task: *Image Class.*	ViT-base			ViT-large			ViT-huge	
	C100	**TinyIN**	*Mem.*	**C100**	**TinyIN**	*Mem.*	**C100**	*Mem.*
Full-FT	91.84	89.73	1200	94.12	93.02	3854	94.55	7984
Head-FT	82.75	77.87	414	87.34	86.47	1339	89.66	2708
LoRA-FT	86.43	83.79	480	90.81	89.16	1564	92.49	2840
Pre-FT Pruning	86.57	81.41	825	91.65	89.13	2657	92.10	5377
Post-FT Pruning	88.93	85.52	1200	92.41	91.10	3854	93.15	7984
Movement Pruning	88.40	85.06	1216	92.08	90.90	3890	92.90	8010
RECAP (Ours)	88.34	83.83	431	91.93	89.95	1251	92.78	2483

Table 1. Image classification results in terms of accuracy (%) with GPU memory footprint (MB) for ViT-base (86M), ViT-large (307M) and ViT-huge (632M) at CIFAR100 and TinyImageNet with various fine-tuning techniques.

Task: *Semantic Seg.*	Mask2Former w/ Swin-base			Mask2Former w/ Swin-large		
	Cityscapes	**KITTI**	*Memory*	**Cityscapes**	**KITTI**	*Memory*
Full-FT	81.61	71.26	3924	82.69	71.45	6092
Head-FT	76.95	68.27	1648	77.00	68.44	2324
LoRA-FT	77.93	69.85	1815	79.15	70.20	2640
Pre-FT Pruning	77.13	68.61	3079	77.75	69.80	4712
Post-FT Pruning	79.10	70.78	3924	79.48	71.14	6092
Movement Pruning	78.77	70.77	3963	79.66	71.20	6156
RECAP (Ours)	78.33	70.82	2577	79.50	70.98	3647

Table 2. Semantic segmentation results in terms of mIoU (%) with GPU memory footprint (MB) for Mask2Former models with backbones Swin-base/Swin-large and evaluated at Cityscapes-dev and KITTI-dev sets with various fine-tuning techniques.

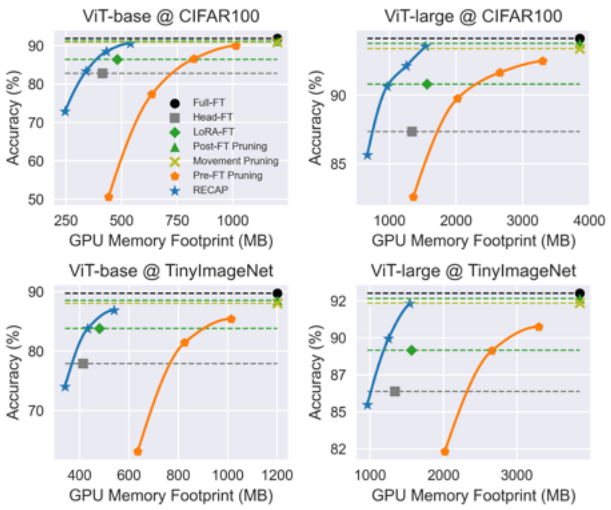

Figure 3. Accuracy vs GPU memory footprint for ViT-base and ViT-large. In the top-left direction, accuracy increases while memory footprint decreases, and RECAP either outperforms other techniques at the same memory footprint or performs on par with significantly lower memory usage. We also plot the results of full finetuning (Full-FT) and head finetuning with frozen backbone (Head-FT). Post-finetune pruning and movement pruning have the same peak GPU memory usage as full finetuning.

To this end, we first provide the results for image classification with ViT variants [11] and semantic segmentation experiments with Mask2Former (M2F) [6] in Tables 1 and 2. Here, we compare various finetuning techniques. *Full-FT* denotes the standard finetuning procedure of the full model, where no efficiency technique has been utilized. In *Head-FT*, we freeze the pre-trained model weights and only update the parameters of the injected task head to reduce the GPU memory footprint caused by activations and optimizer states. In *LORA-FT*, we again freeze the pre-trained model weights, but in addition to the task head, we also inject and finetune adapter modules, specifically following the LoRA [20] methodology. We also compare RECAP with three representative pruning techniques for the finetuning of PLMs. In *Pre-FT* (pre-finetune pruning), which we consider as our baseline, we finetune the model after performing the pruning operation. In *Post-FT* (post-finetune pruning), we finetune the model before and after pruning, following [28]. Lastly, we also report the results obtained with *Movement* [34] pruning, which integrates the pruning process into finetuning through joint optimization. For all pruning techniques, including RECAP, we report the results for $r_p = 33.3\%$.

