APT: Adaptive Pruning and Tuning Pretrained Language Models for Efficient Training and Inference

Bowen Zhao [1] Hannaneh Hajishirzi [1 2] Qingqing Cao* [3]

Abstract

Fine-tuning and inference with large Language Models (LM) are generally known to be expensive. Parameter-efficient fine-tuning over pretrained LMs reduces training memory by updating a small number of LM parameters but does not improve inference efficiency. Structured pruning improves LM inference efficiency by removing consistent parameter blocks, yet often increases training memory and time. To improve both training and inference efficiency, we introduce APT that adaptively *prunes* and *tunes* parameters for the LMs. At the early stage of fine-tuning, APT dynamically adds *salient* tuning parameters for fast and accurate convergence while discarding unimportant parameters for efficiency. Compared to baselines, our experiments show that APT maintains up to 98% task performance when pruning 60% of the parameters in RoBERTa and T5 models. APT also preserves 86.4% of LLaMA models' performance with 70% parameters remaining. Furthermore, APT speeds up LMs' fine-tuning by up to 8× and reduces large LMs' memory training footprint by up to 70%. Our code and models are publicly available at https://github.com/ROIM1998/APT.

1. Introduction

Fine-tuning language models (LMs) (Devlin et al., 2019; Liu et al., 2019; Raffel et al., 2020) is an essential paradigm to adapt them to downstream tasks (Mishra et al., 2022; Wang et al., 2022b). Increasing the parameter scale of LMs improves model performance (Kaplan et al., 2020), but incurs significant training and inference costs. For instance, a 13B LLaMA model (Touvron et al., 2023) costs about 100GB memory for fine-tuning and 30GB for inference with float16 datatype. It is important to improve the training and inference efficiency of LM for practical applications.

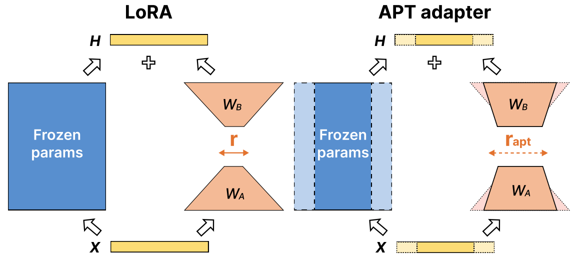


Figure 1. APT provides both training and inference efficiency benefits by pruning and tuning pretrained LM parameters adaptively via the **APT adapter**. We dynamically adjust (add/reduce) APT adapter input/output dimensions and the rank (r_{apt}). Reducing adapter dimensions prunes frozen parameters, making training and inference faster and more memory-efficient. Adding adapter ranks helps recover the pruned LM's task performance. In contrast, existing adapters like LoRA allow efficient training but do not provide inference efficiency since the model size is not reduced.

Parameter-efficient fine-tuning methods (PEFT, summarized in Table 1) (Houlsby et al., 2019; Li & Liang, 2021) reduce the memory consumption of LM fine-tuning via updating a small number of parameters. However, PEFT models do not improve inference efficiency because the LM size remains the same or even increases after fine-tuning. For instance, LoRA (Hu et al., 2022) tunes low-rank decomposed linear layers parallel to frozen parameters to reduce training memory but takes longer to converge (Ding et al., 2023). On the other hand, structured pruning (Kwon et al., 2022; Xia et al., 2022; Ma et al., 2023) improves inference efficiency by removing blocks of parameters such as attention heads and feed-forward neurons in Transformer LMs, showing more inference speedup than sparse unstructured pruning methods (Han et al., 2016; 2015; Sanh et al., 2020). However, training pruned LMs takes extra time to converge and incurs high memory, substantially diminishing LMs' accessibility in usage scenarios with limited computational resources.

Integrating structured pruning and PEFT could increase both training and inference efficiency. However, existing research (Zhao et al., 2023) indicates that combining PEFT

[1] University of Washington [2] Allen Institute for Artificial Intelligence [3]* Apple, work done at the University of Washington. Correspondence to: Bowen Zhao <bowen98@uw.edu>, Qingqing Cao <qicao@apple.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

Method		$\mathcal{A}_{\mathrm{P}}$	$\mathcal{A}_{\mathrm{T}}$	Training		Inference	
				T	M	T	M
PEFT	Adapter (Pfeiffer et al., 2021)	✗	✗	$\Uparrow$High	$\Downarrow$Low	$\Uparrow$Low	$\Uparrow$Low
	LoRA (Hu et al., 2022)	✗	✗	$\Uparrow$High	$\Downarrow$Low	=	=
	AdaLoRA (Zhang et al., 2023b)	✗	✓	$\Uparrow$High	$\Downarrow$Low	=	=
Pruning	MvP (Sanh et al., 2020)	✗	✗	$\Uparrow$High	$\Uparrow$Low	$\Downarrow$Low	$\Downarrow$Low
	BMP (Lagunas et al., 2021)	✗	✗	$\Uparrow$High	$\Uparrow$Low	$\Downarrow$High	$\Downarrow$Low
	CoFi (Xia et al., 2022)	✗	✗	$\Uparrow$High	$\Uparrow$Low	$\Downarrow$High	$\Downarrow$Low
	MT (Kwon et al., 2022)	✗	✗	=	=	$\Downarrow$High	$\Downarrow$Low
Combined	SPA (Hedegaard et al., 2022)	✗	✗	$\Uparrow$High	$\Uparrow$Low	$\Downarrow$High	$\Downarrow$Low
	LRP (Zhang et al., 2023a)	✗	✗	$\Uparrow$High	$\Downarrow$Low	$\Downarrow$High	$\Downarrow$Low
	APT (ours)	✓	✓	$\Uparrow$Low	$\Downarrow$Low	$\Downarrow$High	$\Downarrow$Low

Table 1. Efficiency comparison of existing methods and APT. $\mathcal{A}_{\mathrm{P}}$ stands for adaptive pruning and $\mathcal{A}_{\mathrm{T}}$ for adaptive tuning, where the total and tuning parameter sizes are dynamically adjusted. We measure efficiency using training converge time, inference time (T), and peak memory (M). Symbols $\Uparrow$ and $\Downarrow$ indicate increased and decreased costs, respectively, while = signifies no change in cost. The terms "low" and "high" qualify the extent of cost variations.

and structured pruning, such as applying structured pruning over LoRA-tuned models, causes noticeable performance loss and extra training costs. It remains challenging to prune LMs accurately using limited training resources.

In this paper, we develop an efficient fine-tuning approach named APT that **A**daptively selects model parameters for **P**runing and fine-**T**uning. APT combines the benefits of PEFT and structured pruning to make fine-tuning and inference more efficient. Our intuition is that pre-trained LM parameters contain general knowledge, but their importance to downstream tasks varies. Therefore, we can remove the parameters irrelevant to the fine-tuning task in the early training stage. Early-removing these parameters improves training and inference efficiency while not substantially hurting model accuracy (Frankle et al., 2021; Shen et al., 2022a; Zhang et al., 2023c). Meanwhile, continuously adding more parameters for fine-tuning can improve LM performance because task-specific skills live in a subset of LM parameters (Wang et al., 2022a; Panigrahi et al., 2023).

More specifically, APT learns the pruning masks via an outlier-aware salience scoring function to remove irrelevant LM parameter blocks and adds more tuning parameters during fine-tuning according to tuning layer importance. To make training more efficient, the salience scoring function is lightweight and causes little runtime and memory overhead. Combined with our self-distillation technique that shares teacher and student parameters, APT can accurately prune an LM with less training time and lower memory usage.

Experimental results show that APT prunes RoBERTa and T5 base models 8× faster than the LoRA plus pruning baseline while reaching 98.0% performance with 2.4 × speedup and 78.1% memory consumption during inference. When pruning large LMs like LLaMA, APT costs only 30% memory compared to the state-of-the-art pruning method and still maintains 86.4% performance with 70% parameters. Our ablation study in Section 5.6 indicates the effectiveness of adaptive pruning and tuning. It also demonstrates that efficient distillation with APT adapter substantially recovers small LMs' performance while outlier-aware salience scoring prunes large LMs more accurately. Our analysis in Appendix H demonstrates that controlled adaptive tuning with early pruning during fine-tuning improves LM end-task accuracy better with less training time and memory costs.

2. Related Works

2.1. Parameter-efficient Fine-tuning (PEFT)

PEFT methods aim to tune LMs with limited resources by updating a small number of parameters (Lialin et al., 2023), mainly falling into three categories: selective, additive, and dynamic. Selective methods focus on tuning a subset of parameters in LMs with pre-defined rules (Ben Zaken et al., 2022) or importance metrics (Sung et al., 2021; Guo et al., 2021). Additive methods tune injected layer modules (Houlsby et al., 2019; Pfeiffer et al., 2021) or embeddings (Lester et al., 2021; Li & Liang, 2021). For example, LoRA (Hu et al., 2022) tunes low-rank decomposed layers to avoid inference cost overhead. However, LoRA keeps the tuning layer shapes static without dynamic adjustments. Dynamic methods (He et al., 2022b) adjust tuning parameters during training. For instance, AdaLoRA (Zhang et al., 2023b) gradually reduces tuning parameters but does not benefit inference efficiency. Compared to these methods, APT adaptively adjusts the pruning and tuning parameters simultaneously, improving training and inference efficiency.

2.2. Model Compression

Model compression methods like quantization and pruning boost inference efficiency. Quantization aims to reduce LMs' memory consumption via converting parameters to low-bit data types (Frantar et al., 2023; Dettmers et al., 2022; Lin et al., 2023). However, despite reducing LM's memory consumption, the speedup benefits of quantization require specific framework support, which limits their adaptability. Pruning (LeCun et al., 1989; Han et al., 2016; Frankle & Carbin, 2019; Xu et al., 2021) aims to discard unimportant parameters in LMs for inference efficiency. Unstructured pruning (Sanh et al., 2020) prunes sparse parameters in LMs, which requires dedicated hardware support for efficiency improvements. Meanwhile, structured pruning (Lagunas et al., 2021; Xia et al., 2022) prunes consistent blocks in transformer layers (MHA heads, FFN neurons, and model dimensions) for ubiquitous inference efficiency gains. Such pruning often uses knowledge distillation (Hinton et al., 2015), which causes more training costs. Post-training pruning (Kwon et al., 2022; Frantar & Alistarh, 2023) aims to prune fine-tuned models with limited extra costs but requires initialization from fully fine-tuned models. Moreover, task-agnostic pruning (Sun et al., 2023; Ma et al., 2023) cannot achieve on-par performance with task-specific pruning.

