Efficiently Distilling LLMs for Edge Applications

Achintya Kundu, Fabian Lim, Aaron Chew, Laura Wynter, Penny Chong, Rhui Dih Lee
IBM Research, Singapore

Abstract

Supernet training of LLMs is of great interest in industrial applications as it confers the ability to produce a palette of smaller models at constant cost, regardless of the number of models (of different size / latency) produced. We propose a new method called Multistage Low-rank Fine-tuning of Super-transformers (MLFS) for parameter-efficient supernet training. We show that it is possible to obtain high-quality encoder models that are suitable for commercial edge applications, and that while decoder-only models are resistant to a comparable degree of compression, decoders can be effectively sliced for a significant reduction in training time.

1 Introduction

Given their sizes up to billions of parameters, (Raffel et al., 2020; Brown et al., 2020), it is challenging for enterprises to fine-tune Large Language Models (LLMs), and furthermore they are not suitable for deployment on edge devices with limited memory and computational power. We wish to enable LLMs on edge environments for enterprise use cases. This requires the following two capabilities. (1) Accommodating a variety of edge device hardware: A single fine-tuned model is not optimal across the spectrum of devices. For industrial applications, a palette of fine-tuned LLMs is required for different hardware. (2) Dynamically changing resource levels: At run-time, the available resources on edge devices evolve over time, and appropriate model should be dynamically selected based on the available resources of each device.

A considerable amount of research has focused on compressing LLMs (Zhu et al., 2023; Sanh et al., 2019; Mukherjee and Awadallah, 2020; Mukherjee et al., 2021; Jiao et al., 2020; Hsieh et al., 2023). Methods that train a single small model guided by a large teacher model such as DistilBERT (Sanh et al., 2019) and BERT-PKD (Sun et al., 2019), either achieve limited compression or do not scale to a large number of deployment devices. Supernet training methods (Hou et al., 2020; Xu et al., 2021; Cai et al., 2019; Kundu et al., 2023; Lou et al., 2021; Jawahar et al., 2023) were introduced to address these limitations: multiple smaller subnets within the supernet are trained simultaneously with weight-sharing. This one-time training approach produces a palette of smaller models, helping mitigate the computational cost of fine-tuning a model for each deployment scenario. However, the full-parameter supernet training approach is impractical when fine-tuning of an LLM is required for multiple deployment scenarios, limiting its utility for enterprises.

Parameter-efficient fine-tuning (PEFT) methods such as Low-Rank Adaptation (LoRA) reduces the number of trainable parameters by allowing only rank-decomposition matrices to be trained while freezing the pre-trained weights of the model. PEFT methods, however, are not applicable to supernet training due to the implications on the weight-shared sub-networks. Our work bridges this gap to enable efficient fine-tuning of LLMs for edge devices. Our contributions are:

1. We propose a parameter-efficient, distillation-based approach for supernet training of LLMs.
2. We devise a gradient scaling scheme to improve convergence speed of any form of supernet training.
3. We demonstrate significant compression of encoder models for edge. We highlight the limits of comparable compression for decoder models, while demonstrating a huge reduction in the steps needed for convergence.

2 Related Work

Classical compression methods have been used for LLMs including pruning (McCarley et al., 2019; Voita et al., 2019), low rank approximation (Ma

Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track), pages 52–62

et al., 2019; Lan et al., 2019), and quantization (Shen et al., 2020; Zafrir et al., 2019; Bhandare et al., 2019). Knowledge distillation (KD) is adopted in BERT-PKD (Sun et al., 2019), tinyBERT (Jiao et al., 2020), and distilBERT (Sanh et al., 2019) and (Gu et al., 2023) in MiniLLM to distill knowledge from the layers of a large transformer model to a smaller one. See also the survey (Zhu et al., 2023). All these existing methods produce a single compressed model, unsuitable for edge scenarios with multiple deployment devices having varying computational capability.

Neural architecture search (NAS) based on reinforcement learning (Zoph and Le, 2016) and evolutionary algorithms (Real et al., 2019; Zhu et al., 2019) trains every possible architecture and is very slow. Weight-sharing NAS was thus developed: in Guo et al. (2020); Cai et al. (2018), the building blocks in the same layer are isolated as all architectures are single paths. Weight-sharing NAS does not scale well to large architecture search spaces, hence, weight-entangled NAS, where subnets with common parts share weights, was introduced.

For resource-constrained edge deployment, supernet training (Cai et al., 2019; Kundu et al., 2023; Chen et al., 2021b; Xu et al., 2021; Gao et al., 2022; Dong et al., 2022) was developed as a mode of jointly training multiple sub-networks (subnets) with entangled weights: one trains the supernet only once for all deployment scenarios. Cai et al. (2019) introduced an elastic convolutional neural network with "progressive shrinkage", where larger subnets are trained first. Recent works have improved sampling strategies, e.g. the sandwich rule with in-place distillation (Yu et al., 2020), attentive sampling (Wang et al., 2021), stochastic nature gradient (Zhang et al., 2021), or post-training sampling (Lou et al., 2021). Our work is related to supernet training for transformer models (Hou et al., 2020; Zhang et al., 2021; Wang et al., 2022, 2020; Chen et al., 2021b). This gradient scaling technique can be used with any of the above supernet methods.