From the results in Tables 1 and 2, we make three observations. (1) RECAP consistently outperforms Pre-FT pruning with higher accuracy and lower memory footprint as expected. (2) RECAP achieves competitive accuracy results with Post-FT pruning and Movement pruning at a significantly lower memory footprint as these two techniques require GPU operations on the full model, resulting in a high GPU memory footprint. In comparison, RECAP only requires the pruned model on GPU and updates a portion of those weights at each iteration, and thus, effectively reduces the GPU memory footprint by around 64% and 68% for ViT-base and ViT-large respectively, and by around 34% and 40% for M2F w/ Swin-base and Swin-large respectively. (3) For M2F on Cityscapes and KITTI in Table 2, the performance gap between RECAP and Post-FT and Movement pruning techniques is almost zero.

In Table 2, the higher cost with RECAP for M2F is due to the temporarily stored activations for backpropagation. In M2F, we observe that the cost of activations is dominant (57.4% of total cost whereas 14.2% in ViT-b and 6.4% in ViT-l) due to high-resolution activations. Finetuning with adapters does not require computing the derivative of loss w.r.t attention weights and so, bypassing the storage for some activations or modifying some gradients in-place during backpropagation is possible. So, although RECAP reduces the cost of activations thanks to pruned heads and FF layers, adapters are more efficient in this scenario but with 0.4-1% lower mIoU. We also provide Tables 3 and 4 for the results in image classification experiments at different pruning ratios to observe how the behavior of these techniques changes at different constraint regimes.

	Model	Method	Pruning Ratio (r_p)			
			16.6%	33.3%	50.0%	66.6%
CIFAR100	**ViT-base (86M)** *Full-FT: 91.84%* *Head-FT: 82.75%*	**Pre-FT**	89.98	86.57	77.29	50.62
		Post-FT	91.20	88.93	84.95	75.03
		Movement	90.92	88.40	84.85	74.83
		RECAP (Ours)	90.50	88.34	83.33	72.90
	ViT-large (307M) *Full-FT: 94.12%* *Head-FT: 87.34%*	**Pre-FT**	92.49	91.65	89.75	82.60
		Post-FT	93.76	92.41	91.29	87.76
		Movement	93.39	92.08	91.10	87.33
		RECAP (Ours)	93.55	91.93	90.63	85.65

Table 3. The results for ViT-base and ViT-large evaluated at CIFAR100 with various techniques at different pruning ratios.

	Model	Method	Pruning Ratio (r_p)		
			16.6%	33.3%	50.0%
TinyImageNet	**ViT-base (86M)** *Full-FT: 89.73%* *Head-FT: 77.87%*	**Pre-FT**	85.40	81.41	63.11
		Post-FT	88.52	85.52	78.18
		Movement	88.03	85.06	78.14
		RECAP (Ours)	86.93	83.83	73.96
	ViT-large (307M) *Full-FT: 93.02%* *Head-FT: 86.37%*	**Pre-FT**	90.75	89.13	72.32
		Post-FT	92.66	91.10	77.63
		Movement	92.36	90.90	77.45
		RECAP (Ours)	92.32	89.95	75.47

Table 4. The results for ViT-base and ViT-large evaluated at TinyImageNet with various techniques at different pruning ratios.

To illustrate how these pruning ratios translate to memory footprint, we plot the results in Figure 3, showing that in every pruning regime, RECAP provides the best tradeoff between accuracy and memory efficiency. Due to structured pruning, we observe a significant performance drop with 66% or more pruning ratio in all techniques. But we can achieve 3-4x less memory usage with 33% pruning ratio, without losing significant performance (e.g, -2.19%/-3.08% at CIFAR100/TinyImageNet for ViT-large). Lastly, we illustrate some samples from Cityscapes-dev in Figures 6 and 7 to visually compare the model outputs after pruning with various techniques. We observe that the model after Pre-FT pruning tends to lose details for small objects (people, signs etc.) whereas model outputs after RECAP and Post-FT pruning are very similar.

