Offsite-Tuning: Transfer Learning without Full Model

Guangxuan Xiao [1] **Ji Lin** [1] **Song Han** [1]

Abstract

Transfer learning is important for foundation models to adapt to downstream tasks. However, many foundation models are proprietary, so users must share their data with model owners to fine-tune the models, which is costly and raise privacy concerns. Moreover, fine-tuning large foundation models is computation-intensive and impractical for most downstream users. In this paper, we propose Offsite-Tuning, a privacy-preserving and efficient transfer learning framework that can adapt billion-parameter foundation models to downstream data *without* access to the full model. In offsite-tuning, the model owner sends a lightweight *adapter* and a lossy compressed *emulator* to the data owner, who then fine-tunes the adapter on the downstream data with the emulator's assistance. The fine-tuned adapter is then returned to the model owner, who plugs it into the full model to create an adapted foundation model. Offsite-tuning preserves both parties' privacy and is computationally more efficient than the existing fine-tuning methods that require access to the full model weights. We demonstrate the effectiveness of offsite-tuning on various large language and vision foundation models. Offsite-tuning can achieve comparable accuracy as full model fine-tuning while being privacy-preserving and efficient, achieving 6.5× speedup and 5.6× memory reduction. Code is available here.

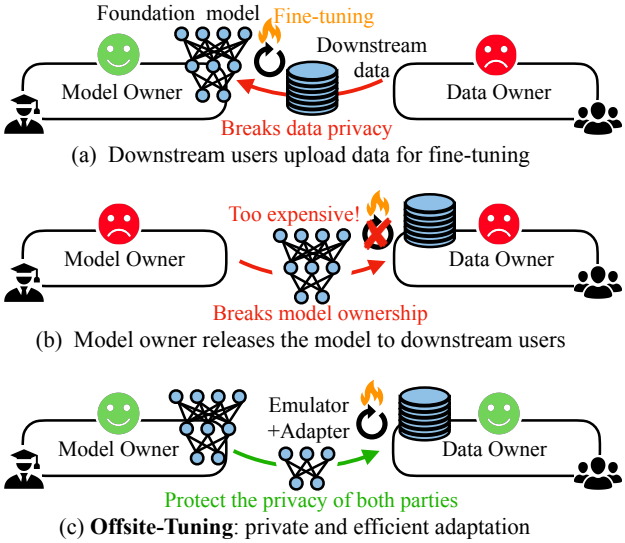

Figure 1. Comparing existing fine-tuning approaches (top and middle) and Offsite-Tuning (bottom). (a) Traditionally, users send labeled data to model owners for fine-tuning, raising privacy concerns and incurring high computational costs. (b) Model owner sending the full model to the data owner is not practical, which threatens the ownership of the proprietary model, and it's not affordable for users to fine-tune the huge foundation model due to resource constraints. (c) Offsite-tuning offers a privacy-preserving and efficient alternative to traditional fine-tuning methods that require access to full model weights.

1 Introduction

Large foundation models have shown exceptional performance in various tasks, including natural language processing (Devlin et al., 2019; Radford et al., 2019; Brown et al., 2020), computer vision (Radford et al., 2021; Fang et al., 2022), and speech recognition (Radford et al., 2022). Through pre-training on vast amounts of data, these models can learn general representations that are useful for a wide range of downstream tasks. Despite the ability of some foundation models to perform zero-shot predictions or in-context learning (Brown et al., 2020; Fang et al., 2022), transfer learning (i.e., fine-tuning) remains a popular and robust method to adapt a general foundation model to a specific task (Wei et al., 2021; Ouyang et al., 2022; Muennighoff et al., 2022; Li & Liang, 2021; Hu et al., 2021). However, tuning foundation models for downstream tasks is difficult due to two reasons (Figure 1). Firstly, training large foundation models usually require enormous computation and data, leading to high training costs (e.g., it is expected that it takes more than $4M to train GPT-3*). Therefore, the trained weights are usually proprietary and not made public. This means downstream users must share their labeled data with the model owners to fine-tune the models (e.g.,

[1]Massachusetts Institute of Technology. Correspondence to: Guangxuan Xiao <xgx@mit.edu>.

*https://lambdalabs.com/blog/demystifying-gpt-3

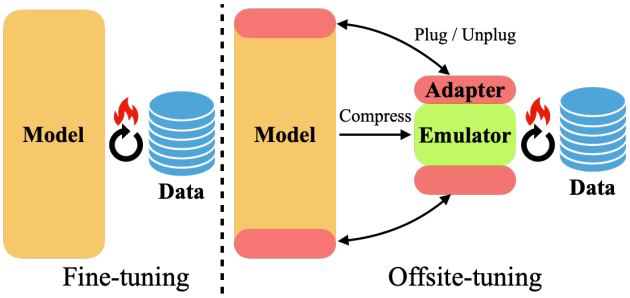

Figure 2. **Overview of Offsite-Tuning.** Fine-tuning (left) requires access to the full model weights and needs both model and data to be in one location. In Offsite-tuning (right), the model owner sends an adapter and an emulator to the data owner, who fine-tunes the adapter on the downstream data with the emulator's assistance. The fine-tuned adapter is then returned and plugged into the full model to create an adapted foundation model. As neither party needs to share full models or data and the emulator is compressed, offsite-tuning is both privacy-preserving and efficient.

the OpenAI Fine-tuning API[†]), which can be costly and raise privacy concerns, putting valuable labeled data at risks. Secondly, even if the downstream users have access to the pre-trained weights, it is quite computationally expensive and difficult to perform the fine-tuning locally. Foundation models typically have a huge number of parameters. For example, the GPT-3 model (Brown et al., 2020) has 175 billion parameters, requiring 350GB GPU memory to store the parameters and perform inference, let alone training. The demanding hardware requirement has made it impossible for most end users to perform transfer learning. Therefore, we need a privacy-preserving and more efficient framework for fine-tuning foundation models.