2.3. Combining Compression and PEFT

Combining model compression and PEFT might achieve both training and inference efficiency improvements: QLoRA (Dettmers et al., 2023) and QA-LoRA (Xu et al., 2023) bring quantization and LoRA together for large LM tuning. SPA (Hedegaard et al., 2022) combines structured pruning and Compacter (Mahabadi et al., 2021), yet suffers substantial performance loss. CPET (Zhao et al., 2023) leverages different task-agnostic model compression methods together with LoRA and knowledge distillation, but the performance loss becomes notable specifically when structured pruning is applied. PST (Li et al., 2022) and LRP (Zhang et al., 2023a) also explored the combination of LoRA and pruning, yet their performance degradations are also substantial because their tuning parameters are static. In contrast, APT identifies tuning and pruning parameters based on their salience in fine-tuning, which can improve training and inference efficiency under a new paradigm with minimal performance loss.

3. Problem Formulation

Our goal is to improve the training and inference efficiency of pretrained LM while maintaining task performance. Intuitively, tuning fewer parameters leads to smaller training memory footprints and shorter time per training step; models with fewer parameters also run faster with less memory footprint during inference but come with task performance degradation. We aim to find the optimal parameters for training and inference without sacrificing task performance.

We formally define the problem objective as minimizing the task loss $\mathcal{L}$ under the constraint that the total LM parameter size Θ reaches a target sparsity (defined as the ratio of the number of parameters pruned to the original LM) γ_T after T training steps. For each training step t, the sparsity of the LM remains above γ_t while the number of tuning parameters is below Δ_t. We control the pruning masks $\mathcal{M}_t$ and tuning ranks $\mathcal{R}_t$ to satisfy these constraints. We describe the optimization process as:

$$\begin{aligned}
&\underset{\Theta_T,\mathcal{M}_T}{\text{argmin}} && \frac{1}{|\mathcal{D}|}\sum_{x,y\in\mathcal{D}}\mathcal{L}(x,y|\Theta_T,\mathcal{M}_T)\\
&\text{s.t.} && 1-\frac{\mathcal{C}(\Theta_t,\mathcal{M}_t)}{\mathcal{C}(\Theta_0,\mathcal{M}_0)}\geq\gamma_t,\\
& && \delta(\Theta_t,\mathcal{M}_t,\mathcal{R}_t)\leq\Delta_t,\\
& && \forall t\in\{0,1,\ldots,T\}.
\end{aligned} \tag{1}$$

where x, y are inputs and labels sampled from the task dataset $\mathcal{D}$, while $\mathcal{C}$ and δ denotes total and tuning parameter numbers of the LM, respectively.

Based on Equation (1), a higher target sparsity γ_T improves inference efficiency with fewer FLOPs and memory usage but sacrifices performance. Increasing γ_t when $t \ll T$ also improves training efficiency. Besides, tuning more parameters with larger Δ costs more training memory but makes the model converge faster with better task performance. Our formulation supports task performance improvements together with training and inference efficiency by dynamically adjusting the LM parameters during fine-tuning.

4. Adaptive Pruning and Tuning

We design **A**daptive **P**runing and **T**uning (**APT**) over LM parameters to allow efficient training and inference while maintaining task performance.

Summarized in the left of Figure 2, existing pruning methods often neglect training costs where the number of tuning parameters is more than a parameter-efficient threshold with $\Delta_t \geq \mathcal{C}(\Theta_t, \mathcal{M}_t)$, resulting in long training time and high memory consumption. Instead, to improve training efficiency, we prune LM parameters (increase γ_t) during early training when $t \ll T$ while keeping $\Delta_t \ll \mathcal{C}(\Theta_t, \mathcal{M}_t)$ to reduce training costs. In addition, we add tuning parameters (increase Δ_t) in early training to effectively mitigate the degradation of LM's performance due to pruning.

Overview. Figure 2 shows the overview of our method that incorporates our new APT adapter for pruning and tuning. Our intuition is that pruning LMs during early fine-tuning will not hurt their task performance while reducing training and inference costs. Meanwhile, unlike existing adapters like LoRA (Hu et al., 2022) that use fixed tuning parameters, APT adapters dynamically add tuning parameters to accelerate LM convergence with superior task performance. We first introduce the architecture of APT adapters in Section 4.1. We then describe how we prune LM parameters at early fine-tuning with low cost in Section 4.2 and adap-

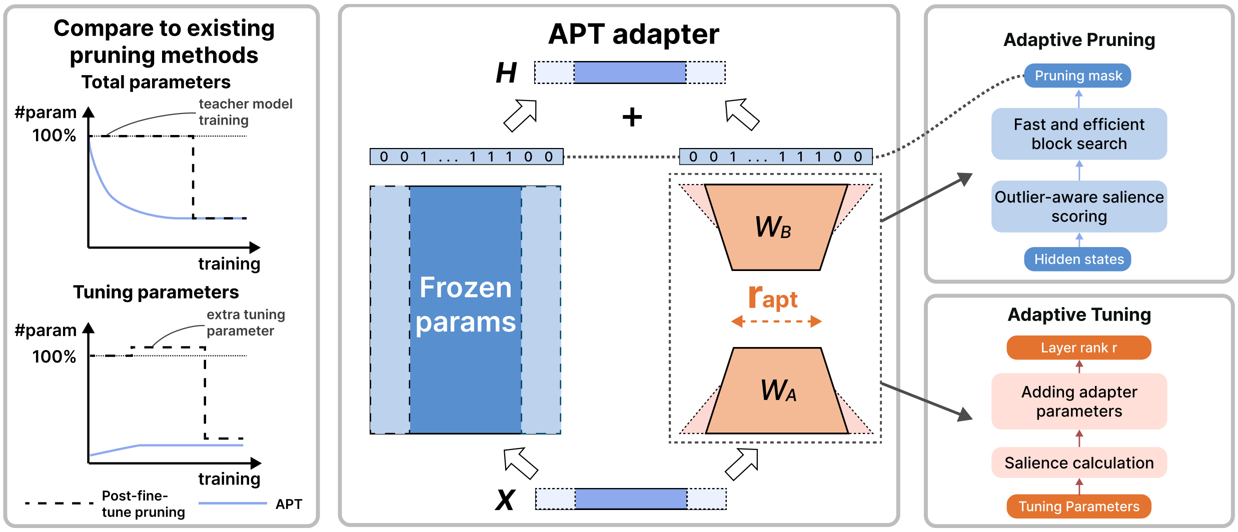


Figure 2. APT adaptively identifies pruning and tuning parameters via APT adapters during fine-tuning with little cost. APT gradually prunes LM parameters with binary pruning masks learned from our lightweight outlier-aware salience scoring function for training and inference efficiency. APT also adds tuning parameters in salient layers in LM fine-tuning through increasing dynamic ranks in APT adapters for performance recovery.

tively tune LMs to recover task performance efficiently in Section 4.3. Additionally, we explain our self-knowledge distillation technique that improves pruned LM's task performance with limited training expense in Section 4.4.

4.1. APT adapter

We build the APT adapter architecture over LoRA, but the key difference is that APT adapter supports dynamic LM pruning and tuning. Assuming an APT adapter projects the input $X \in \mathbb{R}^{d_i}$ to the output $H_{\text{apt}}(X) \in \mathbb{R}^{d_o}$, we design binary pruning masks ($m_i \in \mathbb{R}^{d_i}$ for input and $m_o \in \mathbb{R}^{d_o}$ for output) and dynamic ranks r_{apt} in APT adapter to control the total and tuning LM parameters during fine-tuning, respectively. Specifically, with tuning parameters $W_A \in \mathbb{R}^{r_{\text{apt}} \times d_i}$ and $W_B \in \mathbb{R}^{d_o \times r_{\text{apt}}}$, APT adapter H_{apt} is denoted as:

$$H_{\text{apt}}(X) = m_o \circ (W + s \cdot W_B W_A) X \circ m_i \qquad (2)$$

where s is the constant scaling factor following LoRA's implementation, and $\circ$ denotes the Hadamard product between the masks and their corresponding matrices. The parameter block is pruned when the multiplying mask is set to 0 and retained when set to 1. In the meantime, during fine-tuning, we dynamically increase r_{apt} for the weight matrices W_B and W_A. Compared to LoRA, APT adapters can be more efficient due to more adaptive pruning and tuning over LM parameters.

In transformer-based LM fine-tuning, we add APT adapters in queries and values of multi-head attention (MHA) layers. We also add APT adapter in feed-forward network (FFN) layers when fine-tuning smaller models like RoBERTa and T5 for fast training convergence. In these cases, m_i prunes transformers' hidden dimension and m_o prunes attention heads in MHA and internal neurons in FFN layers. By learning the pruning masks and adjusting the ranks dynamically in the APT adapter, we can achieve the goal defined in Section 3 where the tuning parameter number $\delta(\Theta_t, \mathcal{M}_t, \mathcal{R}_t)$ increases to maintain task performance and the LM parameter size $\mathcal{C}(\Theta_t, \mathcal{M}_t)$ decreases to support more efficient training and inference. Next, we describe the adaptive pruning and tuning procedures in detail.

4.2. Low-cost Adaptive LM Pruning ($\mathcal{A}_\mathbf{P}$)

To benefit the efficiency of LM training and inference, APT adaptively prunes LM parameters since the start of fine-tuning. The problem is finding the parameters to be pruned and discarding them without hurting training stability. Given a task, we compute the outlier-aware salience score of parameter blocks at each early-training step when $t \ll T$. Afterward, we use a fast search algorithm to determine the parameters to be pruned, and then we update their binary pruning masks accordingly. The upper-right of Figure 2 shows this adaptive pruning procedure.

