TrimLLM: Progressive Layer Dropping for Domain-Specific LLMs

Lanxiang Hu, Tajana Rosing, Hao Zhang
University of California, San Diego
{lah003, tajana, haozhang}@ucsd.edu

Abstract

Specializing large language models (LLMs) for local deployment in domain-specific use cases is necessary for strong performance while meeting latency and privacy constraints. However, conventional task-specific adaptation approaches do not show simultaneous memory saving and inference speedup at deployment time. Practical compression techniques like quantization and pruning require dedicated hardware or kernel support to achieve measured inference speedup. We develop TRIM-LLM based on the *layer-wise specialization* phenomenon we empirically observed and verified on contemporary LLMs. TRIMLLM reduces the depth of LLMs via progressive layer dropping. We show it retains LLMs' capacity in specific domains and achieves inference speedup irrespective of hardware and deep learning frameworks. We evaluated TRIM-LLM on LLMs of various sizes for inference; models adapted on medical, legal, and financial datasets all demonstrate $2.1 - 5.7\times$ inference speedup on consumer GPUs and up to $3.1\times$ speedup on A100 when compared to state-of-the-art model compression algorithms, with no loss in accuracy at 50∼60% model compression ratio. Our code is available at https://github.com/snyhlxde1/TrimLLM.

1 Introduction

Large language models (LLMs) are increasingly prominent, evolving to serve specialized domains such as medicine (Thirunavukarasu et al., 2023), law (Yue et al., 2023), and finance (Wu et al., 2023b). Their deployment in local environments is particularly valuable, addressing latency and privacy concerns, especially where sensitive data are involved. For example, understaffed clinics greatly benefit from medical-specialized LLM assistants deployed locally. However, the substantial memory and computation required for inference present significant barriers to deploying specialized LLMs in such resource-limited scenarios.

Post-training quantization (PTQ) has emerged as a key technique for adapting LLMs to resource-limited environments, by reducing weight bit precision to 4 or even 3 bits, with minimal degradation in model performance. However, the practical implementations of PTQ methods (Dettmers et al., 2022; Xiao et al., 2023; Frantar et al., 2022; Lin et al., 2023) depend on the availability of efficient kernels and vendor-specific hardware support for quantized computational operations. Unfortunately, such support is not widely accessible. In reality, applying many existing PTQ techniques oftentimes *slows down* model inference on consumer-level hardware, as shown in Table 1. Similar results are seen with many pruning algorithms (Kwon et al., 2022; Frantar and Alistarh, 2023a; Sun et al., 2023), which fail to translate theoretical speedup into real performance gains when specific hardware or kernel support (e.g., for sparsity) are absent.

To address these limitations, this paper explores a new way of compressing LLMs. Recent insights in LLM model editing show that middle layers in LLMs are crucial for domain-specific knowledge (Meng et al., 2022a; Li et al., 2023; Azaria and Mitchell, 2023), with attention modules handling general semantic correlations while MLP layers being more task-specific (Geva et al., 2020). In this study, we delve deeper into the domain-specific relevance of various layers in LLMs. Figure 1 reveals that when fine-tuning on science-common-sense and medical domains, we can remove up to 20 and 16 out of 32 layers respectively in LLaMA-7B without compromising performance.

Building on these findings, we hypothesize *layer-wise specialization*: the significance of each layer of an LLM, particularly the MLP layer, varies according to the specific knowledge domain; we can fine-tune a more domain-focused LLM by selectively dropping layers unimportant to the targeted domain. This strategy enables us to craft models that are not only more compact but also finely bal-
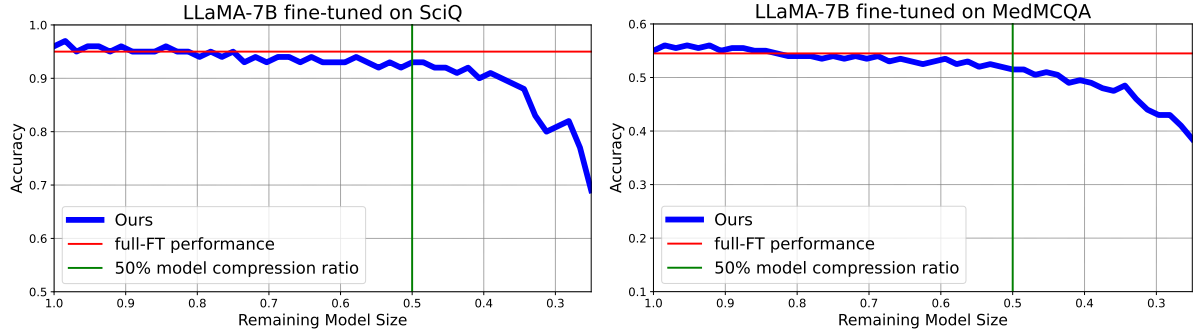
Figure 1: On SciQ and MedMCQA, LLaMA-7B can be reduced to $40\% \sim 50\%$ of its original size with nearly no loss in accuracy. The layer dropping strategy employed is with calibration scanning, activation-norm tie breaker, and sparse udpate at $r = \frac{1}{4}$.

anced in terms of memory usage, inference speed, and domain-specific accuracy.

To validate this hypothesis, we conducted extensive layer-dropping experiments on domain-specific datasets (Pal et al., 2022; Chalkidis et al., 2021; Maia et al., 2023), where one least-important layer is removed after one epoch of fine-tuning. The results indicate that up to a significant number of the layers could be dropped during fine-tuning with negligible accuracy loss using an effective target selection algorithm. Building on these findings, we introduce TRIMLLM, a novel framework that combines fine-tuning with progressive layer-dropping. This approach employs a calibration dataset and an activation-based metric to efficiently identify and eliminate the most non-essential layers after each fine-tuning iteration. Remarkably, TRIMLLM can compress popular LLMs to less than 50% of their original sizes while maintaining domain-specific performance on par with fully fine-tuned models, despite a drastic reduction in parameters. This results in a significant reduction in model depth, therefore memory and computational cost at inference. Moreover, unlike PTQ or existing pruning methods, TRIMLLM does not introduce precision changes or sparse computation and needs no hardware support for measured speedup.

The key contributions of this paper are:

- We observe and empirically validate the *layer-wise specialization* phenomenon in contemporary LLMs.

- We design TRIMLLM, a new model compression approach. TRIMLLM develops a new metric for quantifying layer importance and an algorithm to identify and eliminate layers of minimal importance during the fine-tuning process, compressing LLMs to less than 50% of its original size without compromising its effectiveness.

Our proposed method is orthogonal to the other model compression techniques. They can be used in combination with each other to achieve as much as $8\times$ model compression ratio as detailed in Section 4.3.

- We demonstrate TRIMLLM exceeds the efficiency of full-sized models in domain-specific applications, including medical, legal, and financial fields. We also show TRIMLLM's ability to realize $2.1 - 5.7\times$ inference speedup than the baseline quantization and pruning approaches on consumer-level hardware due to reduced model depth. An additional advantage of TRIMLLM is its ability to provide a flexible continuum of target model sizes, offering greater hardware adaptability than traditional compression methods.

2 Related Work

Task-specific adaptation. A typical workflow for task-specific adaptation is to first fine-tune (Wu et al., 2023a; Yang et al., 2023; Huang et al., 2023b,a) or even pre-train (Wu et al., 2023b; Cui et al., 2023; Shah et al., 2023) LLMs on task-specific datasets before applying any of the following three model compression techniques for reliable performance during inference: quantization, distillation, and pruning. In our case, we adopt layer-dropping to compress the model step-by-step *during* fine-tuning, i.e., we adapt LLMs to domain-specific tasks by identifying and retaining important layers for the target domain.