We perform similar analysis on natural language understanding (NLU) tasks and report the results obtained on the GLUE benchmark in Figure 4. GLUE benchmark has nine NLU tasks, and due to space constraints, we include the results of six tasks, which we list in Figure 5 with descriptions. We also report the results with memory footprints in Table 5 for $r_p = 50\%$. We observe that RECAP can achieve the performance of Post-FT pruning in most settings. For instance, in CoLA, MRPC and SST-2, RECAP achieves very similar results with Post-FT pruning until the 50% pruning ratio. Yet, RECAP consumes a notably lower memory footprint with 45% of Post-FT pruning and 73% of Pre-FT pruning.

4.2. Analysis on the Impact of Masking

In this subsection, we analyze the impact of the masking of updates during the finetuning stage. To this end, we report the results obtained with ViT-base on CIFAR100 and TinyImageNet datasets and with BERT-base on COLA-dev from GLUE at different pruning and masking ratios. We plot the obtained results in Figure 8. We observe that the performance worsens when the masking ratio is zero/too low or too high. As we discuss in Section 3.2, without masking, the process saturates earlier, resulting in suboptimal performance similar to that in pre-finetune pruning. On

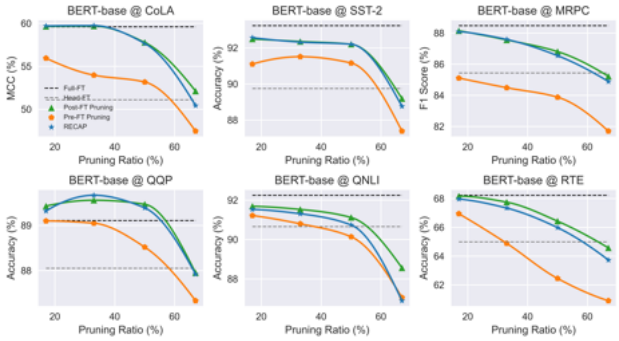

Figure 4. Performance at various pruning ratios for BERT-base on six datasets from GLUE benchmark.

Dataset	Task Description
CoLA	Is the sentence grammatical?
SST-2	Is the movie review positive?
MRPC	Is the sentence B a paraphrase of sentence A?
QQP	Are the two questions similar?
QNLI	Does sentence B contain the answer to the question in sentence A?
RTE	Does sentence A entail sentence B?

Figure 5. Six of the natural language understanding tasks in GLUE benchmark and their task descriptions.

Task: NLU	CoLA	SST-2	MRPC	QQP	QNLI	RTE	*Memory*
Full-FT	59.61	93.23	88.48	89.11	92.25	68.23	1553
Head-FT	51.10	89.75	85.44	88.05	90.66	64.99	524
Pre-FT	53.17	91.16	83.88	89.47	90.14	62.45	957
Post-FT	57.78	92.20	86.81	88.52	91.12	66.43	1553
RECAP (Ours)	57.67	92.20	86.55	89.39	90.74	65.98	702

Table 5. Natural language understanding results in terms of accuracy (%) with GPU memory footprint (MB) for BERT-base (For CoLA and MRPC, we report Matthews Correlation Coefficient (MCC) and F1 Score, respectively.

the other hand, masking too many weights hurts the convergence so we empirically set the masking ratio, which is $r_f = 87.5\%$ for vision experiments and $r_f = 50\%$ for text experiments. We provide further analysis on the conver-