Outlier-aware salience scoring of LM parameters. When determining the influence of pruning parameters on the LM performance for fine-tuning tasks, the key idea is to compute the outlier-aware salience scores of LM activations to consider both tuning and frozen parameters. In detail, salience is defined as the magnitude of parameters' weight-gradient production from previous works (Sanh et al., 2020), where

$$S(W_{i,j}) = |W_{i,j} \cdot \frac{\partial \mathcal{L}}{\partial W_{i,j}}| \qquad (3)$$

However, since the frozen weights' gradients are unreachable in PEFT settings, we compute the salience as the magnitude of the product of activations and their gradients. Additionally, we compress the activation and gradients by summing along batches before production to further reduce the training memory consumption. On the other hand, block outlier parameters play a crucial role in task-specific capabilities, as previous quantization methods suggest (Dettmers et al., 2022; Lin et al., 2023). Such effects brought by outlier parameters will be averaged if salience is only measured on the block level. To keep more outlier parameters in the pruned LMs, we combine the salience score above and the kurtosis[1] of the activation together. Therefore, given the supervised finetuning dataset $\mathcal{D}_t$, the outlier-aware salience score $\hat{S}$ is defined as:

$$\widetilde{S}_t(W_{:,j}) = \sum_{(x,y)\in\mathcal{D}_t} \sum_i \left|\frac{\partial \mathcal{L}(x,y|\Theta_t, \mathcal{M}_t)}{\partial H_{j,i}}\right| \cdot \sum_{(x,y)\in\mathcal{D}_t} \sum_i |H_{j,i}| \tag{4}$$

$$\hat{S}((W_{:,j}) = \widetilde{S}(W_{:,j}) + (\text{Kurt}(O_{j,:}))^{\frac{1}{2}} \tag{5}$$

where H is the activations in the LM, $\text{Kurt}(\cdot)$ stands for kurtosis, and $O_{:,j} = W_{:,j} \circ X^{\intercal}_{j,:}$ represents the activation. We leave details of the salience scoring in Appendix B.

Efficient search of LM block parameters. Given the salience calculated in Equation (5), the next step is to learn the binary pruning masks to increase the LM sparsity above γ_t. Intuitively, we shall prune the blocks with less salience score, which formulates a latency-saliency knapsack (Shen et al., 2022b) task. For an LM with n_L transformer layers, where layer i has n^i_h MHA heads and n^i_f FFN neurons, and all transformer layers' hidden dimension sizes are d_m, the approximated[2] number LM parameter is:

$$\mathcal{C}(\Theta_t; \mathcal{M}_t) \approx d_m \sum_{i=1}^{n_L} (4n^i_h \cdot d_h + 2n^i_f) \tag{6}$$

where d_h is the dimension per MHA head. To keep the constraint in Equation (1), we prune MHA heads, FFN neurons, and the model hidden dimension simultaneously by reducing n^i_h, n^i_f, and d_m. Hence, we first sort the blocks by their salience divided by the parameter number. As the parameter size monotonically increases with block quantity, we use binary search to identify the top salient blocks to be retained given the sparsity constraint γ_t. We leave the implementation details in Appendix C for simplicity.

4.3. Adaptive and Efficient LM Tuning ($\mathcal{A}_\mathbf{T}$)

As using PEFT methods to fine-tune pruned LMs causes notable performance decrease (illustrated in Table 2 and Table 4), we aim to dynamically add tuning parameters in LM fine-tuning to improve the model's end-task performance. However, since more tuning parameters will consume extra training time and memory, we want to add parameters in a controlled way, where new parameters are only added to task-sensitive APT adapters. As a result, we can recover pruned LMs' performance with reasonable training costs. In detail, we first calculate the salience of each APT adapter to determine their importance. Next, we select the top-half APT adapters after sorting them with salience and add their parameters by increasing their r_{apt}.

Salience scoring of APT adapter. Since gradients of tuning parameters information are available when determining the layer salience, we can first calculate each tuning parameter's salience with Equation (3). Then, we define the salience of an APT adapter as the summation of the parameter salience scores in W_B, denoted as $\mathcal{I}(H_{\text{apt}}) = \sum_{i,j} S(W_{Bi,j})$, to represent each tuning APT adapter's importance[3]. Given the calculated $\mathcal{I}(H_{\text{apt}})$ for each APT adapter, we can then decide where to add new tuning parameters to efficiently improve the pruned LM's task accuracy.

Dynamically adding APT adapter parameters to recover task performance. With the importance of APT adapters $\mathcal{I}(H_{\text{apt}})$ calculated, the next step of adaptive tuning is to add tuning parameters by increasing the salient tuning layers' ranks $r_{\text{apt}} \in \mathcal{R}_t$ following budget Δ_t. Therefore, firstly, we sort all tuning layers according to their importance score $\mathcal{I}(H_{\text{apt}})$ and linearly increase the ranks of the top-half salient ones. More specifically, when increasing the tuning parameter from Δ_t to $\Delta_{t'}$, the salient layer's rank is changed from r_{apt} to $r'_{\text{apt}} = \lfloor r_{\text{apt}} \cdot \frac{\Delta_{t'}}{\Delta_t} \rfloor$ where $\lfloor\cdot\rfloor$ denotes the floor operation. For training stability, when adding parameters and converting $W_B \in \mathbb{R}^{d_o \times r_{\text{apt}}}, W_A \in \mathbb{R}^{r_{\text{apt}} \times d_i}$ to $W'_B \in \mathbb{R}^{d_o \times r'_{\text{apt}}}, W'_A \in \mathbb{R}^{r'_{\text{apt}} \times d_i}$, we concatenate random Gaussian initialized parameters $\mathcal{N}(0, \sigma^2)$ in W_A and zeros in W_B same as the LoRA initialization, so the layer's output remains unchanged before and after new parameters added.

4.4. Efficient Self-Knowledge Distillation

As shown in Table 4, training pruned LM without knowledge distillation causes significant end-task performance drops. Therefore, we use knowledge distillation in APT to recover the pruned LM's performance. Still, existing strategies require a fully trained teacher model being put into the GPU with the student during distillation, causing high training time and memory. To avoid extra training costs, we keep

[1] Representing the density of the outlier in a distribution, the more the outliers are, the bigger the kurtosis will be.

[2] We ignore the model's layer norm and bias terms since their sizes are small, and we do not count tuning parameters since they can be fully merged after training.

[3] The salience scores calculated using W_B and W_A are equal, so using either of them will get the same result.

duplicating the tuning student layers as teachers during fine-tuning to reduce total training time. Meanwhile, frozen parameters are shared between the student and teacher model during training to reduce memory consumption. We edit the distillation objective in CoFi (Xia et al., 2022) as

$$\begin{aligned}\mathcal{L} &= \mu\mathcal{L}_{distill} + (1-\mu)\mathcal{L}_{ft} \\ \mathcal{L}_{layer} &= \sum_{i=1}^{\mathcal{T}} \text{MSE}(\text{Tr}(H_s^{\phi(i)}), H_t^i)\end{aligned} \tag{7}$$

where μ is a moving term linearly scales from 0 to 1 during distillation to encourage the pre-pruned model vastly fit to the training data, $\mathcal{L}_{distill}$ is the distillation objective from CoFi, and $\mathcal{L}_{ft}$ is the supervised fine-tuning objective. $\mathcal{T}$ is block-wise randomly sampled teacher layers following (Haidar et al., 2022), $\phi(\cdot)$ is the teacher-student layer-mapping function that matches the teacher layer to its closest, non-pruned student layer. Tr denotes the tunable LoRA layer for layer transformation, initialized as an identical matrix $\mathcal{I}$. More implementation details of our self-distillation technique is introduced in Appendix A.

5. Experiments

To evaluate the training and inference efficiency gains of APT, we compare it with the combined use of PEFT with pruning and distillation baselines. We first describe the natural language understanding and generation tasks targeting different LM backbones, then the setup of baselines and APT. We then report task performance, speed, and memory usage for training and inference costs.

5.1. Tasks

We apply APT to BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), T5(Raffel et al., 2020)[4], and LLaMA (Touvron et al., 2023). For BERT, RoBERTa, and T5 models, we train and evaluate on SST2 and MNLI datasets from the GLUE benchmark (Wang et al., 2019) and report the dev set accuracy. We also train and evaluate RoBERTa$_{\text{base}}$ on SQuAD v2.0 (Rajpurkar et al., 2018) and report the dev set F1 score. For T5 models, we also fine-tune them on CNN/DM (Nallapati et al., 2016) and report the ROUGE 1/2/L scores. Meanwhile, We use the GPT-4 generated Alpaca dataset (Taori et al., 2023) to fine-tune large LLaMA models and evaluate them with the lm-eval-harness package (Gao et al., 2023) on four tasks from the Open LLM Leaderboard, namely 25-shot ARC (Clark et al., 2018), 10-shot HellaSwag (Zellers et al., 2019), 5-shot MMLU (Hendrycks et al., 2021), and zero-shot TruthfulQA (Lin et al., 2022).

5.2. Baselines

We validate the efficiency benefits of APT for both training and inference by comparing with PEFT, pruning, and distillation methods, along with their combinations.

LoRA+Prune: a post-training pruning method over on LoRA-tuned LMs. We use Mask Tuning (Kwon et al., 2022), a state-of-the-art post-training structured pruning method based on fisher information. Due to that post-training pruning performs poorly on high-sparsity settings, we retrain the pruned LM after pruning to recover its performance.

Prune+Distill: knowledge distillation has been proved to be a key technique in recovering pruned LMs' task accuracy. In particular, we use the state-of-the-art pruning plus distillation method called CoFi (Xia et al., 2022) which uses L_0 regularization for pruning plus dynamic layer-wise distillation objectives. We only compare APT to CoFi with RoBERTa models since the training memory usage of CoFi is too high for larger LMs.

LoRA+Prune+Distill: to reduce the training memory consumption in pruning and distillation, a simple baseline is to conduct CoFi pruning and distillation but with LoRA parameters tuned only. More specifically, only the L_0 module and LoRA parameters are tunable under this setting.

LLMPruner (Ma et al., 2023): LLMPruner is the state-of-the-art task-agnostic pruning method on LLaMA that prunes its blocks or channels based on salience metrics while using LoRA for fast performance recovery. We compare APT to LLMPruner with fine-tuning on the same GPT-4 generated Alpaca data for fair comparisons.

We also compare APT to PST (Li et al., 2022) and LRP (Zhang et al., 2023a), which are the state-of-the-art parameter-efficient unstructured and structured pruning methods on BERT model. We leave these results in Appendix D.

5.3. Evaluation Metrics

We evaluate APT and baselines on training and inference efficiency, measured in runtime memory and time consumption as follows:

Training Efficiency Metrics: we report relative training peak memory (Train. Mem.) and relative training speed measured by time to accuracy (TTA[5]) (Coleman et al., 2019) compared to full finetuning. For fair comparisons, we consider the training time of the teacher model plus the student for methods using knowledge distillation.

Inference Efficiency Metrics: we report the inference peak memory (Inf. Mem.) and the relative speedup (Inf. Speed)

[4] For fair comparisons, we use the t5-lm-adapt model, which is only pre-trained on the C4 corpus to make sure the initial LM does not observe downstream tasks in pre-training.