Parameter-efficient fine-tuning (PEFT) has been of great benefit in fine tuning LLMs. BitFit (Ben Zaken et al., 2022) updates the bias terms in pre-trained models while freezing the remaining parameters. LoRA (Hu et al., 2022) decomposes attention weight gradients into low-rank matrices to reduce the number of trainable parameters. AdaLoRA (Zhang et al., 2023) and QLoRA (Dettmers et al., 2023) further improve LoRA (Hu et al., 2022). Note that PEFT allows fine-tuning a base model on a single GPU but does not produce smaller models. None of the PEFT methods can be used for weight-sharing supernet training.

3 Solution Design

For use in enterprise settings, the solution must allow fine-tuning of models on a small GPU footprint. In addition, inference cost in terms of storage must be minimised. We therefore design a solution which does not store the full size model checkpoint for every downstream task but only the frozen weights of the pre-trained base model and the low rank matrices. For inference in commercial edge use cases, we wish to enable storing the desired models locally for a wide variety of edge device resource requirements. We thus develop an approach where storage is minimised, storing only one base model and as many low rank adapter matrices as there are target model size variations, where low-rank adapters are very small. If the model is stored locally on an edge device, our proposed slicing operation takes place where the supernet fine-tuning is performed and the desired model is downloaded for inference. The slicing operation takes place for each model size-task combination and each resulting subnet can be cached for inference.

4 Problem Formulation

First, we provide notation. Given a transformer model with architectural configuration Φ and weights W, we denote its forward-pass mapping by $\mathrm{f}_\Phi(\cdot;\mathrm{W}) : \mathcal{X} \rightarrow \mathcal{Y}$. We consider the output space $\mathcal{Y}$ to be the set of all non-negative vectors in $\mathbb{R}^\nu$ with elements summing to 1, where ν denotes the number of classes / vocabulary size). With slight abuse of notation, we write the forward-pass mapping of an input $x \in \mathcal{X}$ through a transformer model Φ as $\hat{y}, z, \mathbf{h} = \mathrm{f}_\Phi(x;\mathrm{W})$, where $\hat{y} \in \mathcal{Y}$ denotes the predicted probability distribution over the (class labels) vocabulary, z denotes the vector of logits, and $\mathbf{h}$ represents a tuple of features such as hidden state vectors and attention values from different transformer layers. Note that $\hat{y} = \boldsymbol{\sigma}(z)$, where $\boldsymbol{\sigma}$ is the standard soft-max function that maps a vector of logits into a probability vector. Given a training data set $\mathcal{D}_{train} \subset \mathcal{X} \times \mathcal{Y}$, model weights W are learnt by minimizing training loss:

$$\underset{\mathrm{W}}{\operatorname{argmin}} \Big[\mathcal{L}_\Phi(\mathrm{W}) := \mathbb{E}\big[\, \ell\big[\, \mathrm{f}_\Phi(x;\mathrm{W}), y \,\big]\,\big] \Big], \quad (1)$$

where $\mathbb{E}$ denotes expectation over training example (x, y) drawn uniformly at random from $\mathcal{D}_{train}$ and

ℓ denotes a loss function. Most commonly, ℓ is chosen to be a task specific loss function, ℓ_{task}, such as cross-entropy (i.e., $\mathsf{CE}[\cdot,\cdot]$) for classification or causal language modeling loss for generative models.

Next, we introduce the super-transformer and related terminologies. We define three types of networks - *Teacher network*, *Super-transformer* (supernet) and *Sub-transformer* (subnet). The teacher is a fixed network with the same configuration as the pre-trained transformer. A super-transformer is a dynamic model whose architectural dimensions (embedding dimension, number of heads, number of layers, etc.) are configurable at run time. The *maxnet* (resp. *minnet*) is the largest (resp. smallest) network in the super-transformer's architecture space. Weight entanglement (weight-sharing) allows super-transformer weights to be used across sub-transformers, which are subsets of the super-transformer. Pre-trained transformer weights initialise the super-transformer.

The dynamic nature of a super-transformer is explicitly specified via a set $\mathcal{A}$, called *configuration space*, consisting of architectural configurations of all sub-transformer models under consideration. The definition of a super-transformer also includes how the configuration $\Phi \in \mathcal{A}$ is to be mapped to a unique transformer model f_Φ. A weight-sharing super-transformer uses a set of shared weights $\mathrm{W}_{\mathsf{Sup}}$ to define all sub-transformer models' weights. This is done through a weight projection operator $\mathbf{\Pi}$ that slices (selects an appropriate subset of) the super-transformer's weights $\mathrm{W}_{\mathsf{Sup}}$ into weights of a sub-transformer model:

$$\mathrm{W}_\Phi := \mathbf{\Pi}_\Phi(\mathrm{W}_{\mathsf{Sup}}), \forall \Phi \in \mathcal{A}. \tag{2}$$