To address the challenges above, we propose Offsite-Tuning, a privacy-preserving and efficient transfer learning framework that can adapt foundation models to downstream tasks without access to the full model weights. As shown in Figure 2, offsite-tuning involves the model owner sending an *adapter* and an *emulator* to the data owner, who then fine-tunes the adapter on the downstream data with the emulator's assistance. The fine-tuned adapter is then returned to the model owner, who plugs it into the full model to create an adapted foundation model for downstream users. The adapters are used for encoding task-specific knowledge with a small number of parameters, while the compressed emulator mimics the behavior of the rest of the full model and provides approximate gradients for fine-tuning the adapters. Offsite-tuning preserves the *privacy* of data owners, as they do not need to directly share their training data. It also protects the *property* of the foundation model owner, as the full model weights are not shared, and the emulator is lossy compressed with highly degraded performance. Offsite-tuning is also more *resource-efficient* than existing fine-tuning methods that require access to the full model weights, as it allows fine-tuning without the need for the full model through the use of a compressed emulator.

We evaluate the performance of offsite-tuning on a range of language and vision foundation models, including GPT-2, OPT, BLOOM, CLIP, and EVA. Results indicate that offsite-tuning can achieve comparable results to fine-tuning with full model weights on multiple downstream tasks, while also preserving privacy and being more resource-efficient. Data owners can benefit from offsite-tuning by adapting foundation models faster with less resource, achieving up to 6.6× speedup and 5.6× memory reduction compared to full fine-tuning. Additionally, offsite-tuning enables fine-tuning of models that previously could not be achieved on a single GPU, such as OPT-6.7B and BLOOM-7.1B. Overall, we believe that offsite-tuning is a practical framework for safely and efficiently applying foundation models to a broader range of real-world applications.

2 Related Work

Foundation Models (Bommasani et al., 2021), also known as pre-trained models, are large neural networks that have been trained on a large dataset before being used for specific tasks. Although some models like GPT-3 (Brown et al., 2020), CLIP (Radford et al., 2021) and Painter (Wang et al., 2022) can make zero-shot predictions or learn in context, transfer learning remains a mainstream approach for applying models to new tasks (Wei et al., 2021; Muennighoff et al., 2022; Liu et al., 2022). Using foundation models can save time and resources compared to training models from scratch, but fine-tuning and deploying them can be resource-intensive due to their large parameter sizes (Smith et al., 2022; Xiao et al., 2022). Additionally, as many foundation models are non-public, users may need to share their training data with the model's owners for fine-tuning, which can be costly and raise privacy concerns.

Parameter-Efficient Fine-tuning adapts foundation models to downstream tasks by updating or adding only a small number of parameters, rather than updating the entire model. Techniques such as Adapter-tuning (Houlsby et al., 2019), which inserts small task-specific neural networks into transformer layers, Prefix-tuning (Li & Liang, 2021), which prepends task-specific tunable prefix vectors to input sequences, LoRA (Hu et al., 2021), which decomposes the task-specific updates of weights into trainable low-rank vectors, and BitFit (Ben Zaken et al., 2022), which only updates the bias vectors of models, are useful as they require only a small number of parameters to be stored and loaded for each downstream task, while most parameters of the foundation model can be shared. However, it should be noted that

[†]https://beta.openai.com/docs/guides/fine-tuning

while parameter-efficient fine-tuning is useful, it requires the knowledge of the entire model weights, which can compromise either data or model owners' privacy. Additionally, the fine-tuning process remains resource-intensive, as it requires at least one copy of the entire model to be placed on the device.

Federated Learning (McMahan et al., 2017; Konečnỳ et al., 2016; Kairouz et al., 2021; Augenstein et al., 2020) enables users to collectively train or fine-tune a model without sharing their data with a central server. Instead, each user maintains a local copy of the entire model and updates it with their data. The updated model is then sent to the central server, where it is aggregated to create a new global model. However, it is essential to note that while federated learning can protect data privacy by keeping the data on the devices, it does not preserve the model privacy as each user has a copy of the entire model. Furthermore, federated learning assumes that users can perform training on the whole model weights, which is hardly feasible for large foundation models.

Decoupled Learning breaks down the end-to-end optimization problem of neural network training into smaller subproblems. This is achieved through various techniques such as the use of auxiliary variables (Askari et al., 2018; Li et al., 2019; Taylor et al., 2016; Zhang & Brand, 2017), delayed gradient descent (Huo et al., 2018; Xu et al., 2020), and model assembly (Ni et al., 2022). However, current decoupled learning methods have primarily been developed for training neural networks from scratch and have not yet been extensively explored for fine-tuning already trained large foundation models.

3 Problem Definition

Privacy requirements. We consider the privacy of two parties in the transfer learning setting: the *data owner* cannot share their labeled training data with the model owner, and the *foundation model owner* cannot share their model with the data owner. We need to find a way to tune the model on the data owner's data without getting access to the full model weights.

Settings. Given the foundation model $\mathcal{M}$ parameterized by Θ and the downstream dataset $\mathcal{D}$, fine-tuning the model on the downstream datasets yields $\mathcal{M}_\Theta \rightarrow \mathcal{M}_{\Theta+\Delta}$, $\Delta = \arg\min_\delta \mathcal{L}(\Theta+\delta, \mathcal{D})$. To enable private and efficient transfer learning, we want to find a substitute model $\mathcal{M}^*_{\Theta^*}$ (also called as *Emulator*) that is (significantly) smaller and weaker than $\mathcal{M}_\Theta$, so that sharing $\mathcal{M}^*$ with downstream users would not threaten the ownership of the foundation models. Data owners then optimize the substitute model on the dataset, yielding $\mathcal{M}^*_{\Theta^*+\Delta^*}$. We hope that plugging the trained weights Δ^* back to the original model (i.e., $\mathcal{M}_{\Theta+\Delta^*}$) can achieve similar performance compared to directly optimizing $\mathcal{M}$ on the dataset (i.e., $\mathcal{M}_{\Theta+\Delta}$), without giving access to $\mathcal{M}$ itself.