[5] For instance, 97% TTA denotes the time spent reaching 97% of the fully fine-tuned model's performance

Model	Method	MNLI	SST2	SQuAD v2	CNN/DM	Train Time(⇓)	Train Mem(⇓)	Inf Time(⇓)	Inf Mem(⇓)
$\text{RoBERTa}_{\text{base}}$	FT	87.6	94.8	82.9	-	100.0%	100.0%	100.0%	100.0%
	LoRA	87.5	95.1	83.0	-	2137.0%	60.5%	100.0%	100.0%
	LoRA+Prune	84.0	93.0	79.2	-	5128.3%	**60.5%**	**38.0%**	**75.1%**
	Prune+Distill	**87.3**	**94.5**	-	-	1495.3%	168.5%	38.6%	79.2%
	LoRA+Prune+Distill	84.2	91.9	-	-	6534.6%	141.4%	39.4%	82.3%
	APT	86.4	**94.5**	**81.8**	-	**592.1%**	70.1%	41.3%	78.1%
T5_{base}	FT	87.1	95.2	-	42.1/20.3/39.4	100.0%	100.0%	100.0%	100.0%
	LoRA	87.0	95.0	-	38.7/17.2/36.0	255.5%	62.0%	100.0%	100.0%
	LoRA+Prune	80.9	92.3	-	36.7/15.7/33.9	4523.5%	**62.0%**	**47.1%**	**73.4%**
	APT	**87.0**	**95.0**	-	**38.6/17.0/35.8**	**484.7%**	73.9%	74.6%	81.5%

Table 2. RoBERTa and T5 pruning with APT compared to baselines under 60% sparsity. We measure the training and inference efficiency with LMs pruned on the SST2 task. Training speed is measured via 97% accuracy TTA. All efficiency metrics are normalized to FT. ⇓ denotes smaller is better. The best-pruned results are **bold**. Raw efficiency results are reported in Table 11.

Method	ARC	HellaSwag	MMLU	TruthfulQA	Avg.	Train Time(⇓)	Train Mem (⇓)	Inf Time(⇓)	Inf Mem(⇓)
LLaMA 2 7B	53.1	77.7	43.8	39.0	53.4	-	-	-	-
LoRA	55.6	79.3	46.9	49.9	57.9	100.0%	100.0%	100.0%	100.0%
LoRA+Prune	**46.8**	65.2	23.9	46.2	45.5	180.9%	100.0%	115.5%	68.9%
LLMPruner	39.2	67.0	24.9	40.6	42.9	**86.9%**	253.6%	**114.8%**	74.2%
APT	45.4	**71.1**	**36.9**	**46.6**	**50.0**	106.0%	**75.8%**	117.0%	**67.2%**

Table 3. LLaMA 2 7B 30% sparsity pruning results with GPT4-generated Alpaca dataset, evaluated on the Open LLM leaderboard few-shot tasks. Training speed is measured via training time per step. We do not compare to distillation baselines because the training cost of distillation is too large, and we also compare APT to LLMPruner since it is dedicated to large LM pruning. All efficiency metrics are normalized to LoRA. ⇓ denotes smaller is better. The best-pruned results are **bold**. Raw efficiency results are reported in Table 12.

based on throughput (data processed per second) for inference efficiency.

Both training and evaluation are conducted on a single A100 GPU. The inference test batch size is 128 for small models while 32 and 4 for LLaMA 7B and 13B models, respectively. We demonstrate detailed training and evaluation setups/implementations in Appendix A.

5.4. Main Results

Overview We demonstrate the end-task performance of APT comparing to fine-tuning (FT), LoRA-tuning (LoRA), and pruning baselines in Table 2 and Table 3. Overall, up to 99% of fine-tuned LM's task accuracy is maintained when pruning RoBERTa and T5 models leaving 40% parameters, with only about 70% training memory consumption than fine-tuning. When pruning LLaMA2-7B models with 70% parameters remaining, APT recovers 86.4% task performance on average, together with only 75.8% training memory usage than LoRA-tuning. Furthermore, APT also significantly reduces end-task performance and training costs compared to the pruning and distillation baselines. The detailed comparisons are shown as follows.

APT speeds up RoBERTa and T5 training 8× and reduces training memory costs to 30% in LLaMA pruning compared to LoRA+Prune baseline. Shown in Table 2, when pruning RoBERTa models to 60% sparsity, APT converges 8.4× faster than the LoRA+Prune baseline with consuming similar GPU memory. APT also prunes T5 models 8.2× faster than the LoRA+Prune baseline. The reason is that APT adaptively prunes task-irrelevant parameters during training, reducing memory and per-step training time. Adding parameters in salient tuning layers also accelerates LM convergence. Also, APT costs less than 24GB of memory when pruning 30% parameters in LLaMA2-7B models before tuning, which can be easily adapted to the consumer-level GPUs. In contrast, LLM-Pruner costs about 80GB memory when pruning the LLaMA 7B model[6].

APT achieves 2.5%-9.9% higher task performance than the LoRA+Prune baseline with the same pruning sparsities. Presented in Table 2 and Table 3 , when RoBERTa, T5, and LLaMA models, regardless of size, APT consistently reach higher task performance than the LoRA+Prune. With similar inference speedup and memory when pruning RoBERTa models, APT reaches 2.5% more end-task performance on average. When pruning T5 models under the 60% sparsity, the task performance achieved by APT is 5.1% better than the LoRA+Prune baseline. However, the inference efficiency reached by APT (1.3× speedup and 81.5% memory cost) is worse than the LoRA+Prune baseline (2.1×

[6] `https://github.com/horseee/LLM-Pruner/issues/4`

speedup and 73.4% memory cost). This is because APT can adaptively prune more decoder parameters, which are also computationally cheaper than encoder parameters (due to shorter output sequence length) but relatively useless for classification tasks. For LLaMA2-7B model pruning with 70% sparsity, APT outperforms LLMPruner with 16.5% and the LoRA+Prune baseline with 9.9%, where the inference efficiency improvements of APT is slightly better than both LoRA+Prune and LLMPruner baselines.

APT reaches on-par performance with the Prune+Distill baseline given the same pruning sparsity but trains 2.5× faster and costs only 41.6% memory. Compared to the Prune+Distill baseline, APT results in comparable task accuracy (0.9 point drop in MNLI and same in SST2). At the same time, with similar inference efficiency achieved, APT costs only 41.6% training memory and converges 2.5× than the Prune+Distill baseline. This is because of the self-distillation technique in APT where no separated teacher model is required in pruning LMs. Moreover, APT achieves better task performance than the LoRA+Prune+Distill baseline as well, with less training time and memory consumption. These results demonstrate that APT successfully tackles the problem where simply combining PEFT and pruning hurts pruned LM's task accuracy and training efficiency.

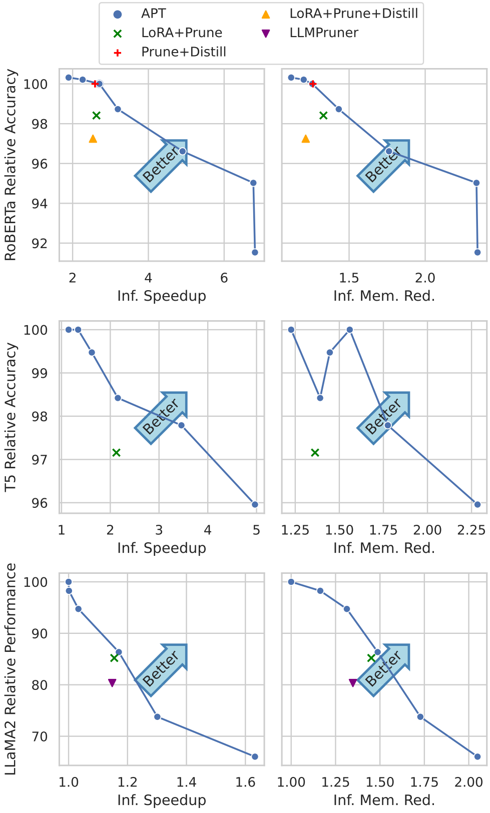


Figure 3. Task performance v.s. relative inference efficiency on RoBERTa, T5, and LLaMA-2 7B models with APT and baselines.

5.5. Pruning Sparsity Analysis

We further show the task performance changing trajectory with different pruning sparsities in Figure 3. APT achieves superior inference speedup with less inference memory consumption than baselines targeting the same task performance. Compared to the LoRA+Prune baseline, when pruning RoBERTa models targeting similar task accuracy, APT is 21.8% faster in inference and is 7% more memory-efficient. For T5 model pruning with 97% of dense model performance, APT results in 62.7% more inference speedup with 24.8% more inference memory reduction compared to the LoRA+Prune baseline. When pruning large LLaMA2-7B models, APT speedup is 6.7% more and reduces 9.2% more inference memory than the LoRA+Prune baseline, maintaining over 85% task performance of the dense model.

5.6. Ablation Study

We evaluate the impact of different components in APT by removing the adaptive pruning ($\mathcal{A}_\mathrm{P}$), adaptive tuning ($\mathcal{A}_\mathrm{T}$), and self-distillation ($\mathcal{D}_\mathrm{S}$). Besides end-task performance, we also report the training efficiency metrics for each ablation.

Adaptive pruning ($\mathcal{A}_\mathrm{P}$) We demonstrate the ablation of adaptive pruning (w/o $\mathcal{A}_\mathrm{P}$) for RoBERTa models in Table 4 and LLaMA models in Table 5. In these cases, we only train LMs with adaptive tuning strategies with supervised fine-tuning objectives without distillation. In such settings, APT w/o $\mathcal{A}_\mathrm{P}$ can be recognized as a PEFT method with tuning parameters' sizes adaptively changing during fine-tuning. Hence, the inference efficiency of the trained LMs are the same as full fine-tuning and LoRA. Without pruning, the task performance of RoBERTa reaches 94.4 for SST2 and 87.5 for MNLI (99.8% fine-tuned LM performance on average). The average performance of the LLaMA model also achieves 96.6% to its LoRA-tuned counterpart. In addition, we surprisingly find that the RoBERTA training speed with APT w/o $\mathcal{A}_\mathrm{P}$ is even 21% faster than full fine-tuning while costing only 62.2% memory. In the meantime, the training memory cost of APT w/o $\mathcal{A}_\mathrm{P}$ in LLaMA tuning is higher than LoRA. The reason is that the tuning parameter number of APT will grow larger than static LoRA-tuning. This ablation demonstrates that adaptive pruning is essential in reducing the training memory consumption of LLaMA model fine-tuning, besides benefiting model inference efficiency.