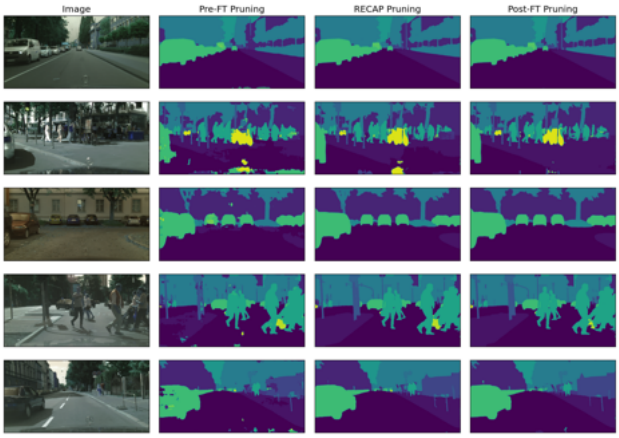

Figure 6. Visual comparison of the Mask2Former outputs for randomly selected samples from Cityscapes after pruning with Pre-FT, RECAP and Post-FT pruning at 33% pruning rate.

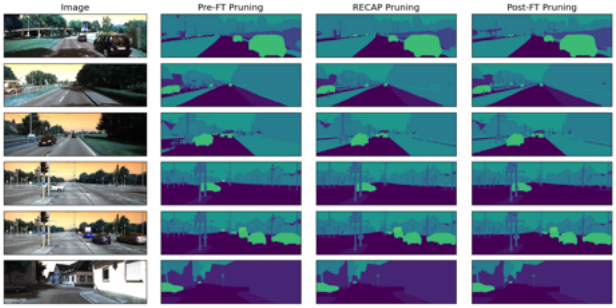

Figure 7. Visual comparison of the Mask2Former outputs for randomly selected samples from KITTI after pruning with Pre-FT, RECAP and Post-FT pruning at 33% pruning rate.

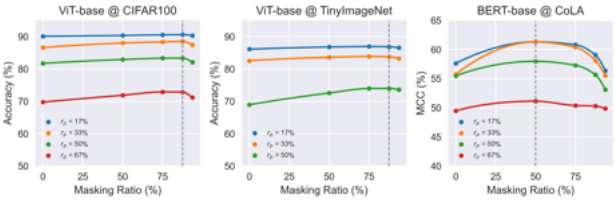

Figure 8. Performance at various finetuning mask ratios. Selected masking rates are shown with vertical dashed lines.

gence of the pruned model structure in the Appendix.

4.3. Memory Footprint Breakdown

In Figure 9, we report the memory footprint due to each component (weights, gradients, activations, optimizer states) for $r_f = 33\%$, $r_f = 87.5\%$, batch size of one, and the process time for ten epochs. RECAP reduces the memory footprint of all components, attributing to only operating on the pruned model at GPU, and also further reducing the cost of optimizer states and gradients with masked finetuning. CPU time contains the time consumed during pruning and masking.

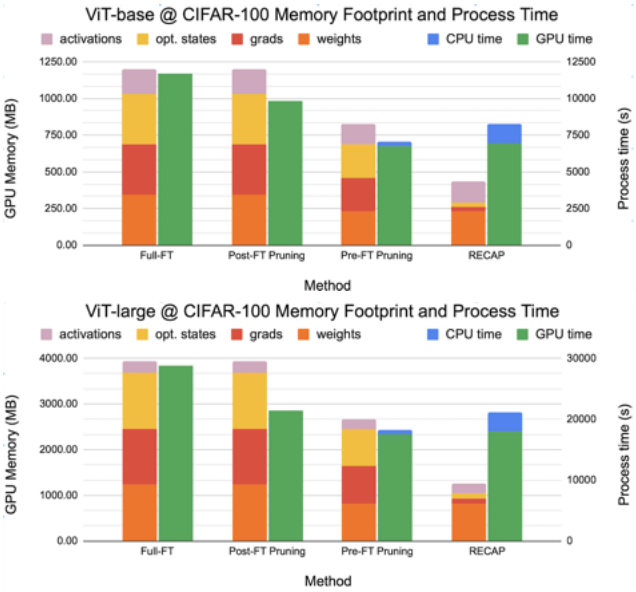

Figure 9. Breakdown of GPU memory footprint (left bars) and process time (right bars) for ViT-base and ViT-large at CIFAR100.