Metrics. To evaluate the performance of the method, we define several metrics. Without loss of generality, we use language models for the definitions.

- *Zero-shot performance* refers to the performance of the pre-trained foundation model when directly evaluated on downstream tasks *without* fine-tuning (i.e., the performance of $\mathcal{M}_\Theta$).
- *Emulator performance* refers to the performance of the small substitute model when fine-tuned on the downstream datasets (i.e., the performance of $\mathcal{M}^*_{\Theta^*+\Delta^*}$)
- *Plug-in performance* refers to the performance of the pre-trained foundation model with plugged-in trained weights from the substitute model (i.e., $\mathcal{M}_{\Theta+\Delta^*}$).
- *Full fine-tuning performance* refers to the performance when we *directly* fine-tune the foundation model on downstream datasets (i.e., $\mathcal{M}_{\Theta+\Delta}$) without considering privacy.

The core concept of offsite-tuning is that downstream users can offsite fine-tune the foundation model on their private data without direct access to the full model. We accomplish this by generating emulated gradients with the emulator that can be leveraged to approximate update the adapters. As such, we term our approach Offsite-Tuning.

To prove the effectiveness of the method, we require:

- Zero-shot performance < plug-in performance, showing that the tuning effectively improves the performance on the specific downstream dataset (otherwise, tuning would not be necessary).
- Emulator performance < plug-in performance, showing that the foundation model still edges in the task (otherwise, the downstream users would be happy to just use the fine-tuned emulator).
- Plug-in performance $\approx$ full fine-tuning performance (so that users do not sacrifice too much performance for data privacy).

4 Offsite-Tuning

4.1 Framework Overview

To attain the desired performance, we divide the foundation model, denoted as $\mathcal{M}$, into two distinct components: a small, trainable adapter, denoted as $\mathcal{A}$, which is intended for downstream adaptation, and the remaining portion of

Algorithm 1 Uniform Layer-drop

Input: a list of layers $[l_0, \ldots, l_{m-1}]$, number of retained layers k
Output: a list of retained layers $[l_{i_0}, \ldots, l_{i_{k-1}}]$
{Make sure the first and last layers are retained}
$stride \leftarrow (m-1)/(k-1)$
for $j \leftarrow 0$ to $k-1$ **do**
  $i_j \leftarrow \lfloor j \times stride \rceil$
end for

the model, denoted as $\mathcal{E}$, which is to be kept frozen. Specifically, $\mathcal{M}$ can be defined as the concatenation of $\mathcal{A}$ and $\mathcal{E}$, such that $\mathcal{M} = [\mathcal{A}, \mathcal{E}]$. To protect model ownership and improve efficiency, lossy compression is applied to the frozen component, resulting in an emulator, denoted as $\mathcal{E}^*$. The downstream user will be provided with the combination of the adapter and emulator, $[\mathcal{A}, \mathcal{E}^*]$, and will be able to perform model tuning by updating $\mathcal{A}$. The updated adapter, denoted as $\mathcal{A}'$, will then be returned to the upstream foundation model owner and integrated into the original model, $\mathcal{M}' = [\mathcal{A}', \mathcal{E}]$, to achieve superior performance on the downstream dataset. It is important to note that due to the lossy compression process, users with $[\mathcal{A}, \mathcal{E}^*]$ will not be able to achieve an acceptable level of performance, even with fine-tuning. Therefore, the integrity of the model ownership is not compromised during this process, and the overall efficiency is improved as a result of the compression.

However, determining an appropriate combination of $\mathcal{A}, \mathcal{E}, \mathcal{E}^*$ is a non-trivial task. Intuitively, the emulator, $\mathcal{E}^*$, should possess a level of similarity to the original frozen component, $\mathcal{E}$, to provide appropriate gradient directions for updating the adapter when fine-tuning on the downstream dataset. At the same time, the emulator cannot be too precise, as this would negate the need for the foundation models by the downstream user. In the following section, we will delve into the design of $\mathcal{A}, \mathcal{E}, \mathcal{E}^*$ and evaluate how different designs impact the aforementioned metrics.

4.2 Adapter Selection

The Transformer architecture (Vaswani et al., 2017) has been widely adopted in foundation models across various modalities, such as language and vision. In this discussion, we will focus on the design of adapters for deep transformer backbones, which can easily be extended to other models, such as convolutional neural networks (CNNs).

We select a small subset of the foundation model as the adapter, which can be trained on various downstream datasets. Since only a subset of the model is updated, the adapter must be generalizable to different downstream tasks. Research has shown that different layers of a transformer, from shallow to deep, encode different levels of feature abstraction, and the selection of layers to update can impact transfer learning performance (Lee et al., 2022; Lin et al., 2022). To cover a wide range of tasks, we choose to include both shallow and deep layers in the adapter, resulting in a sandwich design, $\mathcal{M} = \mathcal{A}_1 \circ \mathcal{E} \circ \mathcal{A}_2$. Our experiments show that this adapter design works well for various downstream tasks and outperforms the common practice of fine-tuning only the last several layers (i.e., $\mathcal{M} = \mathcal{E} \circ \mathcal{A}$), as demonstrated in Figure 3. We include a detailed discussion of the adapter design in Section 5.4.1.