Quantization. Quantization can effectively mitigate memory consumption by reducing the bit-widths of LLMs' weights and activations. Quantization has featured its ability to retain LLM's zero-shot ability with measured memory saving and theoretical speedup. The state-of-the-art quantization algorithms (Dettmers et al., 2022; Xiao

Table 1: Deployment-time model inference overhead breakdown for LLaMA-7B on different GPUs, with sequence length 512 and batch size 1. The "Mem" entry refers to the ratio of final compressed model size versus the original model size in memory. **TRIMLLM consistently achieves a high inference throughput in all GPU types we test on**.

Hardware	Techniques	Throughput (tokens/s)	Mem
A100	FP16	42.3	100%
	SparseGPT	58.9	100%
	LLM.int8()	29.6	$\geq 50\%$
	GPTQ-int4	46.5	$\geq 25\%$
	AWQ-int4	**115.3**	$\geq 25\%$
	TRIMLLM	**103.1**	$\geq 40\%$
V100	FP16	16.6	100%
	SparseGPT	14.5	100%
	LLM.int8()	10.2	$\geq 50\%$
	GPTQ-int4	6.1	$\geq 25\%$
	AWQ-int4	11.0	$\geq 25\%$
	TRIMLLM	**34.9**	$\geq 40\%$
RTX 3090	FP16	13.4	100%
	SparseGPT	13.0	100%
	LLM.int8()	7.5	$\geq 50\%$
	GPTQ-int4	6.9	$\geq 25\%$
	AWQ-int4	7.9	$\geq 25\%$
	TRIMLLM	**26.8**	$\geq 40\%$

Algorithm 1 TRIMLLM

1: **Input:** Training data $x \in \mathcal{X}$ for the domain-specific task, pre-trained LLM $f(\cdot)$ with parameters θ, training function $\mathcal{F}(\cdot)$ that optimizes some objective ℓ, importance score metric s, sparse update ratio r, accuracy thresholding function $\mathcal{C}_a(a_i)$ or efficiency thresholding function $\mathcal{C}_e(M_i, T_i)$, a_i, M_i and T_i are model's accuracy, memory consumption and latency after the i-th layer is dropped. Buffers for sets $\mathcal{A}_\mathcal{X}$ and $\mathcal{M}_\mathcal{X}$ in Hypothesis 1.
2: $i \leftarrow 0, \mathcal{A}_\mathcal{X} \leftarrow \emptyset, \mathcal{M}_\mathcal{X} \leftarrow \emptyset, \mathcal{U}_\mathcal{X} := \mathcal{A}_\mathcal{X} \bigcup \mathcal{M}_\mathcal{X}, \theta_0 \leftarrow \theta$
3: $\mathcal{G}_{\mathcal{U}_{\mathcal{X}_0}} = f(\cdot), n \leftarrow$ total number of layers in $f(\cdot)$
4: **Sparse update:** Calculate initial s_i for each layer. Freeze layers in accordance with r.
5: choose thresholding function $C(\cdot) \in \{C_a, C_e\}$ that decides whether to exit
6: **while** not $C(\cdot)$ **do**
7:    Run training function to update the set of all parameters $\mathcal{F}(\cdot): \theta_i \rightarrow \theta_i'$
8:    $m \leftarrow 0, U \leftarrow \emptyset$
9:    **while** $m \neq n$ **do**
10:       Calculate layer-wise importance score s_m, append s_m to U
11:       $m+=1$
12:    **end while**
13:    Choose which layer to drop with index m s.t. $s_m = \min(U)$, append s_m to $\mathcal{U}_\mathcal{X}$
14:    Remove parameters: $\theta_i' \rightarrow \theta_{i+1}'$
15:    Remove layer m an update the model: $\mathcal{G}_{\mathcal{U}_{\mathcal{X}_i}} \rightarrow \mathcal{G}_{\mathcal{U}_{\mathcal{X}_{i+1}}}$
16: **end while**
17: **Return:** $\mathcal{G}_{\mathcal{U}_\mathcal{X}}$

---

et al., 2023) require implementations of efficient kernels whose efficiency relies on hardware support. To realize measured speedup for inference, decoding implementation for the specific quantization format is required (Dettmers et al., 2023; Lin et al., 2023). TRIMLLM, on the other hand, does not depend on specialized kernels and it's making the model more efficient by reducing its depth. The performance gain can therefore be generalized to any hardware.

Pruning Pruning aims to remove unimportant weights to reduce FLOPs. Latest post-training pruning algorithms for LLMs focus on structured and unstructured sparsity at neuron- or attention-head level (Liu et al., 2023; Sun et al., 2023; Frantar and Alistarh, 2023b; Ma et al., 2023; Ashkboos et al., 2024) that need efficient kernels and hardware support for the corresponding structured sparsity patterns, without which it's hard to achieve measured efficiency improvement. TRIMLLM again requires none.

Layer-dropping. Layer-dropping is a relatively new technique in the context of LLM model compression. Some prior work investigate the feasibility of layer dropping by compressing a foundation model *before it is fine-tuned* on downstream data (Sajjad et al., 2023) or *during the pertraining stage* (Zhang and He, 2020) (accelerate training with layer-dropping) to improve its efficiency. TRIMLLM conducts layer-dropping *during fine-tuning*, reducing model size and adapting the model for specialized task simultaneously.

Model Editing and Knowledge localization. At layer-wise granularity, evidences (Meng et al., 2022b; Frantar and Alistarh, 2023a) show middle decoder blocks in LLMs contribute more to the domain-knowledge generation process while initial blocks are for low-level information (shallow patterns) extraction and last few blocks capture semantic patterns for next-token generation (Azaria and Mitchell, 2023). Within each decoder block, experiments (Geva et al., 2020; Meng et al., 2022a) show that MLP layers are most responsible for task-specific memory retrieval and factual association. The attention layers, on the other hand, are meant to capture semantic correlation among all input tokens and therefore less specialized (Shaw et al., 2018). TRIMLLM leverages different roles MLP and self-attention layers play to localize and drop the most insignificant layer.

3 Method

3.1 Preliminaries and Layer-Wise Specialization

Auto-regressive language models compose of many transformer decoders, where each decoder block is made of one multi-head attention (MHA) layer and one MLP. Many previous studies on model editing (see Section 2) show increasing evidences suggesting that different layers weight differently when it comes to domain-specific inference, that we call *layer-wise specialization*.

Formally, consider a pre-trained model $f(\mathbf{x}; \theta)$, where $\mathbf{x} \in \mathbb{R}^s$ is an input sequence with sequence length s and embedding dimension n, $\theta \in \mathbb{R}^D$ is a parameter vector that parameterizes $f(\cdot)$ with a total parameter size of D.

Consider layernorm to be part of the MHA and MLP layer along with residual connection with each layer indexed by $i \in \{1, \dots, N\}$, where N is the total number of layers in a model. Let the input to each decoder layer $\mathbf{DEC}_i$ be $\mathbf{y}_{i-1}$ at the current generation step, the corresponding output at layer i follows expression in Eq. 1.

$$\mathbf{y}_i = \mathbf{DEC}_i\left(\mathbf{y}_{i-1}\right) := \mathbf{MLP}_i\left(\mathbf{MHA}_i\left(\mathbf{y}_{i-1}\right)\right) \tag{1}$$

At $i = 1$, the input has $\mathbf{y}_{i-1} = \mathbf{y}_0 = (y_{0,1}, \dots, y_{0,T})$, where T is the current timestamp and y_t is token generated by a previous timestamp $t < T$.