Adaptive tuning ($\mathcal{A}_\mathrm{T}$) In Table 4, we show results of ablating adaptive tuning (w/o $\mathcal{A}_\mathrm{T}$) where the tuning parameters are static when pruning RoBERTa models. Without $\mathcal{A}_\mathrm{T}$, the model's performance decreases to 93.2/84.4, leading to a similar performance as the LoRA+Prune baseline (93.0/84.0). Moreover, equally increasing parameters across all layers instead of adding parameters based on salience notably hurts the task accuracy (84.4 on MNLI compared to 86.4). At the same time, $\mathcal{A}_\mathrm{T}$ helps the model converge

16% faster than static LoRA training. For ablation results in LLaMA models shown in Table 5, we observe that $\mathcal{A}_{\mathrm{T}}$ recovers the model performance under 50% pruning setting (38.2 compared to 35.8). However, the difference under 70% pruning is insignificant. Meanwhile, if calculating the pruning parameter salience without using kurtosis to consider outliers parameters, the pruned LM's performance substantially drops from 50.0 to 38.1. We conclude that $\mathcal{A}_{\mathrm{T}}$ substantially improves LM training speed and end-task performance. For large LLaMA-based LM pruning, and outlier parameters are essential to recovering the pruned large LLaMA-based models' capabilities.

Method	SST2	MNLI	Train Time($\Downarrow$)	Train Mem($\Downarrow$)
APT	**94.5**	86.4	592.1%	70.1%
w/o $\mathcal{A}_{\mathrm{P}}$	94.4	**87.5**	**82.6%**	62.2%
w/o salience	94.3	84.7	609.8%	65.0%
w/o $\mathcal{A}_{\mathrm{T}}$	93.2	84.5	684.9%	64.4%
w/o $\mathcal{D}_{\mathrm{S}}$	92.9	85.3	483.1%	**61.9%**

Table 4. Results of ablating salience-based allocation strategy and APT adapter with RoBERTa-base model, with relative training efficiency metrics to fine-tuning.

	Sparsity	T.M.	ARC	HellaSwag	MMLU	TruthfulQA	Avg.
APT	30%	75.8%	45.4	71.1	36.9	46.6	50.0
w/o $\mathcal{A}_{\mathrm{P}}$	100%	102.4%	53.8	79.1	46.9	48.4	57.1
w/o kurtosis	30%	75.9%	47.2	39.7	23.0	42.3	38.1
w/o $\mathcal{A}_{\mathrm{T}}$	30%	76.1%	44.2	70.1	40.8	45.1	50.0
APT	50%	60.2%	29.8	48.9	26.7	47.6	38.2
w/o $\mathcal{A}_{\mathrm{T}}$	50%	60.1%	27.9	46.2	24.5	44.7	35.8

Table 5. LLaMA 2 7B model ablation results under 30% and 50% sparsity settings. T.M. denotes relative training memory compare to LoRA-tuning.

Self-distillation ($\mathcal{D}_{\mathrm{S}}$) Shown in Table 4, tuning APT adapters dynamically without distillation objectives gets 1.35 worse task accuracy on average. However, pruning RoBERTa models without self-distillation is 22.5% faster and costs 11.7% less training memory. This result indicates the effectiveness of leveraging knowledge distillation to recover pruned LM performance, but conducting distillation will result in extra training costs regarding both time and memory. Detailed comparisons of self-distillation and traditional, static distillation strategies are shown in Appendix G.

Besides the ablation study results demonstrated above, we show the detailed analysis of adaptive pruning and tuning's effect on LMs' end-task performance, training, and inference efficiency in Appendix H.

6. Limitation and Discussion

Towards better performance gain and inference speedup of large LM in limited resource settings. By comparing Table 2 to Table 3, we notice the performance gap in pruned LLaMA models is larger than smaller LMs because we use distillation-free settings in large LM pruning to reduce training memory consumption. One can improve performance-efficiency trade-offs with better memory-efficient distillation, parameter sharing, and re-allocation strategies. Furthermore, because of the hardware features of Ampere-architecture GPUs, layer dimensions divisible by 8 for FP16 and divisible by 16 for Int8 would reach more realistic speedups. One possible direction is to explore a higher level of structured pruning, for example, grouped neurons and dimensions, in LLMs.

Training could be unstable because of parameter shape changes. Since we adjust tuning parameters dynamically during training, newly initialized parameters are added to the model while existing parameters are pruned. We reset the optimizer every time after each parameter size changes to avoid stability issues, but this strategy might cause unstable training. Meanwhile, the time of selecting the teacher checkpoints during training highly affects the pruned model's performance, whereas non-converged or sparse teachers do not help in performance recovery. The pruned LMs' end-task accuracy could benefit from better and more stable strategies in adaptive pruning and tuning.

Could non-linear adapters perform better for performance recovery? To avoid inference time and memory overhead, we specifically adapt APT adapter to LoRA since the added tuning parameters can be merged after LMs' training. However, low-rank decomposition does not add more complexity to a LM, whereas the model's overall representation capacity doesn't increase. The adaptation with a wider range of adapters, such as Prefix-tuning (Li & Liang, 2021), HAdapters (Houlsby et al., 2019), and Parallel-adapters (He et al., 2022a), could be better explored.

7. Conclusion

We design APT to adaptively identify LMs' pruning and tuning parameters during fine-tuning, improving both training and inference efficiency. APT prunes small LMs faster while pruning large LMs with less memory consumption. With using similar memory costs as LoRA, APT prunes small LMs 8× faster than the LoRA plus pruning baseline. In large LM pruning, APT maintains 87% performance with only 30% pruning memory usage when 70% LM parameter retained. APT opens new directions to pruning LMs in fine-tuning for resource-limited settings, allowing wider usage of LMs in practical applications. In the future, we could adapt APT to more PEFT architectures and target better performance-efficiency trade-offs for billion-level large LMs. Meanwhile, we hope future research will continue to find efficient and accurate techniques to identify salient structures in LMs based on our formulated setting.

Acknowledgements

This research was supported partly by NSF IIS-2044660, an Allen Investigator Distinguished award. We thank the members of the UW NLP group for their comments and feedback on this paper.

Impact Statement

This paper introduces APT, a paradigm for improving the efficiency of training and inference in pre-trained LMs. While our primary goal is to advance machine learning, particularly in the efficiency of LMs and their applications, we recognize potential broader societal impacts. APT significantly reduces training and inference costs and contributes to lower resource consumption for a wide range of applications. This could have a positive environmental impact but might lead to potential model misuse due to lower resource requirements. Additionally, while APT does not introduce new ethical concerns, it might inherit existing issues in language models, for example, biases in training data. We explicitly ask users of APT to be aware of these risks and follow best practices in data selection and model monitoring to mitigate potential harms.

References

Ben Zaken, E., Goldberg, Y., and Ravfogel, S. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1–9, Dublin, Ireland, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.1.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv preprint*, abs/1803.05457, 2018.

Coleman, C., Kang, D., Narayanan, D., Nardi, L., Zhao, T., Zhang, J., Bailis, P., Olukotun, K., Ré, C., and Zaharia, M. Analysis of dawnbench, a time-to-accuracy machine learning performance benchmark. *SIGOPS Oper. Syst. Rev.*, 53(1):14–25, 2019. ISSN 0163-5980. doi: 10.1145/3352020.3352024.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Gpt3.int8(): 8-bit matrix multiplication for transformers at scale. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 30318–30332. Curran Associates, Inc., 2022.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. *ArXiv preprint*, abs/2305.14314, 2023.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. 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, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423.

Ding, N., Qin, Y., Yang, G., Wei, F., Yang, Z., Su, Y., Hu, S., Chen, Y., Chan, C.-M., Chen, W., et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235, 2023.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Frankle, J., Dziugaite, G. K., Roy, D., and Carbin, M. Pruning neural networks at initialization: Why are we missing the mark? In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

Frantar, E. and Alistarh, D. SparseGPT: Massive language models can be accurately pruned in one-shot. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10323–10337. PMLR, 2023.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. OPTQ: Accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*, 2023.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, 2023.

Guo, D., Rush, A., and Kim, Y. Parameter-efficient transfer learning with diff pruning. 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)*, pp. 4884–4896, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.378.

Haidar, M. A., Anchuri, N., Rezagholizadeh, M., Ghaddar, A., Langlais, P., and Poupart, P. RAIL-KD: RAndom intermediate layer mapping for knowledge distillation. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pp. 1389–1400, Seattle, United States, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-naacl.103.

Han, S., Pool, J., Tran, J., and Dally, W. J. Learning both weights and connections for efficient neural network. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 1135–1143, 2015.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In Bengio, Y. and LeCun, Y. (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

He, J., Zhou, C., Ma, X., Berg-Kirkpatrick, T., and Neubig, G. Towards a unified view of parameter-efficient transfer learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022a.

He, S., Ding, L., Dong, D., Zhang, J., and Tao, D. SparseAdapter: An easy approach for improving the parameter-efficiency of adapters. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 2184–2190, Abu Dhabi, United Arab Emirates, 2022b. Association for Computational Linguistics.

Hedegaard, L., Alok, A., Jose, J., and Iosifidis, A. Structured Pruning Adapters, 2022.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

Hinton, G. E., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *ArXiv preprint*, abs/1503.02531, 2015.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for NLP. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2790–2799. PMLR, 2019.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *ArXiv preprint*, abs/2001.08361, 2020.

Kwon, W., Kim, S., Mahoney, M. W., Hassoun, J., Keutzer, K., and Gholami, A. A fast post-training pruning framework for transformers. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24101–24116. Curran Associates, Inc., 2022.

Lagunas, F., Charlaix, E., Sanh, V., and Rush, A. Block pruning for faster transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 10619–10629, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.829.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *NIPS*, 1989.

Lester, B., Al-Rfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3045–3059, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.243.

Li, X. L. and Liang, P. Prefix-tuning: Optimizing continuous prompts for generation. 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)*, pp. 4582–4597, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353.

Li, Y., Luo, F., Tan, C., Wang, M., Huang, S., Li, S., and Bai, J. Parameter-efficient sparsity for large language models fine-tuning. In Raedt, L. D. (ed.), *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 4223–4229. International Joint Conferences on Artificial Intelligence Organization, 2022. doi: 10.24963/ijcai.2022/586. Main Track.

Lialin, V., Deshpande, V., and Rumshisky, A. Scaling down to scale up: A guide to parameter-efficient fine-tuning. *ArXiv preprint*, abs/2303.15647, 2023.

Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., and Han, S. Awq: Activation-aware weight quantization for llm compression and acceleration. *ArXiv preprint*, abs/2306.00978, 2023.

Lin, S., Hilton, J., and Evans, O. TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3214–3252, Dublin, Ireland, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.229.

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

Ma, X., Fang, G., and Wang, X. Llm-pruner: On the structural pruning of large language models. *ArXiv preprint*, abs/2305.11627, 2023.

Mahabadi, R. K., Henderson, J., and Ruder, S. Compacter: Efficient low-rank hypercomplex adapter layers. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 1022–1035, 2021.

Mishra, S., Khashabi, D., Baral, C., and Hajishirzi, H. Cross-task generalization via natural language crowdsourcing instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3470–3487, Dublin, Ireland, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.244.

Nallapati, R., Zhou, B., dos Santos, C., Gulcehre, C., and Xiang, B. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pp. 280–290, Berlin, Germany, 2016. Association for Computational Linguistics. doi: 10.18653/v1/K16-1028.

Panigrahi, A., Saunshi, N., Zhao, H., and Arora, S. Task-specific skill localization in fine-tuned language models. *ArXiv preprint*, abs/2302.06600, 2023.

Pfeiffer, J., Kamath, A., Rücklé, A., Cho, K., and Gurevych, I. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 487–503, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.39.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020.

Rajpurkar, P., Jia, R., and Liang, P. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 784–789, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2124.

Sanh, V., Wolf, T., and Rush, A. M. Movement pruning: Adaptive sparsity by fine-tuning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Shen, M., Molchanov, P., Yin, H., and Alvarez, J. M. When to prune? a policy towards early structural pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12247–12256, 2022a.

Shen, M., Yin, H., Molchanov, P., Mao, L., Liu, J., and Alvarez, J. M. Structural pruning via latency-saliency knapsack. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 12894–12908. Curran Associates, Inc., 2022b.

Sun, M., Liu, Z., Bair, A., and Kolter, J. Z. A simple and effective pruning approach for large language models. *ArXiv preprint*, abs/2306.11695, 2023.

Sung, Y., Nair, V., and Raffel, C. Training neural networks with fixed sparse masks. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 24193–24205, 2021.

Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *ArXiv preprint*, abs/2302.13971, 2023.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Wang, X., Wen, K., Zhang, Z., Hou, L., Liu, Z., and Li, J. Finding skill neurons in pre-trained transformer-based language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11132–11152, Abu Dhabi, United Arab Emirates, 2022a. Association for Computational Linguistics.

Wang, Y., Mishra, S., Alipoormolabashi, P., Kordi, Y., Mirzaei, A., Naik, A., Ashok, A., Dhanasekaran, A. S., Arunkumar, A., Stap, D., Pathak, E., Karamanolakis, G., Lai, H., Purohit, I., Mondal, I., Anderson, J., Kuznia, K., Doshi, K., Pal, K. K., Patel, M., Moradshahi, M., Parmar, M., Purohit, M., Varshney, N., Kaza, P. R., Verma, P., Puri, R. S., Karia, R., Doshi, S., Sampat, S. K., Mishra, S., Reddy A, S., Patro, S., Dixit, T., and Shen, X. Super-NaturalInstructions: Generalization via declarative instructions on 1600+ NLP tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 5085–5109, Abu Dhabi, United Arab Emirates, 2022b. Association for Computational Linguistics.

Xia, M., Zhong, Z., and Chen, D. Structured pruning learns compact and accurate models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1513–1528, Dublin, Ireland, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.107.

Xu, D., Yen, I. E.-H., Zhao, J., and Xiao, Z. Rethinking network pruning – under the pre-train and fine-tune paradigm. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2376–2382, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.188.

Xu, Y., Xie, L., Gu, X., Chen, X., Chang, H., Zhang, H., Chen, Z., Zhang, X., and Tian, Q. Qa-lora: Quantization-aware low-rank adaptation of large language models. *ArXiv preprint*, abs/2309.14717, 2023.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472.

Zhang, M., Shen, C., Yang, Z., Ou, L., Yu, X., Zhuang, B., et al. Pruning meets low-rank parameter-efficient fine-tuning. *ArXiv preprint*, abs/2305.18403, 2023a.

Zhang, Q., Chen, M., Bukharin, A., He, P., Cheng, Y., Chen, W., and Zhao, T. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*, 2023b.

Zhang, Z., Zeng, Z., Lin, Y., Xiao, C., Wang, X., Han, X., Liu, Z., Xie, R., Sun, M., and Zhou, J. Emergent modularity in pre-trained transformers. *ArXiv preprint*, abs/2305.18390, 2023c.

Zhao, W., Huang, Y., Han, X., Liu, Z., Zhang, Z., and Sun, M. Cpet: Effective parameter-efficient tuning for compressed large language models. *ArXiv preprint*, abs/2307.07705, 2023.

A. Hyperparameter and Training Details

Our hyper-parameter settings are shown in Table 6. For GLUE task fine-tuning, we follow the hyper-parameter setting of CoFi (Xia et al., 2022), separating the tasks into big (MNLI, SST2, QNLI, QQP) and small (MRPC, CoLA, RTE, STSB) based on the dataset size. For instruction tuning on the Alpaca dataset, we train the pruned model for 15 epochs after the pre-tuning pruning process to make sure they converge. However, in practice, such training epochs can be reduced. To adaptively increase the tuning parameters in the LM, at the start of fine-tuning, we initialize adapter ranks to 8, with salient layers' ranks linearly increased. The scaling factors are set as 2 statically. Since evaluating billion-level LLaMA models during instruction tuning with all evaluation tasks would be time-consuming, we did not do the TTA evaluation as small models. We do not conduct any hyper-parameters search for any training for fair comparison.

Hypeparameter	GLUE-small	GLUE-big	SQuAD	CNN/DM	Alpaca
Learning rate	2e-4	2e-4	2e-4	1e-4	1e-4
Batch size	32	32	32	16	32
Epochs	40	40	40	16	15
Distill epochs	20	20	20	6	-

Table 6. Hyperparameters used in APT experiments

When pruning LMs with APT, following (Xia et al., 2022), we first prune and train the LM with the self-distillation objective, and then fine-tune the pruned LM to recover its end-task performance. Given T pruning training steps in total, we set a pre-determined target sparsity γ_T (defined as the ratio of pruned parameter size to the total parameter size) and use cubic scheduling to control the LM parameter size, where $\gamma_t = \gamma_T + (1-\gamma_T)(1-\frac{t}{T})^3$. We adaptively increase the tuning parameters in the pruning stage but restrict them to a specific limit Δ_t at each training step t. Towards better training stability in LM pruning, we gradually decrease the pruning masks of pruned blocks by $\alpha < 1$ instead of instantly setting them from ones to zeros. We also use the exponential moving-averaged salience in (Zhang et al., 2023b) when calculating the salience score during fine-tuning.

B. Block salience calculation and correlations

As addressed in Section 4.1, we use the compressed weight-gradient production as the salience metric for identifying the tuning and pruning parameter blocks in LMs. Previous works (Sanh et al., 2020) use salience score defined as the magnitude of the parameters' weight-gradient production, where given a linear layer $H = WX$ (we omit the bias term here for simplicity) in model parameters Θ trained on the objective $\mathcal{L}$, the salience scoring function S is defined as:

$$
\begin{aligned}
S(W_{i,j}) &= \sum_{(x,y)\in\mathcal{D}} s(W_{i,j}, x, y) \\
&= \sum_{(x,y)\in\mathcal{D}} \left|\frac{\partial\mathcal{L}(x,y|\Theta)}{\partial W_{i,j}} \cdot W_{i,j}\right| \\
S(W_{:,j}) &= \sum_{(x,y)\in\mathcal{D}} \sum_{i} \left|\frac{\partial\mathcal{L}(x,y|\Theta)}{\partial W_{i,j}} \cdot W_{i,j}\right| \\
&= \sum_{(x,y)\in\mathcal{D}} \Big(\sum_{i} \left|\frac{\partial\mathcal{L}(x,y|\Theta)}{\partial X_{j,i}} \cdot X_{j,i}\right|\Big)
\end{aligned}
\tag{8}
$$

where x, y are the inputs and labels sampled from the training batch $\mathcal{D}$. $S(W_{i,j})$ denotes the unstructured, sparse parameter's salience, and $S(W_{:,j})$ denotes the salience score of a block in the weight W (for example, rows, columns, attention heads, etc.).

When applying this equation to APT adapter layers as defined in Equation (2), there are three different consistent dimensions, namely input dimension j, output dimension i, and tuning rank dimension k. Therefore, the combined salience (including

Algorithm 1 Adaptive Pruning and Tuning

```
Input: Model $f$; Training dataset $\mathcal{D}$; total training steps $T$; Adjustment step set $\mathcal{T}$; Training target $\mathcal{L}$; Initial parameters and masks $\Theta_0, M_0$, training memory budget $\Delta$; Parameter number constraint $\gamma$; Hyperparameters $\alpha \beta$.
for $t = 1, \ldots, T$ do
    Forward pass: $L \leftarrow \mathcal{L}(f(\Theta_t, D_t))$
    Cache the batch-sequence summed hidden states: $\widetilde{H} \leftarrow \sum_{i,j}(|H|)_{ij}$
    Backward pass: $\nabla_{\Theta_t} L \leftarrow \frac{\partial \mathcal{L}(f(\Theta_t, D_t))}{\partial \Theta_t}$
    Calculate approximated salience: $\widetilde{S}(m_i) \leftarrow \widetilde{H} \cdot \sum_{i,j}(|\nabla_H L|)_{ij}$
    Update global scores: $\overline{S}^{(t)}(m) \leftarrow \beta \overline{S}^{(t-1)}(m) + (1-\beta)\widetilde{S}(m)$;
    Select blocks: $M_1, M_0 \leftarrow$ Binary search against constraint Equation (6), with scores $\overline{S}^{(t)}(m)$;
    Update masks: $M_1^{(t)} \leftarrow min(1, M_1^{(t-1)} + \alpha)$, $M_0^{(t)} \leftarrow max(0, M_0^{(t-1)} - \alpha)$;
    Update parameters: $\Theta_{t+1} \leftarrow \Theta_t - \alpha \nabla_{\Theta_t} L$
end for
Output: Parameters and masks $\Theta^{(T)}, M^{(T)}$.
```

tuning low-rank weights and the frozen weight) of the parameter block shall be calculated as follows:

$$
\begin{aligned}
S(H,i) &= \sum_l \frac{\partial \mathcal{L}(x,y|\Theta)}{\partial H(X)_{i,l}} \cdot H(X)_{i,l} \\
&= \sum_p \frac{\partial \mathcal{L}(x,y|\Theta)}{\partial W_{i,p}} \cdot W_{i,p} \\
&+ s \cdot \sum_q \frac{\partial \mathcal{L}(x,y|\Theta)}{\partial W_{B i,q}} \cdot W_{B i,q} \\
S(H,j) &= \sum_l \frac{\partial \mathcal{L}(x,y|\Theta)}{\partial X_{j,l}} \cdot X_{j,l} \\
&= \sum_p \frac{\partial \mathcal{L}(x,y|\Theta)}{\partial W_{p,j}} \cdot W_{p,j} \\
&+ s \cdot \sum_q \frac{\partial \mathcal{L}(x,y|\Theta)}{\partial W_{A q,j}} \cdot W_{A q,j} \\
S(H,k) &= s \cdot \sum_l \frac{\partial \mathcal{L}(x,y|\Theta)}{\partial W_{A k,l}} \cdot W_{A k,l} \\
&= s \cdot \sum_l \frac{\partial \mathcal{L}(x,y|\Theta)}{\partial W_{B l,k}} \cdot W_{B l,k}
\end{aligned}
\tag{9}
$$