The aim of a weight-sharing super-transformer is to simultaneously train all the transformer models $\{\mathrm{f}_\Phi(\cdot;\mathbf{\Pi}_\Phi(\mathrm{W})) : \mathcal{X} \to \mathcal{Y} \,|\, \Phi \in \mathcal{A}\}$ through the shared weights $\mathrm{W}_{\mathsf{Sup}}$. A typical training objective for super-transformers is the training loss averaged over all model configurations in $\mathcal{A}$:

$$\underset{\mathrm{W}_{\mathsf{Sup}}}{\operatorname{argmin}} \Big[\mathcal{L}_{\mathsf{Sup}}(\mathrm{W}_{\mathsf{Sup}}) := \mathbb{E}\big[\mathcal{L}_\Phi\big(\mathbf{\Pi}_\Phi(\mathrm{W}_{\mathsf{Sup}})\big)\big]\Big], \tag{3}$$

where $\mathbb{E}$ denotes expectation over model configuration Φ drawn uniformly at random from $\mathcal{A}$ and $\mathcal{L}_\Phi$, as defined in (1), is averaged training loss for configuration Φ. Super-transformer weights, $\mathrm{W}_{\mathsf{Sup}}$, are learnt with stochastic gradient (denoted $\hat{\nabla}$) of the super-transformer's loss $\mathcal{L}_{\mathsf{Sup}}$ estimated as

$$\hat{\nabla}_{\mathrm{W}}\mathcal{L}_{\mathsf{Sup}}(\mathrm{W}_{\mathsf{Sup}}) = \frac{1}{K}\sum_{j=1}^{K}\hat{\nabla}_{\mathrm{W}}\mathcal{L}_{\Phi_j}\big(\mathbf{\Pi}_{\Phi_j}(\mathrm{W}_{\mathsf{Sup}})\big), \tag{4}$$

$$\hat{\nabla}_{\mathrm{W}}\mathcal{L}_\Phi(\mathrm{W}_\Phi) = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\nabla_{\mathrm{W}}\ell\big[\mathrm{f}_\Phi\big(x^i;\mathrm{W}_\Phi\big), y^i\big], \tag{5}$$

where $\{\Phi_1, \cdots, \Phi_K\}$ are K sub-transformer configurations sampled from $\mathcal{A}$ to approximate the expectation in (3) and $\mathcal{B}$ is a mini-batch of training examples sampled from $\mathcal{D}_{train}$ to approximate the expectation in (1). Fine-tuning LLM super-transformers is computationally challenging in enterprise use cases as it involves computing gradients of sub-transformers' loss functions with respect to a huge number of parameters.

5 MLFS

We therefore developed Multistage Low-rank Fine-tuning of Super-transformers (MLFS). Given a teacher model with configuration Φ_{Tch} and pre-trained weights $\mathrm{W}_{\mathsf{Tch}}^{\mathrm{pretrain}}$, we assume that its weights (denoted $\mathrm{W}_{\mathsf{Tch}}$) can be fine-tuned on the given task by learning low-rank matrices A_0, B_0 on top of pre-trained weights $\mathrm{W}_{\mathsf{Tch}}^{\mathrm{pretrain}}$:

$$\mathrm{W}_{\mathsf{Tch}} := \mathrm{W}_{\mathsf{Tch}}^{\mathrm{pretrain}} + A_0 * B_0, \tag{7}$$

where A_0, B_0 are of (low) rank r. Note that pre-trained weights $\mathrm{W}_{\mathsf{Tch}}^{\mathrm{pretrain}}$ remain unchanged during super-transformer fine-tuning. The low-rank matrices, A_0 and B_0, are learnt by minimizing the cross-entropy loss of the teacher model $\mathrm{f}_{\Phi_{\mathsf{Tch}}}(\cdot;\mathrm{W}_{\mathsf{Tch}}) : \mathcal{X} \to \mathcal{Y}$ over the training data set $\mathcal{D}_{train}$. Specifically, we perform E_0 epochs of fine-tuning on the teacher to learn A_0, B_0. This is stage-0 of the multistage fine-tuning algorithm. We denote the teacher weights obtained at the end of stage-0 by $\mathrm{W}_{\mathsf{Tch}}$. We now define a super-transformer with maxnet configuration the same as the teacher's. Thus the super-transformer's weights $\mathrm{W}_{\mathsf{Sup}}$ are of the same size as the teacher weights $\mathrm{W}_{\mathsf{Tch}}$). To fine-tune the super-transformer weights $\mathrm{W}_{\mathsf{Sup}}$, in each of the subsequent stages, we freeze $\mathrm{W}_{\mathsf{Tch}}$ and propose learning two stage-specific low-rank matrices A_s, B_s, of the same rank, r, as A_0, B_0, that are shared across all sub-transformer models in that stage. To be precise, we impose the following structure on the weights of the sub-transformers at stage-s:

$$\begin{aligned} &\mathrm{W}_{\mathsf{Sup}} := \mathrm{W}_{\mathsf{Tch}} + \textstyle\sum_{s=1}^{2} A_s * B_s, \\ &\mathrm{W}_\Phi = \mathbf{\Pi}_\Phi(\mathrm{W}_{\mathsf{Sup}}),\ \forall \Phi \in \mathcal{A}. \end{aligned} \tag{8}$$

Algorithm 1 Multistage Low-rank Fine-tuning of Super-transformers (MLFS)