Let the feature space for inputs of a downstream task be $\mathcal{X}$ and input tokens $y_{0,t} \in \mathcal{X}$, and the feature space for generated output tokens be $y_{N,t} \in \mathcal{Y}$ in Equation 2.

$$\begin{aligned} \mathbf{y}_N &= \mathbf{DEC}_N \circ \mathbf{DEC}_{N-1} \circ \cdots \circ \mathbf{DEC}_0\left(\mathbf{y}_0\right) \\ &= f\left(\mathbf{y}_0; \theta\right) \end{aligned} \tag{2}$$

Our basic assumption is that for each downstream task, there exists a feature space $\mathcal{X}$, where $\mathcal{X}$ can be described as a random variable from a distribution $D_\mathcal{X}$, and $\mathcal{Y}$ is a random variable from $D_\mathcal{Y}$. Our hypothesis is:

Hypothesis 1 *Let the set of all attention layers in Equation 1 be $\mathcal{A}$ and the set of all MLP layers be $\mathcal{M}$. For all input sequences x_0 generated from $\mathcal{X}$, there exists a set of attention and MLP layers $\mathcal{A}_\mathcal{X} \subset \mathcal{A}$, $\mathcal{M}_\mathcal{X} \subset \mathcal{M}$ such that the function composition of $U_\mathcal{X} = \mathcal{A}_\mathcal{X} \bigcup \mathcal{M}_\mathcal{X}$ can be fine-tuned on the joint distribution $D_{\mathcal{X}\mathcal{Y}}$ for the downstream task to get a function $\mathcal{G}_{U_\mathcal{X}}$ with $\mathcal{G}_{U_\mathcal{X}}(\mathbf{y}_0) = \mathbf{y}'_N$. It suffices that output of the model $\mathbf{y}'_N$ is generated with random variable $\mathcal{Y}'$ from $D_{\mathcal{Y}'}$ and $D_{\mathcal{Y}'}$ is a close approximation of $D_\mathcal{Y}$ for the full model.*

Note that the order of function composition for $U_\mathcal{X}$ is in accordance with their original order in Equation 1.

3.2 Fine-Tuning with Progressive Layer Dropping

In addition to the ordinary fine-tuning procedure for language models, TRIMLLM iteratively picks a layer to drop after one epoch of training and gradually reduces the model depth. This gives TRIMLLM the advantages of reduced memory consumption and inference latency at deployment time.

Our empirical experiments and recent works (Syed et al., 2023) show drastically changing the model from $f(\mathbf{y}_0; \theta_0) \rightarrow \mathcal{G}_{\mathcal{U}_\mathcal{X}}(\mathbf{y}_0; \theta_f)$ by dropping many parameters all at a time generally gives bad results. This function $\mathcal{G}_{\mathcal{U}_\mathcal{X}}(\mathbf{y}_0; \theta_f)$ maps the generated outputs to a distribution $D_{\mathcal{Y}_f}$ that's very distinct from $D_\mathcal{Y}$ and result in bad domain-specific performance. Note that θ_f is the parameter vector and $D_\mathcal{Y}$ is the output distribution for the full model after fine-tuning. Successive layer dropping, on the other hand, allows domain-specific specialization to be done step by step with $f(\mathbf{y}_0; \theta_0) \rightarrow \mathcal{G}_{\mathcal{U}_{\mathcal{X}_1}}(\mathbf{y}_0; \theta'_1) \rightarrow \mathcal{G}_{\mathcal{U}_{\mathcal{X}_2}}(\mathbf{y}_0; \theta'_2) \cdots \rightarrow \mathcal{G}_{\mathcal{U}_\mathcal{X}}(\mathbf{y}_0; \theta'_f)$ where θ'_i is the parameter vector after i epochs. $\mathcal{G}_{\mathcal{U}_{\mathcal{X}_i}}(\cdot)$ is the model right after the i-th epoch with the corresponding set of remaining layers being $U_{\mathcal{X}_i}$.

This observation aligns the intuition that gradually changing the function's parameterization with most important layers retained allows generated outputs to transit more smoothly from $D'_{\mathcal{Y}_0} \rightarrow D'_{\mathcal{Y}_1} \rightarrow \cdots \rightarrow D'_{\mathcal{Y}_f}$ such that $D'_{\mathcal{Y}_f}$ is a close approximation of $D_\mathcal{Y}$ for the full model after fine-tuning. It thereby provides more evidences to verify our hypothesis in Section 3.1 with an additional constraint:

Proposition 1 *The functional $\mathcal{R} : f(\cdot) \rightarrow \mathcal{G}_{U_{\mathcal{X}_i}}(\cdot)$ needs to be decomposed into successive layer-dropping operators $\{r_0, \dots, r_f\}$ such that the parameter vector θ'_i's dimensionality only changes by a small decrement at a time to gradually adapts a downstream task with the most representative parameters.*

Due to the iterative nature of the aforementioned layer dropping algorithm, the time complex-

ity of fine-tuning increases as more layers are to be dropped. We address this training cost and reduce it to the same order of magnitude as baseline FT that trains for only a few epochs. This is accomplished by using two techniques: sparse fine-tuning as delineated in Section 3.4 and adaptive layer dropping. See Section A.5 further details. In practice, this approach enables users to efficiently exchange a slightly longer model adaptation time for improved inference-time performance, which aligns with the typical development-deployment cycle observed in many real-world applications. In such cases, developers often have the flexibility to accommodate longer development periods but place higher demands on deployment-time performance.

3.3 Target Selection Algorithms

One important aspect of TRIMLLM is choosing the right layer from $U_{\mathcal{X}_i}$ to drop after the i-th epoch and thereby satisfy the successive distribution shift condition (Proposition 1). We introduce two techniques to assign each layer an importance score, where a lower importance score means the layer contribute less to the model's performance on a downstream task.

Sensitivity-based Scoring. The first method is a performance scanning based on a small calibration dataset. Before each time a layer is to be dropped, a small subset of the fine-tuning dataset's validation set is sampled as the calibration dataset. For each layer, its importance score is the reciprocal of the model's performance after dropping the layer. Calibration scanning gives the importance score of any layer i and the expression is presented in Equation 3, where $a_i \in [0, 100]$ is the accuracy of the model after dropping the i-th layer and δ is a small positive number such that $\frac{100}{1+\delta^2}$ is the maximum importance score when $a_i = 0$.

$$s_{i,\text{scan}} = \frac{100 - a_i}{(1 + \delta^2) + (1 + \delta)\, a_i} \qquad (3)$$

Activation-based Scoring. The second method is to make activation-norm comparison on different layers' activations. Recent studies (Dettmers et al., 2022; Xiao et al., 2023) have shown preserving information carried by activations is critical to model's performance when it comes to model compression techniques.

In our work, our goal is to only preserve activations that are meaningful to the knowledge domain of interest. We can drop the rest to trade the model's generality for efficiency and specialization.

A new metric is therefore needed to quantify the importance of an activation.

Our assumptions consist of two parts: (1) there exists a feature space $\mathcal{X}$ and a corresponding low intrinsic dimension (Aghajanyan et al., 2020). (2) activation tensors are dense with mostly small-magnitude elements and a few large-magnitude outliers based on widely recognized observations (Dettmers et al., 2022; Xiao et al., 2023).