4.3 Emulator Compression

The use of an emulator is to provide the rough gradient directions to update the adapters while remaining similar to the original frozen component, $\mathcal{E}$. However, the emulator must not be too precise, as this would reveal information about the original model. Additionally, a smaller emulator size leads to a more efficient fine-tuning process for downstream users. Therefore, we aim to find a balance between these three requirements.

To achieve this balance, we consider various compression methods, including pruning (Han et al., 2016), quantization (Jacob et al., 2018; Xiao et al., 2022), layer-drop (Sajjad et al., 2023), and knowledge distillation (Hinton et al., 2015; Sanh et al., 2019). Our experiments indicate that the layer-drop-based compression method provides the best balance between the aforementioned criteria. Specifically, we uniformly drop a subset of layers from the frozen component, $\mathcal{E}$, and use the remaining layers as the emulator, $\mathcal{E}^*$. We find it beneficial always to include the first and last layer of the frozen part in the emulator (as shown in Algorithm 1). We provide a detailed comparison of different compression methods in Section 5.4.2.

Additionally, to achieve a higher compression ratio while maintaining approximation accuracy, we apply knowledge distillation to the layer-dropped emulator, $\mathcal{E}^*$, under the supervision of the original component, $\mathcal{E}$, on the pre-training dataset when computing resources are available. The distillation process is performed using mean squared error (MSE) as the loss function, as shown in the following equation:

$$\mathcal{L}_{\text{distill}} = \frac{1}{N} \sum_{i=1}^{N} \|\mathcal{E}^*(x_i) - \mathcal{E}(x_i)\|^2 \quad (1)$$

where x_i is the hidden representation of the i-th input sample produced by previous layers $\mathcal{A}_1$, N is the number of samples in the pre-training dataset. We find that a proper distillation will help maintain similar transfer learning accuracy compared to direct fine-tuning of the full model while still keeping an accuracy gap compared to the compressed model. We discuss the effect of the distillation in Section 5.4.3.

Table 1. Offsite-tuning (OT Plug-in) improves zero-shot (ZS) performance across all tasks, with only slight decreases compared to full fine-tuning (FT). Also, a consistent performance gap is observed between the emulator fine-tuning and plug-in, indicating offsite-tuning effectively preserves the privacy of the original proprietary model (users can not use the emulator to achieve the same performance).

Setting	OpenBookQA	PIQA	ARC-E	ARC-C	HellaSwag	SciQ	WebQs	RACE	WikiText (↓)
GPT2-XL (2-16-2 Distill)									
Full ZS	23.0%	70.9%	58.2%	25.1%	40.0%	83.2%	1.5%	33.0%	20.44
Emulator ZS	18.8%	67.7%	53.2%	20.8%	33.5%	77.0%	0.2%	30.0%	25.12
FT	30.0%	73.2%	62.9%	30.0%	40.7%	92.5%	26.4%	43.2%	13.58
OT Emulator	24.0%	70.3%	58.2%	23.9%	35.8%	92.7%	18.9%	39.4%	17.64
OT Plug-in	28.2%	73.6%	61.4%	28.5%	41.6%	93.2%	19.9%	39.9%	14.94
OPT-1.3B (2-8-2 Distill)									
Full ZS	23.4%	71.6%	56.9%	23.5%	41.5%	84.4%	4.6%	34.2%	31.48
Emulator ZS	19.4%	68.7%	53.9%	21.5%	35.1%	80.9%	1.3%	33.0%	38.55
FT	31.4%	75.2%	61.3%	27.7%	42.7%	92.5%	31.2%	37.0%	12.52
OT Emulator	24.8%	71.6%	58.1%	26.1%	37.0%	92.2%	24.3%	38.6%	15.54
OT Plug-in	29.0%	74.5%	59.4%	27.8%	43.3%	92.9%	26.2%	38.9%	13.15

4.4 Connection to Parameter-Efficient Fine-tuning

Our work is orthogonal to existing parameter-efficient fine-tuning methods for foundation models, such as LoRA (Hu et al., 2021), Adapter (Houlsby et al., 2019), BitFit (Ben Zaken et al., 2022), and Prefix-tuning (Li & Liang, 2021). Our focus is on fine-tuning a small subset of layers for data and model privacy while simultaneously improving efficiency through emulator compression. In contrast, parameter-efficient fine-tuning methods aim to reduce the number of trainable parameters while having access to the *full* pre-trained weights. Our framework can easily integrate with these parameter-efficient fine-tuning methods by incorporating small trainable modules into the adapter component, $\mathcal{A}$. We will present the details of this integration and the results of our experiments in Section 5.5.

5 Experiments

5.1 Setup

Models and Datasets. We evaluate offsite-tuning on large language models, including GPT-2 (Radford et al., 2019), OPT (Zhang et al., 2022), and BLOOM (Scao et al., 2022), as well as vision models such as CLIP (Radford et al., 2021) and EVA (Fang et al., 2022). We evaluate language models on the Wikitext language modeling dataset (Merity et al., 2016) and eight question answering benchmarks, including OpenBookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), ARC (Clark et al., 2018), HellaSwag (Zellers et al., 2019), SciQ (Johannes Welbl, 2017), WebQustions (Berant et al., 2013), and RACE (Lai et al., 2017). We report perplexity on the WikiText dataset and accuracy on the other benchmarks. We evaluate vision models on six fine-grained image classification datasets: Flowers (Nilsback & Zisserman, 2008), Cars (Krause et al., 2013), Pets (Parkhi et al., 2012), Food (Bossard et al., 2014), CIFAR-10 (Krizhevsky, 2009), and CIFAR-100 (Krizhevsky, 2009). The performance of these models is measured in terms of accuracy.