Input: Transformer model (teacher) with configuration Φ_{Tch} & off-the-shelf pre-trained weights $\mathrm{W}_{\mathsf{Tch}}^{\mathrm{pretrain}}$, model configuration space $\mathcal{A}$ consisting of smaller (than Φ_{Tch}) transformer architectures of interest, $\mathcal{D}_{train}$: fine-tuning data set for the target task, r: rank of the low-rank matrices and distillation factor $\alpha \in [0, 1]$.
Loss functions: Target task loss ℓ_{task}, knowledge distillation loss ℓ_{KD}, feature distillation loss ℓ_{FD}.
Multistage Training:

for stage $s = 0, 1, 2$ **do**
  Initialize the low-rank matrices $\{A_s, B_s\}$ to be learned at stage s .
  for $\mathtt{iteration} = 1, ...$ **do**
    Get a mini-batch $\mathcal{B}$ of training examples from data set $\mathcal{D}_{train}$: $\{(x^i, y^i) \in \mathcal{D}_{train} \,|\, i \in \mathcal{B}\}$.
    Load the super-transformer model with weights $\mathrm{W}_{\mathsf{Sup}} \leftarrow \mathrm{W}_{\mathsf{Tch}}^{\mathrm{pretrain}} + \sum_{l=0}^{s} A_l * B_l$.
    $\mathcal{A}_s := \{\Phi_1, \Phi_2, \cdots\} \leftarrow$ **sample_sub-transformers**$(\mathcal{A}, \text{stage} = s)$. [$\Phi_1$ is the maxnet].
    for each $\Phi_j \in \mathcal{A}_s$ **do**
      Load the sub-transformer model Φ_j with weights $\mathrm{W}_{\Phi_j} := \mathbf{\Pi}_{\Phi_j}(\mathrm{W}_{\mathsf{Sup}})$.
      $n_j :=$ # of fine-tuning weights in model configuration Φ_j .
      Compute forward-pass on the sub-transformer Φ_j: $\hat{y}_j^i, z_j^i, \mathbf{h}_{\Phi_j}^i \leftarrow \mathrm{f}_{\Phi_j}(x^i; \mathrm{W}_{\Phi_j}), \forall i \in \mathcal{B}$.
      For the case of maxnet (Φ_1) set the distillation factor α to 0.
      Find the loss: $\mathtt{loss}_j^i \leftarrow (1-\alpha)\, \ell_{\mathsf{task}}[\hat{y}_j^i, y^i] + \alpha \left(\ell_{\mathsf{KD}}[z_j^i, z_1^i] \,+\, \ell_{\mathsf{FD}}[\mathbf{h}_{\Phi_j}^i, \mathbf{h}_{\Phi_1}^i] \right), \forall i \in \mathcal{B}$.
      Compute gradients $(\nabla_{A_s} \mathtt{loss}_j^i, \nabla_{B_s} \mathtt{loss}_j^i)$ using backward-pass on sub-transformer Φ_j.
    end for
    Update A_s, B_s using the gradients $(\hat{\nabla}_{A_s} \mathcal{L}_{\mathsf{Sup}}, \hat{\nabla}_{B_s} \mathcal{L}_{\mathsf{Sup}})$ of the super-transformer's loss:

$$\hat{\nabla}_{\mathrm{W}} \mathcal{L}_{\mathsf{Sup}} = \frac{1}{|\mathcal{A}_s|} \sum_{\Phi_j \in \mathcal{A}_s} \left(\frac{n_1}{n_j} \right)^{\gamma} \hat{\nabla}_{\mathrm{W}} \mathcal{L}_{\Phi_j}, \;\; \hat{\nabla}_{\mathrm{W}} \mathcal{L}_{\Phi_j} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_{\mathrm{W}} \mathtt{loss}_j^i, \;\; \forall\, \mathrm{W} \in \{A_s, B_s\}. \quad (6)$$

  end for
end for
Output: $\{A_s, B_s\}_{s=0}^{2}$ and fine-tuned super-transformer weights: $\mathrm{W}_{\mathsf{Sup}} = \mathrm{W}_{\mathsf{Tch}}^{\mathrm{pretrain}} + \sum_{s=0}^{2} A_s * B_s$.

Stage-s of the fine-tuning process involves learning only the low-rank matrices, A_s, B_s, by minimizing the super-transform loss as in (3). In stage-1, we sample sub-transformer models by sampling different widths from the super-transformer keeping the depth (number of layers) same as the maxnet. In stage-2, we sample sub-transformer models by sampling different widths as well as depths. We always sample the maxnet model from the super-transformer as the 1^{st} sub-transformer model, Φ_1, at every iteration. We call this **M**ultistage **L**ow-rank **F**ine-tuning of **S**uper-transformers (MLFS) and present it in Algorithm 1.

Proposition 1 *Let the individually fine-tuned weights of a subnet, Φ, be expressed as $\mathrm{W}_{\Phi} = \mathbf{\Pi}_{\Phi}(\mathrm{W}_{\mathsf{Tch}}^{\mathrm{pretrain}}) + \Delta\mathrm{W}_{\Phi}$. Then, MLFS has the following structure on $\Delta\mathrm{W}_{\Phi}$:*

$$\Delta\mathrm{W}_{\Phi} = \mathbf{\Pi}_{\Phi} \left(\textstyle\sum_{s=0}^{2} A_s * B_s \right), \; \forall \Phi \in \mathcal{A}, \quad (9)$$

where $\{A_s, B_s\}_{s=0,1,2}$ are low-rank matrices shared across all sub-transformers $\Phi \in \mathcal{A}$.

To illustrate the computational savings, recall $\mathrm{W}_{\mathsf{Tch}}^{\mathrm{pretrain}} \in \mathbb{R}^{d \times d}$, where d is typically of the order $10^4 - 10^6$. For rank r (typically < 10) for the low-rank matrices: $A_s \in \mathbb{R}^{d \times r}$, $B_s \in \mathbb{R}^{r \times d}$, $s = 0, 1, 2$, where $r \ll d$. Then, the number of parameters to be learnt in the MLFS approach is $6rd$. In contrast, full fine-tuning requires updating d^2 parameters at every iteration.