Among common matrix norms including the $\ell_{2,1}$ norm, the Forbenius norm and the nuclear norm, at the same numerical value, nuclear norm should be the best metrics for directly measuring the rank of a matrix which is defined as the sum of the singular values of the matrix: $||W||_* = \sum_i \sigma_i$. The nuclear norm is a convex surrogate for the rank function and is often used in rank minimization problems. However, The nuclear norm introduces extra computational overhead because it requires the computation of the SVD of the matrix. Computing the SVD is computationally intensive, especially for large matrices, as it has a complexity of $O(\min(nm^2, mn^2))$ for $m \times n$ matrix. As a result, we use the Forbenius norm to approximate the nuclear norm. By expanding the Forbenius norm with SVD, it follows: $||W||_F = \sqrt{\sum_i \sigma_i^2}$.

Therefore, we choose the Forbenius norm to identify activations with high-rank representations and sparse domain-specific knowledge. Dropping the one with highest norm is analogous to Forbenius norm minimization. Let $\{||\boldsymbol{x}_j||_F\}$ be the set of Forbenius norm for all remaining layers in the model $f(\cdot)$. This activation-norm importance score can be expressed in the form of Equation 4 such that $s_{i,\text{norm}} \in (0, 100]$.

$$s_{i,\text{norm}} = \frac{100 \min\{||\boldsymbol{x}_j||_F\}}{||\boldsymbol{x}_i||_F} \qquad (4)$$

3.4 Sparse update as a Regularization

In TRIMLLM, an important observation is that some less important layers will eventually be dropped regardless whether they have been tuned. Moreover, empirical evidences in Table 3 show fine-tuning all layers could, in effect, perform worse than full fine-tuning.

There are two reasons for the possible performance degradation. First, catastrophic forgetting has been a well recognized problem when a language model is trained on downstream data with all parameters are updated (Lee et al., 2022). Second, layer dropping in TRIMLLM is conducted
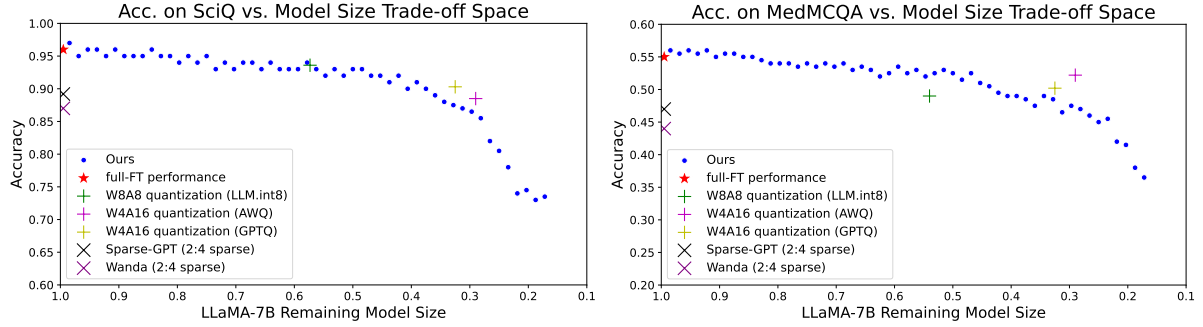
Figure 2: The Pareto Frontier of LLaMA-7B-TRIMLLM on SciQ and MedMCQA. TRIMLLM has a much wider spectrum of operating points to fit the model into different hardware with competitive performance. The layer dropping strategy employed is with calibration scanning and activation-norm tie breaker and + sparse udpate at $r = \frac{1}{4}$.

on the premise that some layers carry less information for a task and can be discarded. However, fine-tuning all layers is based on a contradictory premise that all layers need to be updated for downstream adaptation. As a result, it's natural to adopt a sparse update scheme where we only update the layers with greatest chance to be kept after layer dropping.

To identity which layers to be updated and which to be frozen, we run layer-wise importance score scanning with a calibration dataset before any fine-tuning is done. This gives an initial distribution of all layers' importance scores and probability to be dropped in the first epoch. According to Section A.1, Layer-dropping patterns, since the initial distribution is highly correlated with the latter ones, we can assume fine-tuning with layer dropping won't significantly disturb each layer's importance score and use this initial distribution to infer each layer's overall probability to be dropped. For a sparse update ratio r, only up to $N' = r \times N$ layers will be updated in TRIMLLM. It's possible for any of the N' layers to be dropped during fine-tuning. Each time this occurs, no additional layers will be made trainable.

4 Experiments

In this section, we present experiments that provide empirical evidences for our hypothesis as well as the effectiveness of TRIMLLM. The test suite spans a wide range of knowledge domains including common-sense, medical, legal and financial QA benchmarks. All experiments reported in this section are conducted on LLaMA-7B and LLaMA-13B with training performed on NVIDIA V100 32GB servers. Deployment-time inference speeds are tested on NVIDIA A100 40GB, V100 32GB and RTX 3090 GPUs.

4.1 Performance on QA Benchmarks

To test which of the methods can compress the model to the fullest extent while maintaining more than 90% performance of the full-finetuning baseline, we compare the performance of different sparse update schemes and target selection algorithms. The results are summarized in Table 3. On each QA benchmark, we also compare TRIMLLM and other model compression techniques. The results are presented in Table 2.

Baselines. We use full fine-tuning (full-FT) as our most basic baseline. We also include a sparse fine-tuning (sparse-FT) baseline that only updates the salient layers identified by calibration scanning with the optimal sparse update ratio $\left(r = \frac{1}{4}\right)$. While LLM pruning approaches with structured pruning methods can give inference speedup as shown in Table 1, they are generally incapable of reducing memory consumption without hardware support. As a result, we benchmark TRIMLLM with the state-of-the-art LLM quantization techniques: LLM.int8(), GPTQ and AWQ. They are used as stronger baselines that permit both memory saving and potential inference speedup.

QA benchmarks. We use common-sense QA benchmarks inculuding SciQ (Johannes Welbl, 2017) and PIQA (Bisk et al., 2020) to test LLM's ability of understanding and making basic inference about the physical world the way ordinary humans do. To further assess TRIMLLM's capacity for domain-specific adaptation, we also evaluate its performance on medical, legal, and financial QA datasets: MedMCQA (Pal et al., 2022), LexGLUE-casehold (Chalkidis et al., 2021), and FinanceQA (Bharti, 2023) respectively. For LexGLUE, evaluations are done on the "law" subset of MMLU (Hendrycks et al., 2020). For FinanceQA, the dataset includes a combination of

Table 2: Performance comparison of LLaMA-7B and LLaMA-13B variants on QA benchmarks. The numerical values are percentage in accuracy. TRIMLLM here uses the best strategy with sparse update at $r = \frac{1}{4}$, calibration scanning and activation-norm tie breaker. For sparse-FT, the frozen layers are determined by calibration scanning and $r = \frac{1}{4}$. The "Final Mem" entry refers to the ratio of final compressed model size versus the original model size in memory. The "LLM-Pruner" baseline here uses the best-performing element-wise method at 50% sparsity ratio. The "SparseGPT" baseline uses 2:4 structured sparsity at 50% sparsity ratio, which yields the most significant latency reduction on most hardware.