Therefore, we can notice that the real block-wise salience of the LoRA layer shall be the sum of the block-wise frozen weight salience and the corresponding tuning weight. Hence, the existing work (Zhang et al., 2023a) that only uses the tuning block salience as layer salience leads to sub-optimal pruning results. Meanwhile, we shall also notice the correlation between the input-, output-dimension, and tuning rank dimensions, which are the summation of the weight-gradient production of parameters on different dimensions.

C. Adaptive Pruning and Tuning Details

We show the detailed algorithm description of our Lightweight Parameter Adjustment as described in Section 4.1 in Algorithm 1. For the details of the algorithm, we first sort all blocks by the salience density, defined as the block salience divided by the number of parameters in the block. For instance, given a RoBERTa-base model with the hidden dimension $d_m = 768$, the number of transformer layers $n_L = 12$, the number of attention heads $n_h = 12$, and the number of FFN neurons $n_f = 3072$, we have:

$$\mathcal{C}_{\text{head}} = 4 \times d_m \times d_m / n_h = 196608 \tag{10}$$

$$\mathcal{C}_{\text{neuron}} = 2 \times d_m = 1536 \tag{11}$$

$$\mathcal{C}_{\text{dimension}} = n_L \times (4d_m + 2n_f) = 110592 \tag{12}$$

We also omit the bias term for density calculation since it takes up less than 1% of LM's parameters. Since the number of heads, neurons, and hidden dimensions is ever-changing during pruning, we re-calculate the density after executing each parameter size change. Meanwhile, for T5 and LLaMA-like models, the FFN layers are gated, consisting of up-, gate-, and down-projection linear layers. Therefore, the number of layers in FFN shall be three instead of two in these LMs. Furthermore, for encoder-decoder LMs like T5, the cross-attention layers in the decoder shall also be counted.

After sorting the blocks by salience density, as LM's parameter size monotonically increases with the number of MHA heads, FFN neurons, and hidden dimensions, we conduct a binary search algorithm to identify the blocks shall be retained as LM's parameter size monotonically increases with the number of MHA heads, FFN neurons, and hidden dimensions. Specifically, given a sorted list of N blocks $B = \{b_1, b_2, ..., b_N\}$ and function f for identifying the block's category where

$$f(b_i) = \begin{cases} 0 & \text{if } b_i \text{ is a head} \\ 1 & \text{if } b_i \text{ is a neuron} \\ 2 & \text{if } b_i \text{ is a dimension} \end{cases} \tag{13}$$

given any index i, we can calculate the parameter number of the LM consisting of the top-i blocks by:

$$\begin{aligned} \mathcal{C}_{\text{top-}i} &= (4d'_h \cdot n'_h + 2n'_f) \cdot d'_m \\ n'_h &= \sum_{j=0}^{i-1} \delta(0, f(b_j)) \\ n'_f &= \sum_{j=0}^{i-1} \delta(1, f(b_j)) \\ d'_m &= \sum_{j=0}^{i-1} \delta(2, f(b_j)) \end{aligned} \tag{14}$$

where $\delta(i, j)$ is the Kronecker delta function that valued 1 if $i = j$ and otherwise 0. Hence, we can use binary search to get the top-i salient blocks, which shall be retained given a parameter constraint, and the rest of the block shall be pruned. In our implementation, for training stability, we do not set the pruned blocks' corresponding masks to 0 directly but gradually decrease their values by $\alpha = 0.01$.

D. Additional Baseline Comparisons

In this section, we further compare APT to existing parameter-efficient pruning methods, such as PST and LRP. In the meantime, we also show detailed results of APT pruning compared to the LoRA+Distill baseline with more tasks in the GLUE benchmark and LLaMA-2 13B model pruning results.

D.1. Comparison to PST and LRP

We compare APT with the state-of-the-art joint use of unstructured pruning (Li et al., 2022) and structured pruning (Zhang et al., 2023a) with PEFT on BERT$_{\text{base}}$ model, showing in Table 7. We can see that APT outperforms existing baselines in both 50% and 10% pruning density settings with a notable margin. The performance gain is credited to our more accurate pruning strategy considering frozen and tuning parameters. At the same time, our efficient self-distillation technique used in conjunction with salient parameters added in training also boosts performance recovery.

D.2. Further Comparison to LoRA+Distill

We show the detailed comparison between APT and the LoRA+Distill baseline in Table 8. APT reaches superior task performance compared to the baseline in all seven GLUE tasks listed in the table, with on average 93.5% fine-tuned LM

Density	Method	MNLI	QQP	QNLI	SST2	CoLA	STS-B	MRPC	RTE	GLUE Avg.
50%	MaP	**83.6**	87.8	**91.5**	91.0	**60.1**	**89.8**	90.7	67.2	82.7
	MvP	82.3	87.3	90.8	90.8	57.7	89.4	91.1	67.2	82.1
	PST	81.0	85.8	89.8	91.3	57.6	84.6	90.7	67.9	81.0
	LRP	82.4	87.2	89.6	90.9	54.1	88.7	89.8	69.3	82.2
	APT	82.8	**90.1**	90.1	**92.7**	59.6	88.3	**91.8**	**70.4**	**83.2**
10%	MaP	78.2	83.2	84.1	85.4	27.9	82.3	80.5	50.1	71.4
	MvP	**80.1**	84.4	**87.2**	87.2	28.6	84.3	84.1	57.6	74.2
	PST	79.6	86.1	86.6	89.0	**38.0**	81.3	83.6	63.2	75.9
	LRP	79.4	86.0	85.3	89.1	35.6	83.3	84.4	62.8	75.7
	APT	78.8	**89.4**	85.5	**90.0**	30.9	**86.3**	**88.2**	**65.3**	**76.8**

Table 7. Comparison of APT to existing unstructured pruning baseline with using PEFT in conjunction. The best results are **bold** while the second-best ones are underlined.

Sparsity	Method	MNLI	QQP	QNLI	SST2	CoLA	MRPC	RTE	GLUE Avg.
0%	FT	87.6	91.9	92.8	95.2	91.2	90.2	78.7	89.7
	LoRA	87.5	90.8	93.3	95.0	63.4	89.7	72.1	84.5
40%	LoRA+Distill	84.2	88.3	90.1	91.9	49.9	86.8	68.6	80.0
	APT	86.4	90.9	92.3	94.5	56.5	92.3	74.4	83.9

Table 8. Detailed results of RoBERTa pruning with APT compared to the LoRA+Distill baseline. We ignore the evaluation results of the STS-B task since it cannot be successfully reproduced with CoFi (the distillation backbone).

performance maintained, notably outperforming the joint use of LoRA and knowledge distillation. In particular, the results of STS-B cannot be reproduced when conducting CoFi distillation with LoRA parameters tuned only, so we exclude the comparison on STS-B. Among the other seven tasks in the GLUE benchmark, we find that tasks with relatively smaller dataset sizes, namely CoLA, MRPC, and RTE, reach superior performance gain when using APT. We conclude that this is because, compared to simple fine-tuning, knowledge distillation with salient parameters added in training is more robust and not prone to overfitting the training data.

D.3. LLaMA-2 13B Pruning Results

As shown in Table 9, when pruning LLaMA-2 13B models, APT maintains 90.0% performance of the unpruned LoRA-tuned baseline. Compared to the pruning result on 7B models that maintain 86.4% dense model performance, better accuracies can be recovered in larger models (13B). At the same time, under the same pre-tuning pruning settings, APT performs better than the LLMPruner baseline on all four evaluation tasks, indicating the effectiveness of considering outlier parameters in large LM pruning. Nonetheless, the LoRA+Prune baseline reaches slightly better results than APT when pruning 13B models, illustrating that there is still room for improving pre-tuning pruning methods in future works. More specifically, among the four tasks we use for evaluating large LMs, TruthfulQA benefits the most from Alpaca fine-tuning. We can see that APT reaches superior results on TruthfulQA than existing baselines regardless of model size. The LM's capabilities on ARC and HellaSawg downgrade the most when pruning large LM before fine-tuning, implying possibilities of catastrophic forgetting in this paradigm.

E. Efficiency and Performance Tradeoff Analysis

We use Figure 4 to clearly show the LMs' end-task performance and efficiency tradeoffs between different tuning, pruning, and distillation baselines. We add several extra baselines to conduct more detailed comparisons between APT with existing PEFT, pruning, and distillation methods:

LoRA+Prune w/distill: we first use LoRA to fully converge a model on the task dataset, and then use Mask-Tuning (Kwon

Method	ARC	HellaSwag	MMLU	TruthfulQA	Avg.
LLaMA2 7B	53.1	77.7	43.8	39.0	53.4
LoRA	55.6	79.3	46.9	49.9	57.9
LoRA+Prune	**46.8**	65.2	23.9	46.2	45.5
LLMPruner	39.2	67.0	24.9	40.6	42.9
APT	45.4	**71.1**	**36.9**	**46.6**	**50.0**
LLaMA2 13B	59.4	82.1	55.8	37.4	58.7
LoRA	60.8	82.8	56.0	46.5	61.5
LoRA+Prune	**56.4**	**79.1**	50.7	42.1	**57.1**
LLMPruner	46.8	74.0	24.7	34.8	45.1
APT	49.5	75.8	**52.5**	**44.7**	55.6

Table 9. LLaMA2 7B and 13B 30% sparsity pruning results with GPT4-generated Alpaca dataset, evaluated on the Open LLM leaderboard few-shot tasks.