Gradient Scaling For faster convergence of the smaller sub-transformers within a super- transformer, we propose a novel weighted-combination of the gradients of the sampled sub-transformers.

Proposition 2 *Let 1^{st} sampled sub-transformer, Φ_1, be the maxnet be in every iteration. Then the scaled gradient of the super-transformer training*

loss, $\mathcal{L}_{\mathsf{Sup}}$, in Algorithm 1 is given by

$$\sum_{j=1}^{K}(n_1/n_j)^{\gamma}\,\nabla_{\mathrm{W}}\mathcal{L}_{\Phi_j}, \tag{10}$$

where ∇_{W} denotes gradient w.r.t. only those weights that are being fine-tuned (in this case only the LoRA matrices), n_j denotes the actual number of trainable weights in model configuration Φ_j and $\gamma \geq 1$ is a hyper-parameter.

Proof: Each sub-transformer gradient in (10), $\mathtt{grad}^j$, is scaled by (n_1/n_j), which is obtained from the relative weighting of the loss functions. Let $\mathcal{L}_j(\mathrm{W})$ denote the j-th sub-transformer's loss. Using first-order Taylor expansion, we get:

$$\mathcal{L}_{\Phi_j}(\mathrm{W}+\boldsymbol{\delta}) \approx \mathcal{L}_{\Phi_j}(\mathrm{W}) + \langle \nabla_{\mathrm{W}}\mathcal{L}_{\Phi_j}(\mathrm{W}), \boldsymbol{\delta}\rangle,$$

where $\langle\cdot,\cdot\rangle$ denotes inner (dot) product operation. Therefore, the steepest possible decrease in the loss function $\mathcal{L}_{\Phi_j}$ can be approximated as:

$$\Delta\mathcal{L}_{\Phi_j} \approx \|\nabla_{\mathrm{W}}\mathcal{L}_{\Phi_j}(\mathrm{W})\|_1\,|\boldsymbol{\delta}|_{max} \approx O(n_j)|\boldsymbol{\delta}|_{max},$$

where we approximate the $\|\cdot\|_1$ norm using the zero-th norm, i.e., number of non-zero elements and n_j stands for the actual number of trainable parameters in sub-transformer configuration Φ_j. Since the decrease in the loss of a sub-transformer model Φ_j is approximately proportional to the number of trainable model parameters (n_j), we scale the losses using $(n_1/n_j)^{\gamma}, \gamma \geq 1$ so that training losses of smaller sub-transformer models converge at a rate similar to that of larger sub-transformer configurations. Recall that n_1 is the maximum number of trainable parameters as 1^{st} sampled sub-transformer Φ_1 is always the maxnet. □

Distillation Loss for Super-transformers: Knowledge distillation is straightforward in a fixed-network fine-tuning setting. However, it is less so when fine-tuning a supernet, and in particular, fine-tuning a supernet using the proposed multistage LoRA based approach. Specifically, the subnets receive two types of knowledge distillation (KD) from the teacher: (a) the usual KD loss that utilizes the output logits of the teacher and (b) distillation of features from transformer layers (Jiao et al., 2020) of the teacher.

To define the distillation based losses precisely, let the forward-pass mapping of an input training sample x^i through sub-transformer Φ_j be $\hat{y}^i_j, z^i_j, \mathbf{h}^i_{\Phi_j} \leftarrow \mathrm{f}_{\Phi_j}(x^i; \mathrm{W}_{\Phi_j})$, where $\mathbf{h}^i_j := (\mathbf{h}^{i,1}_j, \ldots, \mathbf{h}^{i,l}_j, \ldots)$ with $\mathbf{h}^{i,l}_j$ denoting the feature vector from l-th layer of sub-transformer Φ_j. In super-transformers, the model (maxnet) having the largest configuration, Φ_1, acts as the teacher and knowledge distillation loss for all other sub-transformers w.r.t the teacher is defined as

$$\ell_{\mathsf{KD}}[z^i_{\Phi_j}, z^i_{\Phi_1}] = \mathsf{KL}[\,\boldsymbol{\sigma}(z^i_{\Phi_j}/t),\, \boldsymbol{\sigma}(z^i_{\Phi_1}/t)\,],\ \forall j > 1,$$

where $\mathsf{KL}[\cdot,\cdot]$ denotes the standard KL divergence between two probability vectors, and $t \geq 1$ is a hyper-parameter called the temperature. Let d_j denote the embedding dimension (hidden size) in sub-transformer Φ_j. We compute feature based distillation loss by projecting features $\mathbf{h}^{i,l}_{\Phi_j} \in \mathbb{R}^{d_j}$ to a low-dimensional space $\mathbb{R}^{d_{\mathrm{low}}}$:

$$\ell_{\mathsf{FD}}[\mathbf{h}^i_{\Phi_j}, \mathbf{h}^i_{\Phi_1}] = \sum_l \beta^l_j\, \|\mathbf{U}^l_j \mathbf{h}^{i,l}_{\Phi_j} - \mathbf{U}^l_1 \mathbf{h}^{i,g_j(l)}_{\Phi_1}\|_2^2,$$