Implementation Details. Our experiments are based on the Huggingface transformers library[‡]. During training and fine-tuning, we employ the AdamW (Kingma & Ba, 2015) optimizer and a cosine learning rate scheduler. We tune the learning rate on a grid of values: {2e-5, 5e-5, 1e-4, 2e-4, 3e-4}, and report the runs with the highest emulator performance. This is because, in real-world scenarios, users will return the adapter with the best Emulator performance to the model owner. We use `lm-eval-harness`[§] for language model evaluation and NVIDIA A6000 GPUs for all experiments.

5.2 Results of Language Models

Medium-sized models. We first evaluate offsite-tuning on medium-sized language models with less than 2 billion parameters, including GPT-2-XL (Radford et al., 2019) and OPT-1.3B (Zhang et al., 2022). Specifically, GPT-2-XL has 48 layers and 1.6 billion parameters, and OPT-1.3B has 24 layers. On the medium-sized language models (less than 2 billion parameters), we have the compute resource to fine-tune the full model and use knowledge distillation to compress the emulator. We use the top and bottom two transformer layers of the model as the adapter. We reduced the number of layers to 16 and 8 for GPT-2-XL and OPT-1.3B, respectively, as the initialization for emulator distillation. Next, we distilled the emulator on the first of 30 chunks of the Pile corpus training set for a single epoch. The results

[‡]https://github.com/huggingface/transformers
[§]https://github.com/EleutherAI/lm-evaluation-harness

Table 2. Offsite-tuning (OT Plug-in) improves zero-shot (ZS) performance across all tasks without the access to the full model. We observe a consistent performance gap between the emulator and plug-in performance, indicating offsite-tuning effectively preserves the privacy of the original proprietary model.

Setting	OpenBookQA	PIQA	ARC-E	ARC-C	HellaSwag	SciQ	WebQs	RACE	WikiText (↓)
OPT-6.7B (2-18-2 Layer-drop)									
Full ZS	27.6%	76.2%	65.6%	30.6%	50.5%	90.1%	8.8%	38.2%	24.24
Emulator ZS	15.8%	56.0%	33.8%	20.1%	28.3%	51.1%	0.0%	22.8%	44.92
OT Emulator	23.4%	59.4%	45.0%	19.5%	27.9%	74.7%	8.0%	26.7%	13.48
OT Plug-in	33.8%	77.7%	66.8%	33.9%	52.1%	91.9%	23.9%	44.1%	10.78
BLOOM-7.1B (2-12-2 Layer-drop)									
Full ZS	24.8%	72.7%	64.9%	30.3%	46.3%	90.0%	2.3%	36.6%	15.34
Emulator ZS	16.4%	64.0%	44.1%	20.1%	34.1%	78.2%	0.1%	28.2%	78.58
OT Emulator	22.4%	66.4%	52.9%	27.0%	35.9%	85.7%	15.4%	30.3%	22.40
OT Plug-in	29.6%	74.6%	66.9%	36.8%	48.3%	94.2%	25.0%	41.7%	14.58

are reported in Table 1. We find that offsite-tuning can effectively adapt medium-sized language models while maintaining a high level of performance as the plug-in performance is comparable to full model fine-tuning performance, while the emulator performance is significantly lower.

Large models. We then evaluate offsite-tuning on large language models with more than 6 billion parameters, including OPT-6.7B (Zhang et al., 2022) with 32 layers and BLOOM-7.1B (Scao et al., 2022) with 30 layers. Due to the limited computational resources, we are *unable* to perform full model fine-tuning or emulator distillation on these models (model owner should be able to perform the distillation, which costs a small fraction compared to pre-training). As a result, we compare the performance of offsite-tuning with the zero-shot performance and directly use the layer-drop method to get the emulator. We use the top and bottom two transformer layers of the model as the adapter. We dropped the middle 28 and 16 layers to 18 layers and 12 layers for OPT-6.7B and BLOOM-7.1B, respectively. The results are reported in Table 2. From the results, we find that the plug-in performance is significantly better than the zero-shot performance, while there is a noticeable gap between the emulator performance and the plug-in performance. These findings indicate that offsite-tuning can effectively adapt large language models while preserving the privacy of the model and the data owners.

5.3 Results of Vision Models

We further evaluate offsite-tuning on two state-of-the-art vision foundation models: CLIP (Radford et al., 2021) and EVA (Fang et al., 2022). Both models utilize ViT-G backbones with 1 billion parameters. We use the OpenCLIP (Ilharco et al., 2021) checkpoint trained on LAION-2B (Schuhmann et al., 2022)[¶] for CLIP and the EVA-CLIP checkpoint trained on LAION-400M (Schuhmann et al., 2021)[‖] for EVA. We use the top and bottom four transformer layers plus the classification head as the adapter. To initialize the emulator, we reduce the number of layers to 16 and 22 for CLIP and EVA, respectively. We then distill the emulator on ImageNet (Deng et al., 2009) for a single epoch. As shown in Table 3, we find that offsite-tuning effectively adapts the vision models while maintaining a high level of performance. The plug-in performance is comparable to full model fine-tuning performance, while the emulator performance is slightly lower. This may be due to the fact that the difference between using large and small vision models on these datasets is not significant, and thus the emulator performance is not significantly lower than the plug-in performance. We anticipate that offsite-tuning will yield more significant results on more challenging vision tasks.

Table 3. Offsite-tuning also works on vision foundation models. Scores are percent accuracy.