models	PIQA	SciQ	MedMCQA	LexGLUE	FinanceQA	Final Mem ($\downarrow$)
LLaMA-7B						
w/o training	77.4	89.7	22.4	32.1	33.6	100%
+ Full-FT	82.4	95.6	54.6	42.9	45.1	100%
+ Sparse-FT	83.1	95.4	53.7	43.4	46.9	100%
+ LLM-Pruner	70.3	85.0	23.1	30.8	27.3	100%
+ SparseGPT (2:4)	76.5	90.1	52.3	37.9	41.6	100%
+ LLM.int8()	81.7	93.6	52.0	40.9	**44.9**	> 50%
+ AWQ-int4	80.9	93.0	50.7	41.0	42.1	> 25%
+ TRIMLLM (50%)	**81.8**	**94.2**	**53.1**	**42.0**	43.6	≥ 50%
+ TRIMLLM (40%)	77.6	91.2	47.5	39.5	41.3	≥ 40%
+ TRIMLLM (30%)	68.5	87.3	45.8	36.8	36.0	≥ 30%
LLaMA-13B						
w/o training	79.4	92.0	24.0	37.2	35.3	100%
+ Full-FT	83.9	97.2	57.2	48.3	50.1	100%
+ Sparse-FT	84.1	97.6	56.5	48.0	49.5	100%
+ LLM-Pruner	65.3	80.2	45.2	33.0	29.5	100%
+ SparseGPT	76.0	95.4	53.6	40.8	48.1	100%
+ LLM.int8()	81.5	**96.0**	54.2	45.5	48.4	> 50%
+ AWQ-int4	80.5	95.9	53.5	44.1	45.6	> 25%
+ TRIMLLM (50%)	**82.4**	95.8	**56.9**	**46.3**	**47.5**	≥ 50%
+ TRIMLLM (40%)	79.1	93.5	50.1	43.8	45.8	≥ 40%
+ TRIMLLM (30%)	76.2	91.2	47.4	39.7	41.0	≥ 30%

FiQA (Maia et al., 2023), Stanford-Alpaca (Taori et al., 2023), and ChatGPT QA dialogues. Evaluations of are conducted on the "economics" subset of MMLU for its pertinence to financial knowledge. Cross-evaluations are conducted on the PubMedQA (Jin et al., 2019) and Legal-bench (Guha et al., 2023) benchmarks to test the specialized models' performance on a similar knowledge domain. All experiments adhere to established academic evaluation standards, utilizing the lm-evaluation-hardness[1] repository.

Results. In addition to the two target selection methods introduced in Section 3.3, we device a new two-step algorithm that leverages both methods, which corresponds to the entry "both" in Table 3. This method adopts the more effective calibration scanning as the primary method for layer dropping target selection and uses activation-norm comparison as the tie-breaker strategy when there are more than one layer have the same importance score from calibration scanning. We can see from Table 3 the

two-step algorithm gives the best specialized model at every sparse update ratios.

For each of the three methods, we evaluate specialized models performance when they are trained with different sparse update ratio $r = \left\{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}\right\}$. As we can see in Table 3, in comparison with other target selection techniques, we find at a sparse update ratio of $r = \frac{1}{4}$, the model performs the best. See Section A.4 for detailed ablation on different values of r. The results also show that all three rule-based methods employed in Sajjad et al. (2023) perform very poor in comparison with TRIMLLM.

To demonstrate TRIMLLM's effectiveness on other LLMs, experiments are conducted on OPT-1.3B and OPT-6.7B for one common-sense, medical, and legal benchmark each, in Table 8. The results validate the generalizability of our method as its effectiveness on the OPT models.

4.2 Memory Consumption and Latency

We argue the TRIMLLM has a two-fold advantage. The first one is efficiency and the other is flexibility.

On the efficiency side, TRIMLLM has both

---

[1]The repository is publically available at https://github.com/EleutherAI/lm-evaluation-harness.

Table 3: Performance comparison of LLaMA-7B TRIMLLM variants on QA benchmarks with various target selection algorithms (Section 3.3). The optimal sparse update ratio at $r = \frac{1}{4}$ is used. For sparse-FT, the frozen layers are determined by calibration scanning and $r = \frac{1}{4}$. Check Table 6 for a comprehension ablation on different sparse-update ratios.

methods	PIQA	SciQ	MedMCQA	LexGLUE	FinanceQA	Final Mem ($\downarrow$)
			LLaMA-7B			
w/o fine-tuning	77.4	89.7	22.4	32.1	33.6	100%
+ Full-FT	82.4	95.6	54.6	42.9	45.1	100%
+ Sparse-FT	83.1	95.4	53.7	43.3	46.9	100%
			LLaMA-7B-TRIMLLM $\left(r = \frac{1}{4}\right)$			
+ calibration	80.5	94.0	52.4	41.5	42.3	$\geq 55\%$
+ activation-norm	79.6	93.5	51.5	39.8	41.7	$\geq 85\%$
+ both	**81.8**	**94.2**	**53.1**	**42.0**	**43.6**	$\geq \mathbf{50\%}$
			LLaMA-7B Rule-based Layer Dropping (Sajjad et al., 2023)			
+ random 50% layers	65.4	78.5	35.0	21.3	21.3	$\geq 50\%$
+ top 50% layers	68.7	82.2	34.2	23.1	26.0	$\geq 50\%$
+ bottom 50% layers	50.6	63.9	20.3	17.4	16.8	$\geq 50\%$

deployment-time memory saving and inference speedup. We compare TRIMLLM with other model compression baselines as shown in Table 1 and Figure 2. The state-of-the-art quantization techniques are able to reduce inference-time memory consumption to nearly a quarter in size. TRIM-LLM exploits the model depth degree of freedom and is able to achieve competitive memory saving compared to the quantization baselines with faster inference speed (Table 1) on consumer-level hardware, V100 and RTX 3090 GPUs, where hardware support for low-precision inference and structured sparsity are unavailable in the tensor cores.

On the flexibility side, as we can see from Figure 2, quantization and pruning offers a very limited set of operating points corresponding to each of the bit precision scheme for each model. Since sparsity ratio in pruning can not be easily translated into memory saving, pruning oftentimes gives even fewer operating points in the trade-off space. In contrast, the Pareto frontiers of TRIMLLM span a wide range of operating points. As a result, TRIM-LLM is more flexible and is capable of fitting a model to a wide spectrum of hardware.

4.3 Applying Other Model Compression Techniques to TRIMLLM

Our method is orthogonal to all model compression techniques. Applying TRIMLLM alongside with other post-training model compression techniques like quantization can provide further speedup. Results from applying AWQ and SparseGPT with 2:4 structured sparsity to TRIMLLM, and the corresponding compressed models' accuracy, memory consumption and inference latency are reported in

Table 7 with LLaMA-7B across a variety of benchmarks.

The results show that on A100, other model compressible techniques can be seamless integrated with TRIMLLM to obtain even more efficient domain-specific models with as much as $8\times$ model compression ratio in terms of memory consumption, which translates to $4.5\times$ speedup in comparison with the uncompressed baseline.

4.4 Limitations

While increasing TRIMLLM can extend LLM accessibility to a wider audience in domain-specific use cases, specializing LLMs may raise robustness concern when applying the models to tasks that require knowledge from multiple domains. Striking a balance between accessibility and maintaining the integrity and reliability of language models is essential to ensure their responsible use in various applications.