where g_j maps each layer index of the sub-transformer configuration Φ_j to that of the super-transformer (/ maxnet Φ_1). In this paper, we propose to share the maxnet's feature projection matrices $\{\mathbf{U}^l_1 \in \mathbb{R}^{d_{\mathrm{low}} \times d_1}\}$ across all sub-transformer models. We do so by slicing the matrices $\{\mathbf{U}^l_1\}$:

$$\mathbf{U}^l_j \;:=\; [\,\mathbf{U}^{g_j(l)}_1\,]_{\Phi_j} \in \mathbb{R}^{d_{\mathrm{low}} \times d_j}, \tag{11}$$

where the operation $[\ ]_{\Phi_j}$ selects appropriate subset of columns depending on the configuration Φ_j. To reduce the number of user-chosen hyper-parameters, we propose the following hyper-parameter sharing: $\beta^l_j := \beta_{g_j(l)}, \forall j, l = 1, 2, \ldots$. Thus, apart from setting fewer hyper-parameters, one needs to learn only maxnet's feature projection matrices $\{\mathbf{U}^l_1 : l = 1, 2, \ldots\}$, making feature distillation in a super-transformer setting computationally efficient. Additionally, we save computation through use of features only from a fixed subset of maxnet layers for distillation across all sub-transformers: i.e., we use the following subset of maxnet layers: $\{\, g_{\min}(l) \;:\; l = 1, \ldots, L_{\min}\}$, where $L_{\min}$ denotes the number of transformer layers in the smallest sub-transformer $\Phi_{\min}$ and $g_{\min}$ maps layer indices of $\Phi_{\min}$ to that of maxnet Φ_1.

6 Results on Encoder and Decoder LLMs

We report performance on encoder tasks using GLUE (Wang et al., 2018) with BERT$_{base}$ as the teacher model Φ_{Tch}. For decoder LLMs, we use Santacoder (Allal et al., 2023) and Codellama7B (Rozière et al., 2023) on a python coding task using *bigcode/the-stack* data (Kocetkov et al., 2022). We

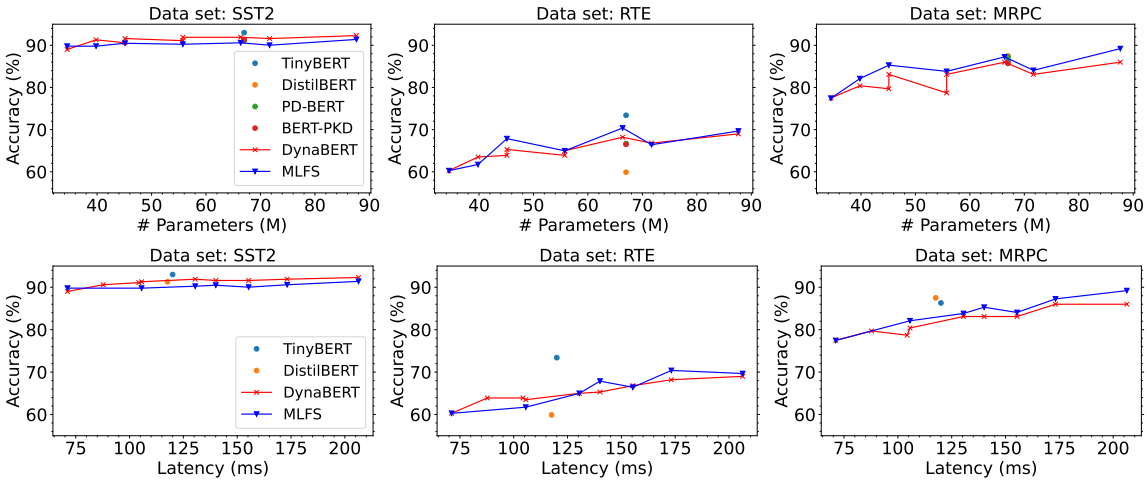

Figure 1: Performance of task-specific BERT models produced by MLFS vs. other methods on 3 GLUE data sets.

report performance of the sub-transformer models at the end of stage $s = 2$. On GLUE, we use the train set for fine-tuning and the dev set for accuracy evaluation. For santacoder, we evaluate performance using HumanEval (Chen et al., 2021a) and report pass@1 scores. All experiments were conducted using PyTorch on a single Nvidia A100 (40GB) GPU. Additional details on the experiment settings are provided in the Appendix.

6.1 Performance of Encoder Models

We compare performance of encoder models obtained with the MLFS approach against a static, fixed model (BERT base) from (Zhang et al., 2021; Hou et al., 2020), two popular distilled variants of the fixed model: TinyBERT (Jiao et al., 2020) and DistilBERT (Sanh et al., 2019), and models trained using existing super-transformer methods (DynaBERT (Hou et al., 2020). Figure 1 shows the performance of the palette of models, from a 45M param. minnet to full-size 110M maxnet. Encoder models produced by MLFS are at par or better than much costlier methods. Results of PD-BERT, BERT-PKD are from (Zhang et al., 2021), static BERT from (Zhang et al., 2021) for all except MRPC for which we use (Hou et al., 2020). Note that TinyBERT performs data augmentation leading to higher accuracy but much longer computation time. We do not perform data augmentation for fairness of the comparison to the other methods. The main observation is that MLFS provides accurate, smaller encoder models at 1/4 the size of the teacher and 1/3 its runtime latency on a single GPU.