Setting	Flowers	Cars	Food	Pets	CF-10	CF-100
EVA-ViT-G (4-22-4 Distill)						
FT	99.59	95.77	95.43	95.80	99.39	93.96
OT Emulator	99.15	94.58	94.90	96.24	99.52	93.31
OT Plug-in	99.33	94.99	95.33	96.48	99.63	94.21
CLIP-ViT-G (4-16-4 Distill)						
FT	99.33	95.65	94.29	94.99	99.13	92.33
OT Emulator	99.01	95.10	92.91	95.26	99.11	90.47
OT Plug-in	99.27	95.16	93.88	95.53	99.13	91.00

[¶] https://huggingface.co/laion/CLIP-ViT-g-14-laion2B-s12B-b42K

[‖] https://huggingface.co/BAAI/EVA/blob/main/eva_clip_vis_enc_sz224_lincls_86p5.pth

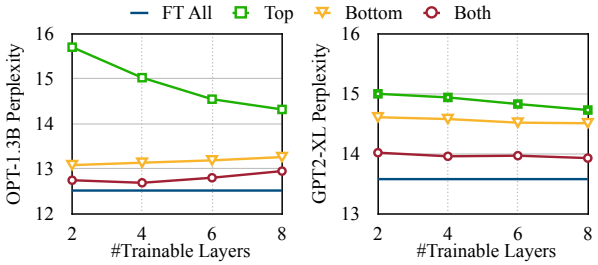

Figure 3. Ablation study of the number and position of adapter layers. Fine-tuning both the top and bottom layers of the language model is significantly more effective than fine-tuning only the top or bottom layers, given the same number of trainable layers.

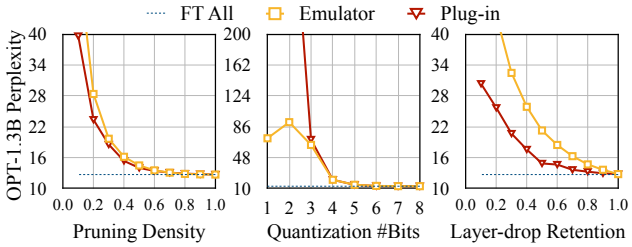

Figure 4. Ablation study of compression methods for creating the emulator. The layer-drop method is superior in two aspects: (1) it effectively maintains the plug-in performance while reducing the size of the emulator; (2) it creates a gap between the plug-in performance and the emulator performance, preserving the privacy of the model owner.

5.4 Ablation Study

5.4.1 Position and Number of Adapter Layers

In Figure 3, we show the effect of the position and number of adapter layers on the fine-tuning performance. We compare the performance of only tuning the top layers (near the output), bottom layers (near the input), and evenly both top and bottom layers while freezing the rest of the layers. We find that given the same budget of trainable layers, evenly tuning both the top and bottom layers significantly outperforms tuning only the top or bottom layers. Additionally, the performance gap between partial fine-tuning and full fine-tuning is not significantly reduced when we increase the number of tunable layers. Based on these results, we chose to use the top and bottom two layers as the adapter throughout our experiments.

5.4.2 Compression Methods for the Emulator

In addition to the layer-drop and distillation methods that we primarily applied in our experiments, we also investigate other compression techniques to create an emulator, including magnitude-based pruning (Han et al., 2016) and quantization (Jacob et al., 2018; Xiao et al., 2022). To construct the emulator, we use two bottom and top layers as the adapter and compress the middle layers with various compression methods. Our results, presented in Figure 4, show that layer-drop achieves the best performance when plugging into the original model. Furthermore, we observe a clear gap between the plug-in and emulator performance when using the layer-drop method, while this gap is not significant when using other compression methods. This suggests that the layer-drop-based compression method for creating the emulator can effectively protect the model's privacy while maintaining a high level of plug-in performance. Overall, our results demonstrate that the layer-drop-based method is an effective approach for creating efficient and privacy-preserving emulators.

Table 4. Distilling the emulator improves offsite-tuning's plug-in performance while maintaining full model privacy. Scores are validation perplexities on WikiText-2 (lower is better).

	w/o Distillation		w/ Distillation	
	Emulator	Plug-in	Emulator	Plug-in
GPT2-XL (2-16-2)	30.48	19.69	17.64	14.94
OPT-1.3B (2-8-2)	18.44	14.63	15.54	13.15

5.4.3 Effect of Emulator Distillation

In Table 4, we show that the emulator distillation can further improve the plug-in performance. Specifically, with the same number of emulator layers, the plug-in performance is improved by 2.47 on the OPT-1.3B model and 4.75 on the GPT-2-XL model. Despite these improvements, we still observe a clear gap between the plug-in and emulator performance, so the privacy of the full model is still well preserved. This suggests that there is further potential to improve the parameter efficiency of the emulator if we have access to more compute resources to perform additional distillation epochs.

5.5 Combining with Parameter-Efficient Fine-tuning

Offsite-tuning is orthogonal and can be seamlessly combined with existing parameter-efficient fine-tuning methods to further reduce the number of trainable parameters for each task. To combine offsite-tuning with parameter-efficient fine-tuning, we only need to apply the method on the adapter layers. We use Adapter-tuning (Houlsby et al., 2019), LoRA (Hu et al., 2021), and BitFit (Ben Zaken et al., 2022) as examples to demonstrate the effectiveness of this combination. We set the adapter size to 64 for Adapter-tuning and rank 4 for LoRA. We perform our experiments on the OPT-1.3B and GPT-2-XL models on the WikiText dataset. As shown in Table 5, we find that Adapter-tuning and LoRA can significantly reduce the number of trainable parameters while maintaining the plug-in performance. However, we also observe that BitFit failed to adapt the

Table 5. Offsite-tuning is orthogonal to and can be combined with parameter-efficient fine-tuning techniques. Scores are validation perplexities on WikiText-2 (lower is better).