5 Conclusion

We propose TRIMLLM, a task-specific adaption and model compression pipeline for contemporary LLMs. TRIMLLM reduces deployment-time memory cost and inference latency by identifying and discarding less significant layers to reduce the specialized model's depth. Unlike baselines, TRIM-LLM can obtain both wall-clock inference speedup and memory saving without the need for specialized hardware and efficient computational kernels. We hope that TRIMLLM paves the path for making LLMs accessible to the wider public in personal and professional use cases.

6 Use of AI Assistants

In adherence to the ACL Publication Ethics Policy, we did not employ AI assistants to generate the initial draft of this paper. We used AI assistants (GPT-4o) exclusively at the sentence level to enhance writing quality and correct grammatical errors.

References

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. 2020. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*.

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. 2024. Slicegpt: Compress large language models by deleting rows and columns. *Preprint*, arXiv:2401.15024.

Amos Azaria and Tom Mitchell. 2023. The internal state of an llm knows when its lying. *arXiv preprint arXiv:2304.13734*.

Gaurang Bharti. 2023. gbharti/finance-alpaca. Accessed: 2023-09-20.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Ilias Chalkidis, Abhik Jana, Dirk Hartung, Michael Bommarito, Ion Androutsopoulos, Daniel Martin Katz, and Nikolaos Aletras. 2021. Lexglue: A benchmark dataset for legal language understanding in english. *arXiv preprint arXiv:2110.00976*.

Jiaxi Cui, Zongjian Li, Yang Yan, Bohua Chen, and Li Yuan. 2023. Chatlaw: Open-source legal large language model with integrated external knowledge bases. *arXiv preprint arXiv:2306.16092*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*.

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. 2023. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*.

Elias Frantar and Dan Alistarh. 2023a. Massive language models can be accurately pruned in one-shot. *arXiv preprint arXiv:2301.00774*.

Elias Frantar and Dan Alistarh. 2023b. Sparsegpt: Massive language models can be accurately pruned in one-shot.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2020. Transformer feed-forward layers are key-value memories. *arXiv preprint arXiv:2012.14913*.

Neel Guha, Julian Nyarko, Daniel E Ho, Christopher Ré, Adam Chilton, Aditya Narayana, Alex Chohlas-Wood, Austin Peters, Brandon Waldon, Daniel N Rockmore, et al. 2023. Legalbench: A collaboratively built benchmark for measuring legal reasoning in large language models. *arXiv preprint arXiv:2308.11462*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Quzhe Huang, Mingxu Tao, Zhenwei An, Chen Zhang, Cong Jiang, Zhibin Chen, Zirui Wu, and Yansong Feng. 2023a. Lawyer llama. https://github.com/AndrewZhe/lawyer-llama.

Quzhe Huang, Mingxu Tao, Zhenwei An, Chen Zhang, Cong Jiang, Zhibin Chen, Zirui Wu, and Yansong Feng. 2023b. Lawyer llama technical report. *ArXiv*, abs/2305.15062.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William W Cohen, and Xinghua Lu. 2019. Pubmedqa: A dataset for biomedical research question answering. *arXiv preprint arXiv:1909.06146*.

Matt Gardner Johannes Welbl, Nelson F. Liu. 2017. Crowdsourcing multiple choice science questions.

Woosuk Kwon, Sehoon Kim, Michael W Mahoney, Joseph Hassoun, Kurt Keutzer, and Amir Gholami. 2022. A fast post-training pruning framework for transformers. *Advances in Neural Information Processing Systems*, 35:24101–24116.

Yoonho Lee, Annie S Chen, Fahim Tajwar, Ananya Kumar, Huaxiu Yao, Percy Liang, and Chelsea Finn. 2022. Surgical fine-tuning improves adaptation to distribution shifts. *arXiv preprint arXiv:2210.11466*.

Xiaopeng Li, Shasha Li, Shezheng Song, Jing Yang, Jun Ma, and Jie Yu. 2023. Pmet: Precise model editing in a transformer. *arXiv preprint arXiv:2308.08742*.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. 2023. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. 2023. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pages 22137–22176. PMLR.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*.

Macedo Maia, André Freitas, and Alexandra et al. Balahur. 2023. fiqa. Accessed: 2023-09-20.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022a. Locating and editing factual associations in gpt. *Advances in Neural Information Processing Systems*, 35:17359–17372.

Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. 2022b. Mass-editing memory in a transformer. *arXiv preprint arXiv:2210.07229*.

Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. 2022. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In *Conference on Health, Inference, and Learning*, pages 248–260. PMLR.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2023. On the effect of dropping layers of pre-trained transformer models. *Computer Speech & Language*, 77:101429.

Nigam H Shah, David Entwistle, and Michael A Pfeffer. 2023. Creation and adoption of large language models in medicine. *JAMA*, 330(9):866–869.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*.

Aaquib Syed, Phillip Huang Guo, and Vijaykaarti Sundarapandiyan. 2023. Prune and tune: Improving efficient pruning techniques for massive language models.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

Arun James Thirunavukarasu, Darren Shu Jeng Ting, Kabilan Elangovan, Laura Gutierrez, Ting Fang Tan, and Daniel Shu Wei Ting. 2023. Large language models in medicine. *Nature medicine*, pages 1–11.

Chaoyi Wu, Xiaoman Zhang, Ya Zhang, Yanfeng Wang, and Weidi Xie. 2023a. Pmc-llama: Further fine-tuning llama on medical papers. *arXiv preprint arXiv:2304.14454*.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. 2023b. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR.

Hongyang Yang, Xiao-Yang Liu, and Christina Dan Wang. 2023. Fingpt: Open-source financial large language models. *arXiv preprint arXiv:2306.06031*.

Shengbin Yue, Wei Chen, Siyuan Wang, Bingxuan Li, Chenchen Shen, Shujun Liu, Yuxuan Zhou, Yao Xiao, Song Yun, Wei Lin, et al. 2023. Disc-lawllm: Fine-tuning large language models for intelligent legal services. *arXiv preprint arXiv:2309.11325*.

Minjia Zhang and Yuxiong He. 2020. Accelerating training of transformer-based language models with progressive layer dropping. *Advances in Neural Information Processing Systems*, 33:14011–14023.

A Appendix

A.1 Layer-dropping Patterns

For each of the downstream task shown in Figure 3, there are a few key observations can be made: (1) LLaMA-7B have different layer dropping patterns on different tasks, (2) there are significantly more MLP layers are dropped than the self-attention ones. The first observation provides more empirical evidences for layer-wise specialization while the second for knowledge localization, which argues domain knowledge is stored in MLPs.
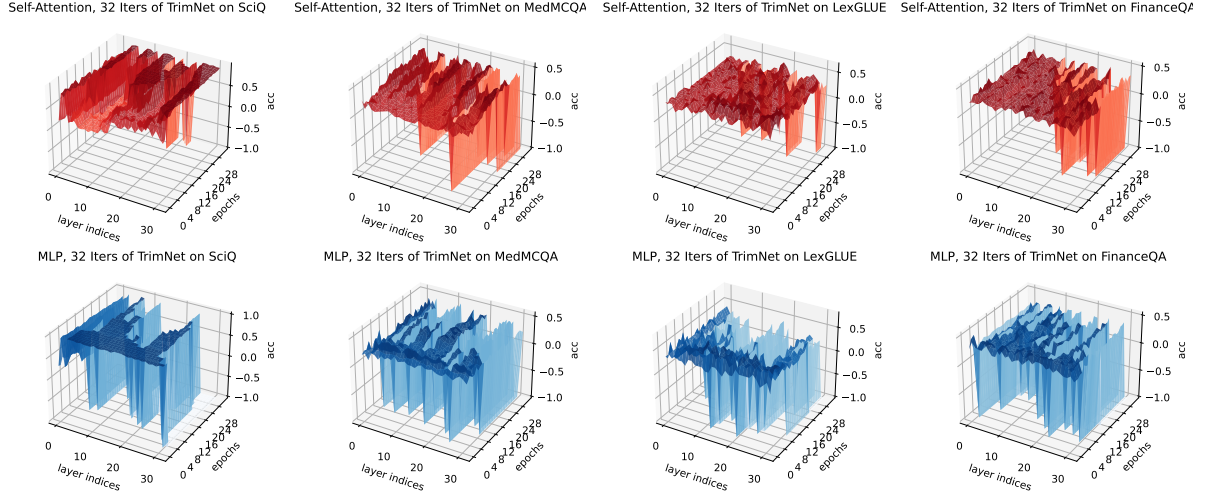