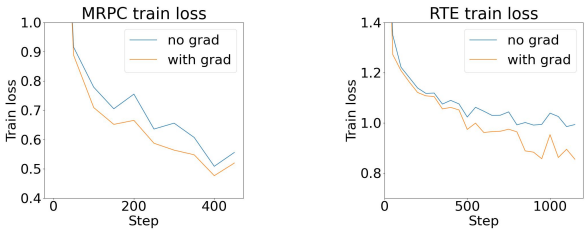

Figure 2: Ablation study on gradient scaling: MLFS minnet convergence is improved using gradient scaling.

Ablation Study on Gradient Scaling In supernet training, the weights of maxnet and subnets are shared and trained simultaneously. The maxnet tends to converge and overfit earlier than smaller subnets. The different convergence rates renders selecting a single supernet checkpoint for all networks difficult. Gradient scaling solves this by speeding up convergence of the smaller subnets to match that of the larger subnets or the maxnet. Fig. 2 shows that gradient scaling improves minnet convergence, indicated by lower minnet loss.

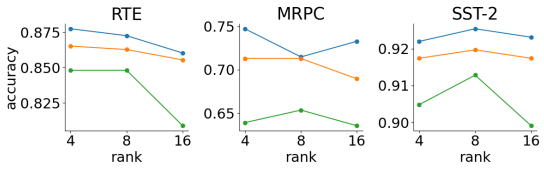

Figure 3: Ablation study on MLFS rank of A, B. Maxnet (top: blue), minnet (bottom: green), and average of two medium-sized subnets (middle: orange). Rank $r = 8$ is optimal for small and medium subnets.

Ablation Study on Rank in MLFS Finally, in Fig. 3, we examines the impact of rank r of the

matrices A, B on performance. Note that the actual number of parameters fine-tuned vary as we vary the rank r. The aim is to provide good results for the smaller networks. Here, rank $r = 8$ works well across the GLUE data sets. Therefore, we use rank $r = 8$ for A, B for all other MLFS experiments. From the scale of the y-axis in 3, observe that MLFS is not overly sensitive to the chosen rank.

6.2 Performance of Decoder Models

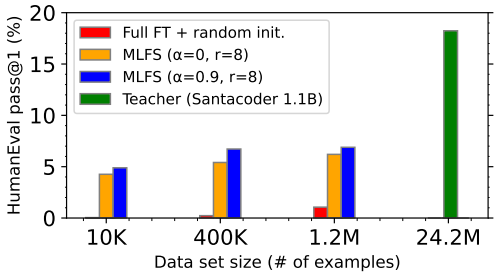

Figure 4: Performance of MLFS on a custom Santacoder 0.7B model using 10K/400K/1.2M training examples.

Data set size	Model size		
	0.5B	**0.7B**	**0.9B**
10K	4.5	8.6	13.4
400K	4.7	9.5	13.5

Table 1: HumanEval pass@1 (%) performance of 3 small models produced by MLFS from Santacoder 1.1B.

Data set size	Model size		
	4.5B	**5.3B**	**6B**
200K	11.0	19.5	23.2
400K	14.0	28.1	30.5

Table 2: HumanEval pass@1 (%) performance of 3 small models produced by MLFS from CodeLlama-7B-Python

Turning now to decoder models, we consider two code-pre-trained LLMs, Santacoder (Allal et al., 2023) and Codellama7B (Rozière et al., 2023). We evaluate a custom 0.7B parameter Santacoder model obtained from the 1.1B teacher. Due to an inability to fine-tune on the full 24M coding examples, we use up to 1.2M. Fig. 4 shows that MLFS pass@1 improves rapidly as number of tokens increases from a low 10k to 400k to 1.2M examples, only 5% of the 24M examples. Table 1 shows analogous results with 3 small MLFS models. The

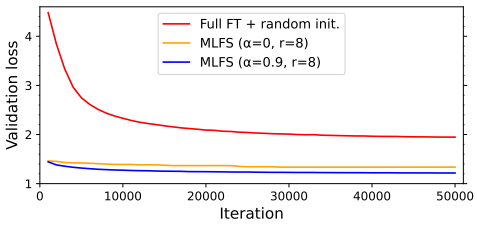

Figure 5: Convergence comparison of validation loss while fine-tuning a custom model from random vs using MLFS. MLFS achieves low validation loss much faster.

improvement in pass@1 indicates that the smaller models retain the ability to learn from the larger teacher. Again, from Table 2, we see that smaller models produced by MLFS from *CodeLlama-7B-Python* retain their ability to learn and improve quickly as the number of examples increases. Note that the full data set includes 24M examples; MLFS achieves nearly 75% of the performance of fullsize CodeLlama after less than 2% of the examples.

Contrary to encoder models, the compression levels that retain sufficient performance of the teacher with decoders is less. While MLFS retains accuracy performance of encoder models at 1/4 the size of the teacher, the decoder models are reduced to at most 2/3 the teacher's size.

MLFS slicing of the teacher model can, however, benefit decoder models by reducing substantially the training/fine-tuning time needed compared to a randomly-initialised model, as shown in Fig. 5 on Santacoder sliced from 1.1B to 0.7B. In other words, when a smaller model is required for edge inference, one can train it from a random initialisation, or slice from a teacher as does MLFS, and train starting from the sliced weights. The latter significantly reduces training time as seen in the validation loss curves. See (Samragh et al., 2023) for a similar observation.