	#Trainable Parameters	Emulator PPL	Plug-in PPL
GPT2-XL (2-16-2 Distill)			
FT	1475M	-	13.58
OT	123M	17.64	14.94
+ Adapter	1.65M	17.88	14.99
+ LoRA	410K	18.42	14.84
+ BitFit	83K	21.96	18.75
OPT-1.3B (2-8-2 Distill)			
FT	1208M	-	12.52
OT	201M	15.55	13.15
+ Adapter	2.11M	15.93	13.40
+ LoRA	590K	15.66	13.20
+ BitFit	106K	18.53	17.88

Table 6. Fine-tuning speedup and peak memory saving of offsite-tuning on a single A6000 GPU. Throughputs are in tokens per second and memory is in megabytes (MB). Batch sizes are 1 and sequence lengths are 512.

	Method	Throughput (↑)	Memory (↓)
GPT2-XL	FT	957	39922
	+ LoRA	1992	17113
	OT (2-16-2)	4905	8966
	+ LoRA	**6257**	**7155**
OPT-1.3B	FT	1316	30458
	+ LoRA	3379	12470
	OT (2-8-2)	5207	9336
	+ LoRA	**8525**	**6192**

model as there is a significant gap between the plug-in performance and the full model fine-tuning performance.

5.6 Efficiency

The key efficiency advantage of offsite-tuning is its ability to reduce not only the number of trainable parameters but also the total number of parameters that need to be placed on the device during fine-tuning. This results in a significant increase in fine-tuning throughput and a reduction in memory footprint. To demonstrate the effectiveness of offsite-tuning, we conduct experiments and present the results in Table 6. The results show that when offsite-tuning is combined with LoRA, we achieve an impressive 6.5x speedup and a 5.6x reduction in memory usage. This makes offsite-tuning an attractive solution for fine-tuning large foundation models on resource-constrained devices.

6 Discussion

Use cases. Offsite-tuning is an effective approach for personalizing large language models on edge devices, making it useful for various applications such as voice assistants and chatbots. For example, users can utilize offsite-tuning to adapt a large language model to their personal information directly on their devices, which is more efficient and preserves privacy by eliminating the need to send data to a server. The adapted model can then be used to generate personalized text, such as emails and messages. Additionally, offsite-tuning can be applied in domains where training data is extremely sensitive and cannot be shared, such as in a hospital setting where it can be used to adapt a large language model to patient records without sharing the records with the model owner. The adapted model can then be used to generate personalized medical reports for patients. In both cases above, the method also protects the privacy of the model owner, as they do not need to share their full model with the data owner.

Inference privacy. In this work, we focus on addressing the data privacy and efficiency concerns during the process of adapting or fine-tuning foundation models, rather than addressing privacy issues during inference. The proprietary training datasets for downstream tasks are often well-labeled and contain significant business values, making privacy a crucial consideration. However, privacy concerns during inference can be addressed through other methods such as (Chou et al., 2018; Li et al., 2022).

Limitation and future works. While our results indicate that using approximately one-third of the middle layers of a foundation model as an emulator is viable, it remains huge for models such as GPT-3. Also, compression of the emulator through compute-intensive distillation techniques may be cost-prohibitive for larger models. Furthermore, we have yet to fully demonstrate that our method does not inadvertently result in model and data information leakage. Future research should investigate the possibility of reconstructing the full model and downstream data from the emulator and adapter. Additionally, the theoretical foundation of our method remains unclear and further investigation is needed to provide insights into the emulator and adapter design.

7 Conclusion

We propose offsite-tuning, a privacy-preserving and efficient transfer learning framework that can adapt foundation models to downstream tasks without access to full model parameters. Offsite-tuning is effective on billion-parameter language and vision foundation models. Offsite-tuning enables users to efficiently customize foundation models without worrying about data privacy and model privacy.

Acknowledgements

We thank MIT-IBM Watson AI Lab, MIT AI Hardware Program, Amazon and MIT Science Hub, NVIDIA Academic Partnership Award, Microsoft Turing Academic Program, Qualcomm Innovation Fellowship, and NSF for supporting this research. We thank Tianwei Yin for the helpful discussions.

References

Askari, A., Negiar, G., Sambharya, R., and Ghaoui, L. E. Lifted neural networks. *arXiv preprint arXiv:1805.01532*, 2018.

Augenstein, S., McMahan, H. B., Ramage, D., Ramaswamy, S., Kairouz, P., Chen, M., Mathews, R., and y Arcas, B. A. Generative models for effective ML on private, decentralized datasets. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SJgaRA4FPH.

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, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.1. URL https://aclanthology.org/2022.acl-short.1.

Berant, J., Chou, A., Frostig, R., and Liang, P. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1533–1544, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D13-1160.

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Bossard, L., Guillaumin, M., and Van Gool, L. Food-101 – mining discriminative components with random forests. In Fleet, D., Pajdla, T., Schiele, B., and Tuytelaars, T. (eds.), *Computer Vision – ECCV 2014*, pp. 446–461, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10599-4.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Chou, E., Beal, J., Levy, D., Yeung, S., Haque, A., and Fei-Fei, L. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *arXiv preprint arXiv:1811.09953*, 2018.

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:1803.05457v1*, 2018.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

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, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

Fang, Y., Wang, W., Xie, B., Sun, Q., Wu, L., Wang, X., Huang, T., Wang, X., and Cao, Y. Eva: Exploring the limits of masked visual representation learning at scale. *arXiv preprint arXiv:2211.07636*, 2022.

Han, S., Mao, H., and Dally, W. J. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *ICLR*, 2016.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv: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. URL http://proceedings.mlr.press/v97/houlsby19a.html.

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. *arXiv preprint arXiv:2106.09685*, 2021.

Huo, Z., Gu, B., Huang, H., et al. Decoupled parallel backpropagation with convergence guarantee. In *International Conference on Machine Learning*, pp. 2098–2106. PMLR, 2018.