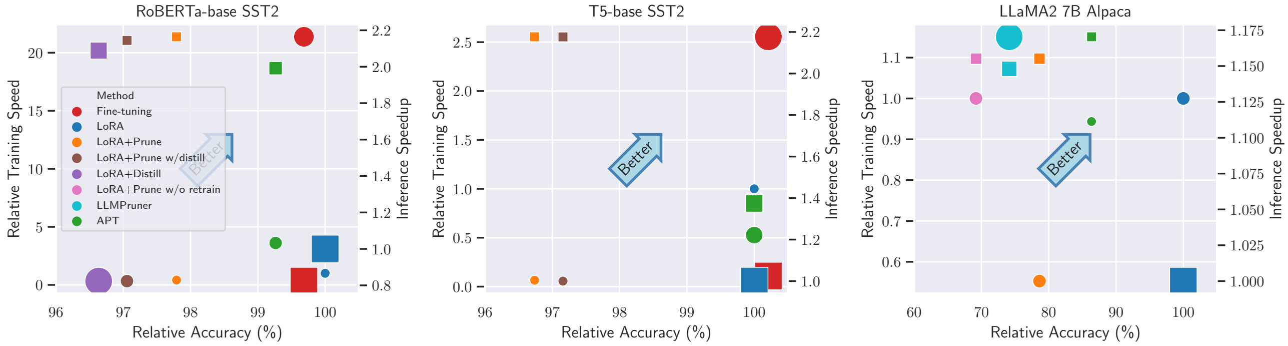


Figure 4. The performance-efficiency tradeoff of APT compared to baseline methods. All metrics are normalized using LoRA tuning w/o pruning as the baseline. The circular dots with vertical axes on the left indicate training speed v.s. performance, with their sizes denoting the peak training memory usage. The squared dots with axes on the right indicate inference speedup v.s. performance, with sizes denoting inference memory usage.

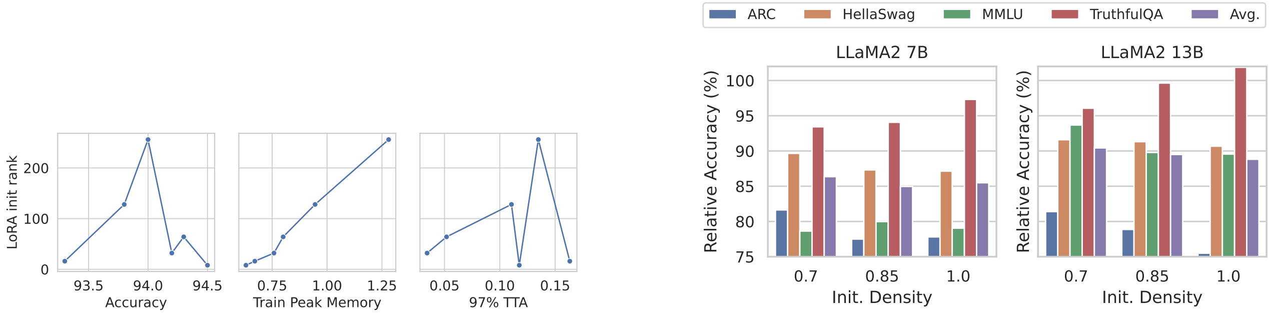
et al., 2022) to prune the LM. Afterward, we utilize the converged model before pruning as the teacher model and distill its knowledge to the pruned student model with static knowledge distillation objectives.

LoRA+Prune w/o retrain: we use Mask-Tuning to prune a LoRA-tuned converged model but do not conduct any retraining to recover the pruned models' performance. Therefore, the LM's training time will be reduced, yet its performance is lower than the LoRA+Prune baseline.

With the same target sparsity in RoBERTa and LLaMA pruning setups, APT achieves on-par end-task performance with full fine-tuning and LoRA tuning baselines. Meanwhile, APT-tuned models reach similar or even better inference time and memory efficiency than existing baselines. APT-pruned T5 LMs' inference efficiency is slightly worse because more decoder parameters (with less computations happening) are pruned than the baselines. Moreover, when pruning RoBERTa and T5 models, APT achieves faster training time than all pruning and distillation baselines. Specifically, the training speed of APT in RoBERTa models is even higher than LoRA tuning without pruning. In LLaMA model pruning, APT costs significantly less training memory than both LLMPruner and LoRA+Prune baselines.

F. Pruning Sparsity Analysis

We further show the task performance changing trajectory with different pruning sparsities in Figure 3. APT achieves superior inference speedup and less inference memory consumption than baselines targeting the same task performance. Compared to the LoRA+Prune baseline, when pruning RoBERTa models targeting similar task accuracy, APT gains 21.8% more inference speedup and 7% more memory reduction. For T5 model pruning with 97% dense model performance maintained, APT results in 62.7% more inference speedup with 24.8% more inference memory reduced compared to the



(a) Comparison of different initial ranks of LoRA layers pruning with APT on RoBERTa with SST2 task accuracy, relative training peak memory and speed to 97% fine-tuning accuracy to the fine-tuning model.

(b) Training initial sparsity trade-off with 30% target sparsity model's relative performances to the LoRA-tuned LLaMA2-7B and 13B models.

Figure 5. Detailed analysis in APT with different initial, target sparsities, and adaptive tuning schedules.

LoRA+Prune baseline. When pruning large LLaMA2-7B models, APT prunes gets 6.7% more speedup and 9.2% more inference memory reduction than the LoRA+Prune baseline, with about 85% dense model performance maintained.

G. Distillation Strategy Comparison

	SST2	Train. Speed($\Uparrow$)	Train. Mem.($\Downarrow$)
APT	94.5	16.9%	70.1%
w/o $\mathcal{L}_{layer}$	93.7	17.4%	69.8%
w/o self-distillation	92.9	20.7%	69.2%
FT teacher	94.3	7.9%	111.8%
LoRA teacher	93.7	1.7%	96.1%

Table 10. Ablation study of distillation strategies and comparison to non-efficient distillation techniques. The training speed and memory are relative metrics compared to fine-tuning the dense model.

We show the further analysis in Table 10 to compare the self-distillation technique we use in APT and traditional knowledge distillation methods. When ablating the dynamic layer mapping strategy in our self-distillation approach, the LM performance decreased by 0.8% with similar training time and memory consumption. When training without distillation objectives (w/o self-distillation), the LM performance drops by 1.7%. Nonetheless, the training is slightly faster with less memory costs. These results present that using distillation objectives for better LM task performance will sacrifice training efficiency as a tradeoff. Furthermore, we also demonstrate the comparisons with existing static knowledge distillation strategies, using the converged full-parameter fine-tuned LM (FT teacher) and LoRA-tuned LM (LoRA teacher) as the teacher model. We calculate the time consumption for both teacher and student training when using these distillation baselines. As shown in Table 10, using fully fine-tuned models as the teacher will incur more memory cost than dense model fine-tuning, while APT only consumes 70%. In the meantime, the training convergence speed of APT training is two times faster than the traditional knowledge distillation method with a fine-tuned teacher. Furthermore, using a LoRA-tuned model as the teacher will result in extremely slow training speed. In addition, simply tuning the LoRA layers with knowledge distillation objectives doesn't help reduce the training memory consumption, as the memory consumption is still 96.1% than full fine-tuning.

H. Adaptive Pruning and Tuning Analysis

Effects of adaptive tuning strategies on end-task performance and training efficiency. As the trajectories shown in Figure 5a, simply enlarging the initial tuning parameter number in APT will not improve or even hurt the model's final performance. Moreover, the training memory consumption grows even higher than fine-tuning when the tuning layer ranks become extremely large (initial ranks set as 256). Therefore, this result proves that adding tuning parameters according to layer salience is better than uniformly increasing them before tuning.

Effects of early pruning on task accuracy and training memory in LLaMA pruning. Figure 5b shows the effect of the initial density on LLaMA models' task performance under the 30% sparsity pruning setting. We find that densely-trained models only perform better in TruthfulQA with fewer parameters pruned before tuning. The accuracy reaches 48.6 and 47.4 when not pruning before tuning, compared to 46.6 and 44.7 when directly pruning to the target sparsity for both 7B and 13B models. Training the LM densely harms the model performance while costing extra memory for all other tasks. These results demonstrate that pruning during training hurts large LM performance under distillation-free settings, and we hypothesize this is due to the training instability issue when parameters are set to zeros during fine-tuning.

I. Absolute Efficiency Metrics

We report the raw efficiency evaluation results in Table 11 and Table 12, including training and inference time and memory consumption. The training times are measured in seconds, and the inference times are measured in milliseconds. All memory footprints are measured in MB. We report the time-to-accuracy for RoBERTa and T5 model training to measure the training time. For LLaMA model training, we measure the training time per epoch to represent training time consumption.

Model	Method	Sparsity	97% TTA (s)	Train Mem. (MB)	Inf. Time (ms)	Inf. Mem (MB)
RoBERTa$_{\text{base}}$	FT	0%	127	2,696	220.8	1,157
	LoRA	0%	2,714	1,630	181.8	1,157
	LoRA+Prune	60%	6,513	1,630	84.0	869
	Prune+Distill	60%	1,899	4,544	85.2	917
	LoRA+Prune+Distill	60%	8,299	3,813	87.0	952
	APT	60%	752	1,890	91.3	904
T5$_{\text{base}}$	FT	0%	366	7,217	248.1	2,347
	LoRA	0%	935	4,476	254.2	2,347
	LoRA+Prune	60%	14,417	4,476	116.8	1,724
	APT	60%	1,774	5,332	185.0	1,913

Table 11. Raw efficiency metrics, including time to accuracy, training peak memory, inference time and memory footprints, when using different methods to fine-tune RoBERTa$_{\text{base}}$ and T5$_{\text{base}}$models on SST2.

Method	Train Time (s)	Train Mem. (MB)	Inf. Time (ms)	Inf. Mem (MB)
LoRA	980	32,185	2457.5	45,311
LoRA+MT	980	32,185	2127.5	31,207
LoRA+MT+retrain	1,773	32,185	2127.5	31,207
LLMPruner	852	23,425	2140.6	33,625
APT	1,039	24,408	2099.7	30,469

Table 12. Raw efficiency metrics, including time to accuracy, training peak memory, inference time, and memory footprints, when using different methods to fine-tune LLaMA2 7B models on Alpaca.