Figure 3: Layer dropping patterns when TRIMLLM (calibration + activation-norm tie breaker) is applied to LLaMA-7B on QA benchmarks. Results for the first 32 iterations are shown. At this point, the model has been reduced to one half of its original size with nearly no performance loss, evidenced in Figure 1. The numerical value -1 is assigned to discarded layers as accuracy no longer applies.

A.2 Ablation: TRIMLLM Robustness on a Similar Domain

We conduct additional experiments to test TRIMLLM 's robustness. Model fine-tuned with the MedMCQA dataset is validated on another dataset, PubMedQA from the medical knowledge domain. Model fine-tuned using the LexGLUE dataset is tested on the Legalbench benchmark. Results in Table 4 show the model specialized on MedMCQA can perform relatively well on PubMedQA, in comparison with benchmarks from totally different knowledge domains. Same conclusion applies to the model specialized on LexGLUE.

Table 4: Performance of specialized LLaMA-7B on different but similar benchmarks.The percentage in parenthesis indicates the percentage of total parameters remained in the specialized model.

model	MedMCQA	PubMedQA	LexGLUE	LegalBench
w/o fine-tuning (100%)	22.4	5.2	32.1	44.9
MedMCQA specialized (40%)	**47.5**	**58.9**	12.4	19.8
LexGLUE specialized (40%)	9.1	34.5	**39.5**	**49.4**

A.3 Ablation: Performance Degradation on Unrelated Domains.

We test specialized models' performance on other domain-specific tasks and it demonstrates a significant performance degradation. Results of each specialized model's performance on other tasks are provided in Table 5.

By analyzing the outcomes of both the robustness and generalizability experiments in Section A.2 and Table 5, it becomes evident that a layer deemed insignificant in one domain is likely to hold similar irrelevance in a closely related domain. This observation provides further empirical support to the concept of layer-wise specialization within LLMs. Intriguingly, the balance between such specialization and a model's overall generalizability presents itself as a captivating avenue for research, meriting exploration in future studies.

Table 5: Performance of specialized LLaMA-7B on other QA benchmarks. The percentage in parenthesis indicates the percentage of total parameters remained in the specialized model.

model	PIQA	SciQ	MedMCQA	LexGLUE	FinanceQA
w/o fine-tuning (100%)	**77.4**	89.7	22.4	32.1	33.6
PIQA specialized (40%)	**77.6**	81.1	14.4	17.8	18.2
SciQ specialized (40%)	61.5	**91.2**	18.9	13.0	16.5
MedMCQA specialized (40%)	54.9	78.2	**47.5**	12.4	14.8
LexGLUE specialized (40%)	62.4	73.1	9.1	**39.5**	18.3
FinanceQA specialized (40%)	55.3	72.5	13.8	21.7	**41.3**

A.4 Ablation: Different Sparse Update Ratios

As we can see in Table 6, results show TRIMLLM performs the worst when all layers are updated with a sparse update ratio $r = 1$. With a ratio of $r = \frac{1}{4}$, the model can be compressed to a greatest extent with more than 16 decoder layers (out of 32) dropped with nearly no loss in accuracy.

A.5 TRIMLLM Fine-tuning Time Complexity Analysis

For conventional full FT, assume the average time it takes to train a layer for one epoch can be approximated by some parameter c. In practice, c is a function of positional index (depth of the layer), parameter size, dataset size, sequence length, operator types, hardware types, and other factors for each layer. Specifically, c differs significantly for the MLP and the attention layers. This difference in forward time can be used to assign MLP and attention layers with different weights in addition to the metrics in Equation 3 and Equation 4. In this analysis, we perform order-of-magnitude approximation, assuming c is given as prior knowledge, and leave the opportunities of dynamically estimating c for future work.

With this approximation, the time it takes to train N layers for one epoch is $T(N) = cN$ and $T_{\text{FFT}} = cN \times n$ for n epochs. With one layer dropped at a time, let the total number of layers to be dropped be n_d, the time can be approximated as:

$$T_{\text{FFT},\Delta=1}(n_d) = T(N) + T(N-1) + \cdots + T(N - n_d + 1) = c \cdot n_d \cdot \left(N - \frac{n_d - 1}{2} \right) \quad (5)$$

Similarly, we can write down approximated time for dropping two layers at a time, and it amounts to:

$$T_{\text{FFT},\Delta=2}(n_d) = T(N) + T(N-2) + \cdots + T(N - n_d + 2) = c \cdot \frac{n_d}{2} \cdot \left(N - \frac{n_d}{2} + 1 \right) \quad (6)$$

On the other hand, if we apply **sparse FT** introduced in Section 3.4, empirical results show it reduces training time to $\sim 60\%$ at the sparsity ratio of $r = 1/4$ as evidenced in profiling results from Table 9. The table shows TRIMLLM take around $\sim 40\%$ of the time to train in comparison with full FT, and it's now possible to run the proposed fine-tuning scheme that iteratively compress a model without sacrificing too much training overhead.

However, it's arguable that the training overhead is still significant as demonstrated in Figure 4. We further introduce **adaptive layer dropping**, which is to drop more than one layer per epoch. From table 9, we see the conversion factor for training time from full FT to sparse FT is roughly 0.6. Plug the scaling

factor into Equation 5 and Equation 6, we can take the estimation $T_{\text{SFT},\Delta=1}(n_d) = 0.6T_{\text{FFT},\Delta=1}(n_d)$, and $T_{\text{SFT},\Delta=2}(n_d) = 0.6T_{\text{FFT},\Delta=2}(n_d)$.