5. Conclusion

We have presented RECAP, a pruning framework and a suite of optimization techniques, for memory-efficient finetuning of transformer-based large DNN models. This paper makes two original contributions. First, we develop a novel three-stage approach for integrating pruning and finetuning through an iterative process cycling between pruning, finetuning, and updating stages, allowing RECAP to explore different chunks of the given large-scale model. Second, unlike existing pruning methods that demand full model finetuning at GPU, our approach effectively reduces the memory footprints of model weights, gradients, activations, and loss optimizer states. Extensive experiments with large transformer-based architectures on four vision benchmarks and six NLU tasks on the GLUE benchmark demonstrate that RECAP offers significant improvements in GPU memory efficiency, reducing the footprint by up to 65%, while maintaining competitive performance.

Acknowledgement

This research is partially sponsored by the NSF CISE grants 2302720, 2312758, and 2038029, and an IBM faculty award. The first author thanks for the summer 2023 internship at IBM Research with the group led by Dr. Donna Dillenberger.

References

[1] Hassan Alhaija, Siva Mustikovela, Lars Mescheder, Andreas Geiger, and Carsten Rother. Augmented reality meets computer vision: Efficient data generation for urban driving scenes. *International Journal of Computer Vision (IJCV)*, 2018. 5, 2

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, pages 1877–1901. Curran Associates, Inc., 2020. 1

[3] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jegou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9630–9640, 2021. 1

[4] Zhe Chen, Yuchen Duan, Wenhai Wang, Junjun He, Tong Lu, Jifeng Dai, and Yu Qiao. Vision transformer adapter for dense predictions. In *The Eleventh International Conference on Learning Representations*, 2023. 2

[5] Bowen Cheng, Alexander G. Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation. In *Neural Information Processing Systems*, 2021. 1

[6] Bowen Cheng, Ishan Misra, Alexander G. Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1280–1289, 2022. 1, 6

[7] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 5, 2

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 2

[9] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*, 2022. 2, 3

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019. 1

[11] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. 1, 6, 2

[12] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. 1, 2

[13] Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot, 2023. 2

[14] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *ECCV*, 2019. 2

[15] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural network. In *Neural Information Processing Systems*, 2015. 4

[16] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. 1, 2

[17] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 2

[18] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1398–1406, 2017. 4

[19] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, 2019. 2

[20] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *ICLR*, 2022. 2, 6

[21] Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning. In *ECCV*, 2022. 2

[22] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 5, 2

[23] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY. In *International Conference on Learning Representations*, 2019. 2, 4

[24] Yang Lin, Tianyu Zhang, Peiqin Sun, Zheng Li, and Shuchang Zhou. Fq-vit: Post-training quantization for fully

quantized vision transformer. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 1173–1179, 2022. 2

[25] Zhenhua Liu, Yunhe Wang, Kai Han, Siwei Ma, and Wen Gao. Post-training quantization for vision transformer. In *Neural Information Processing Systems*, 2021. 2

[26] Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. *ArXiv*, abs/2106.04489, 2021. 2

[27] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. 2, 4

[28] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz. Importance estimation for neural network pruning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11256–11264, Los Alamitos, CA, USA, 2019. IEEE Computer Society. 2, 6

[29] Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*, 2019. 1

[30] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2023. 1

[31] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. 1

[32] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, 2021. 1

[33] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. 1

[34] Victor Sanh, Thomas Wolf, and Alexander M. Rush. Movement pruning: Adaptive sparsity by fine-tuning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2020. Curran Associates Inc. 2, 6

[35] Yi-Lin Sung, Varun Nair, and Colin Raffel. Training neural networks with fixed sparse masks. *ArXiv*, abs/2111.09839, 2021. 5

[36] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023. 1

[37] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, 2018. Association for Computational Linguistics. 2

[38] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi 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 M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, 2020. Association for Computational Linguistics. 2

[39] Nakyeong Yang, Yunah Jang, Hwanhee Lee, Seohyeong Jeong, and Kyomin Jung. Task-specific compression for multi-task language models using attribution-based pruning. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 594–604, Dubrovnik, Croatia, 2023. Association for Computational Linguistics. 2, 4

[40] Yihua Zhang, Yuguang Yao, Parikshit Ram, Pu Zhao, Tianlong Chen, Mingyi Hong, Yanzhi Wang, and Sijia Liu. Advancing model pruning via bi-level optimization. In *Advances in Neural Information Processing Systems*, 2022. 2