Ilharco, G., Wortsman, M., Wightman, R., Gordon, C., Carlini, N., Taori, R., Dave, A., Shankar, V., Namkoong, H., Miller, J., Hajishirzi, H., Farhadi, A., and Schmidt, L. Openclip, July 2021. URL https://doi.org/10.5281/zenodo.5143773. If you use this software, please cite it as below.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

Johannes Welbl, Nelson F. Liu, M. G. Crowdsourcing multiple choice science questions. 2017.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Konečnỳ, J., McMahan, H. B., Ramage, D., and Richtárik, P. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

Krause, J., Stark, M., Deng, J., and Fei-Fei, L. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, 2009.

Lai, G., Xie, Q., Liu, H., Yang, Y., and Hovy, E. RACE: Large-scale ReAding comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 785–794, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1082. URL https://aclanthology.org/D17-1082.

Lee, Y., Chen, A. S., Tajwar, F., Kumar, A., Yao, H., Liang, P., and Finn, C. Surgical fine-tuning improves adaptation to distribution shifts. *arXiv preprint arXiv:2210.11466*, 2022.

Li, D., Shao, R., Wang, H., Guo, H., Xing, E. P., and Zhang, H. Mpcformer: fast, performant and private transformer inference with mpc. *arXiv preprint arXiv:2211.01452*, 2022.

Li, J., Fang, C., and Lin, Z. Lifted proximal operator machines. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4181–4188, 2019.

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, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353.

Lin, J., Zhu, L., Chen, W.-M., Wang, W.-C., Gan, C., and Han, S. On-device training under 256kb memory. *arXiv preprint arXiv:2206.15472*, 2022.

Liu, H., Tam, D., Muqeeth, M., Mohta, J., Huang, T., Bansal, M., and Raffel, C. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *arXiv preprint arXiv:2205.05638*, 2022.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR, 2017.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models, 2016.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018.

Muennighoff, N., Wang, T., Sutawika, L., Roberts, A., Biderman, S., Scao, T. L., Bari, M. S., Shen, S., Yong,

Z.-X., Schoelkopf, H., Tang, X., Radev, D., Aji, A. F., Almubarak, K., Albanie, S., Alyafeai, Z., Webson, A., Raff, E., and Raffel, C. Crosslingual generalization through multitask finetuning, 2022.

Ni, Z., Wang, Y., Yu, J., Jiang, H., Cao, Y., and Huang, G. Deep model assembling. *arXiv preprint arXiv:2212.04129*, 2022.

Nilsback, M.-E. and Zisserman, A. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pp. 722–729, 2008. doi: 10.1109/ICVGIP.2008.47.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., and Lowe, R. Training language models to follow instructions with human feedback, 2022. URL https://arxiv.org/abs/2203.02155.

Parkhi, O. M., Vedaldi, A., Zisserman, A., and Jawahar, C. V. Cats and dogs. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3498–3505, 2012. doi: 10.1109/CVPR.2012.6248092.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. 2019.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. Learning transferable visual models from natural language supervision. In *ICML*, 2021.

Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., and Sutskever, I. Robust speech recognition via large-scale weak supervision., 2022. URL https://cdn.openai.com/papers/whisper.pdf.

Sajjad, H., Dalvi, F., Durrani, N., and Nakov, P. On the effect of dropping layers of pre-trained transformer models. *Computer Speech & Language*, 77:101429, 2023.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Scao, T. L., Fan, A., Akiki, C., Pavlick, E., Ilić, S., Hesslow, D., Castagné, R., Luccioni, A. S., Yvon, F., Gallé, M., et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

Schuhmann, C., Vencu, R., Beaumont, R., Kaczmarczyk, R., Mullis, C., Katta, A., Coombes, T., Jitsev, J., and Komatsuzaki, A. Laion-400m: Open dataset of clip-filtered 400 million image-text pairs, 2021. URL https://arxiv.org/abs/2111.02114.

Schuhmann, C., Beaumont, R., Vencu, R., Gordon, C. W., Wightman, R., Cherti, M., Coombes, T., Katta, A., Mullis, C., Wortsman, M., Schramowski, P., Kundurthy, S. R., Crowson, K., Schmidt, L., Kaczmarczyk, R., and Jitsev, J. LAION-5b: An open large-scale dataset for training next generation image-text models. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=M3Y74vmsMcY.

Smith, S., Patwary, M., Norick, B., LeGresley, P., Rajbhandari, S., Casper, J., Liu, Z., Prabhumoye, S., Zerveas, G., Korthikanti, V., et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

Taylor, G., Burmeister, R., Xu, Z., Singh, B., Patel, A., and Goldstein, T. Training neural networks without gradients: A scalable admm approach. In *International conference on machine learning*, pp. 2722–2731. PMLR, 2016.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, X., Wang, W., Cao, Y., Shen, C., and Huang, T. Images speak in images: A generalist painter for in-context visual learning. *arXiv preprint arXiv:2212.02499*, 2022.

Wei, J., Bosma, M., Zhao, V. Y., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

Xiao, G., Lin, J., Seznec, M., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models. *arXiv*, 2022.

Xu, A., Huo, Z., and Huang, H. On the acceleration of deep learning model parallelism with staleness. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2088–2097, 2020.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *CoRR*, abs/1905.07830, 2019. URL http://arxiv.org/abs/1905.07830.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., Mihaylov, T., Ott, M., Shleifer, S., Shuster, K., Simig, D., Koura, P. S., Sridhar, A., Wang, T., and Zettlemoyer, L. Opt: Open pre-trained transformer language models, 2022. URL https://arxiv.org/abs/2205.01068.

Zhang, Z. and Brand, M. Convergent block coordinate descent for training tikhonov regularized deep neural networks. *Advances in Neural Information Processing Systems*, 30, 2017.