Assume $c = 1$, we can plot all equation together in Figure 4 to compare the training cost of different schemes. If we compare $T_{\text{FFT},\Delta=2}(n_d)$ with $T_{\text{FFT}} = cN \times n$, it can be seen that at the 50% compression ratio, it take roughly the same amount of time to obtain TRIMLLM by dropping two layers at a time with sparse update as performing full FT for 5 epochs. This amounts to roughly the same amount of time it takes for standard LLM fine-tuning practice. In addition, our experiments show dropping two layers epoch using the current method is feasible without introducing significant overhead when the number of layer dropped is small. We leave this to our future work for further efficiency improvement.
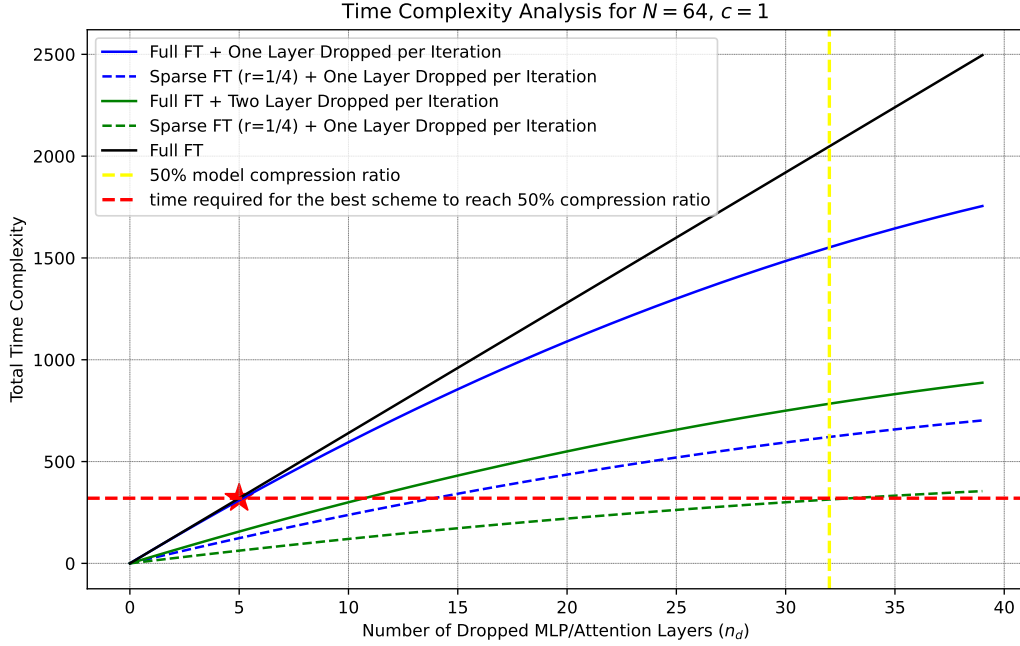
Figure 4: An illustration of fine-tuning time complexity for different combinations of sparse fine-tuning schemes and number of layers to be dropped versus full fine-tuning. For all curves drawn, we use normalized $c = 1$, $N = 64$ to simulate the total number of layers in LLaMA-7B. When two layers are dropped per iteration with sparse FT at $r = \frac{1}{4}$, it only requires 5 epochs of fine-tuning to achieve 50% model compression ratio.

Table 6: Performance comparison of LLaMA-7B TRIMLLM variants on QA benchmarks with different combinations of sparse update techniques (Section 3.2) and target selection algorithms (Section 3.3). For sparse-FT, the frozen layers are determined by calibration scanning and $r = \frac{1}{4}$.

methods	PIQA	SciQ	MedMCQA	LexGLUE	FinanceQA	Final Mem ($\downarrow$)
LLaMA-7B						
w/o fine-tuning	77.4	89.7	22.4	32.1	33.6	100%
+ Full-FT	82.4	95.6	54.6	42.9	45.1	100%
+ Sparse-FT	83.1	95.4	53.7	43.3	46.9	100%
LLaMA-7B-TRIMLLM ($r = 1$)						
+ calibration	81.1	94.2	52.1	41.8	42.5	$\geq 85\%$
+ activation-norm	78.5	93.8	51.2	40.4	41.4	$\geq 90\%$
+ both	80.5	94.0	52.7	42.0	43.3	$\geq 70\%$
LLaMA-7B-TRIMLLM $\left(r = \frac{1}{2}\right)$						
+ calibration	80.2	93.6	51.5	42.3	41.9	$\geq 60\%$
+ activation-norm	79.5	93.4	51.8	40.1	42.0	$\geq 85\%$
+ both	80.6	93.9	52.5	41.4	42.2	$\geq 55\%$
LLaMA-7B-TRIMLLM $\left(r = \frac{1}{4}\right)$						
+ calibration	80.5	94.0	52.4	41.5	42.3	$\geq 55\%$
+ activation-norm	79.6	93.5	51.5	39.8	41.7	$\geq 85\%$
+ both	**81.8**	**94.2**	**53.1**	**42.0**	**43.6**	$\geq \mathbf{50}\%$
LLaMA-7B-TRIMLLM $\left(r = \frac{1}{8}\right)$						
+ calibration	81.3	94.5	52.6	42.0	42.8	$\geq 70\%$
+ activation-norm	80.4	94.0	51.9	39.6	42.3	$\geq 90\%$
+ both	81.5	94.3	52.8	41.6	43.1	$\geq 60\%$

Table 7: Performance comparison of LLaMA-7B variants when applying AWQ and sparseGPT to TRIMLLM on domain-specific tasks. The numerical values are percentage in accuracy. Throughputs are meausred on A100 GPUs and are reported in tokens/s with sequence length 512 and batch size 1.

models	SciQ	MedMCQA	FinanceQA	Throughput	Final Mem ($\downarrow$)
LLaMA-7B					
w/o training	89.7	22.4	33.6	42.3	100%
+ Full-FT	95.6	54.6	45.1	42.3	100%
+ Sparse-FT	95.4	53.7	46.9	42.3	100%
+ SparseGPT (2:4)	90.1	52.3	41.6	58.9	100%
+ AWQ-int4	93.0	50.7	42.1	115.3	$> 25\%$
LLaMA-7B-TRIMLLM $\left(r = \frac{1}{4}\right)$					
w/o PT compression	**94.2**	**53.1**	**43.6**	103.1	$\geq 50\%$
+ SparseGPT (2:4)	89.1	47.8	38.9	132.0	$\geq 50\%$
+ AWQ-int4	91.5	49.2	40.5	**188.7**	$> \mathbf{12.5\ \%}$

Table 8: Performance comparison of OPT-1.3B and OPT-6.7B variants on QA benchmarks. The numerical values are percentage in accuracy. TRIMLLM here uses the best strategy with sparse update at $r = \frac{1}{4}$, calibration scanning and activation-norm tie breaker. For sparse-FT, the frozen layers are determined by calibration scanning and $r = \frac{1}{4}$.

models	SciQ	MedMCQA	LexGLUE	Final Mem ($\downarrow$)
		OPT-1.3B		
w/o training	84.9	12.5	18.1	100%
+ Full-FT	91.7	46.2	24.7	100%
+ Sparse-FT	91.5	45.6	25.1	100%
+ SparseGPT (2:4)	88.3	42.1	20.0	100%
+ LLM.int8()	90.8	44.2	24.0	> 50%
+ TRIMLLM (50%)	88.4	43.9	21.3	$\geq$ 50%
+ TRIMLLM (40%)	85.0	39.0	18.5	$\geq$ 40%
		OPT-6.7B		
w/o training	89.0	14.5	22.7	100%
+ Full-FT	95.3	49.8	41.0	100%
+ Sparse-FT	94.9	48.3	40.2	100%
+ SparseGPT (2:4)	91.0	44.2	36.2	100%
+ LLM.int8()	95.1	46.9	39.9	> 50%
+ TRIMLLM (50%)	88.5	44.3	37.5	$\geq$ 50%
+ TRIMLLM (40%)	82.2	41.9	33.7	$\geq$ 40%

Table 9: Evaluation of wall clock time of running TRIMLLM and other baselines on 2A100-80G GPUs on the SciQ. We use the best TRIMLLM scheme reported in Table 3.

Methods	epochs	FT Time (h)
Full FT	2	4.1
	4	8.3
	8	16.5
Sparse FT $\left(r = \frac{1}{4}\right)$	2	2.5
	4	5.0
	8	10.0
TRIMLLM	2	1.8
	4	3.4
	8	6.5