7 Perspectives

Enterprise users require an efficient way to fine-tune LLMs for inference on edge devices of many sizes. We developed MLFS for such edge deployment scenarios. We demonstrate its benefits on encoder LLMs. We show the limitation of compressing decoder LLMs to a comparable degree; however, MLFS offers significant gains for smaller decoder training/fine-tuning by slicing from a larger pre-trained teacher.

References

Loubna Ben Allal, Raymond Li, Denis Kocetkov, Chenghao Mou, Christopher Akiki, Carlos Munoz Ferrandis, Niklas Muennighoff, Mayank Mishra, Alex Gu, Manan Dey, et al. 2023. Santacoder: don't reach for the stars! *arXiv preprint arXiv:2301.03988*.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. 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)*, pages 1–9, Dublin, Ireland. Association for Computational Linguistics.

Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram Saletore. 2019. Efficient 8-bit quantization of transformer neural machine language translation model. *arXiv preprint arXiv:1906.00532*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2019. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*.

Han Cai, Ligeng Zhu, and Song Han. 2018. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021a. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. 2021b. Autoformer: Searching transformers for visual recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 12270–12280.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*.

Peijie Dong, Xin Niu, Lujun Li, Linzhen Xie, Wenbin Zou, Tian Ye, Zimian Wei, and Hengyue Pan. 2022. Prior-guided one-shot neural architecture search. *arXiv preprint arXiv:2206.13329*.

Jiahui Gao, Hang Xu, Han Shi, Xiaozhe Ren, LH Philip, Xiaodan Liang, Xin Jiang, and Zhenguo Li. 2022. Autobert-zero: Evolving bert backbone from scratch. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36-10, pages 10663–10671.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2023. Knowledge distillation of large language models. *arXiv preprint arXiv:2306.08543*.

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2020. Single path one-shot neural architecture search with uniform sampling. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*, pages 544–560. Springer.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2020. Dynabert: Dynamic bert with adaptive width and depth. *Advances in Neural Information Processing Systems*, 33:9782–9793.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. *arXiv preprint arXiv:2305.02301*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Ganesh Jawahar, Haichuan Yang, Yunyang Xiong, Zechun Liu, Dilin Wang, Fei Sun, Meng Li, Aasish Pappu, Barlas Oguz, Muhammad Abdul-Mageed, Laks V. S. Lakshmanan, Raghuraman Krishnamoorthi, and Vikas Chandra. 2023. Mixture-of-supernets: Improving weight-sharing supernet training with architecture-routed mixture-of-experts. *arXiv preprint arXiv:2306.04845*.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online. Association for Computational Linguistics.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and

Harm de Vries. 2022. The stack: 3 tb of permissively licensed source code. *arXiv preprint arXiv:2211.15533*.

Achintya Kundu, Laura Wynter, Rhui Dih Lee, and Luis Angel D. Bathen. 2023. Transfer-once-for-all: AI model optimization for edge. In *IEEE International Conference on Edge Computing and Communications, EDGE 2023, Chicago, IL, USA, July 2-8, 2023*, pages 26–35. IEEE.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Wei Lou, Lei Xun, Amin Sabet, Jia Bi, Jonathon Hare, and Geoff V Merrett. 2021. Dynamic-ofa: Runtime dnn architecture switching for performance scaling on heterogeneous embedded platforms. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3110–3118.

Xindian Ma, Peng Zhang, Shuai Zhang, Nan Duan, Yuexian Hou, Ming Zhou, and Dawei Song. 2019. A tensorized transformer for language modeling. *Advances in neural information processing systems*, 32.

JS McCarley, Rishav Chakravarti, and Avirup Sil. 2019. Structured pruning of a bert-based question answering model. *arXiv preprint arXiv:1910.06360*.

Subhabrata Mukherjee and Ahmed Awadallah. 2020. Xtremedistil: Multi-stage distillation for massive multilingual models. *arXiv preprint arXiv:2004.05686*.

Subhabrata Mukherjee, Ahmed Hassan Awadallah, and Jianfeng Gao. 2021. Xtremedistiltransformers: Task transfer for task-agnostic distillation. *arXiv preprint arXiv:2106.04563*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33-01, pages 4780–4789.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code.

Mohammad Samragh, Mehrdad Farajtabar, Sachin Mehta, Raviteja Vemulapalli, Fartash Faghri, Devang Naik, Oncel Tuzel, and Mohammad Rastegari. 2023. Weight subcloning: direct initialization of transformers using larger pretrained ones.

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

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34-05, pages 8815–8821.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355*.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*.

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

Dilin Wang, Meng Li, Chengyue Gong, and Vikas Chandra. 2021. Attentivenas: Improving neural architecture search via attentive sampling. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6418–6427.

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. 2020. Hat: Hardware-aware transformers for efficient natural language processing. *arXiv preprint arXiv:2005.14187*.

Rui Wang, Qibing Bai, Junyi Ao, Long Zhou, Zhixiang Xiong, Zhihua Wei, Yu Zhang, Tom Ko, and Haizhou Li. 2022. Lighthubert: Lightweight and configurable speech representation learning with once-for-all hidden-unit bert. *arXiv preprint arXiv:2203.15610*.

Jin Xu, Xu Tan, Renqian Luo, Kaitao Song, Jian Li, Tao Qin, and Tie-Yan Liu. 2021. Nas-bert: task-agnostic and adaptive-size bert compression with neural architecture search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1933–1943.

Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. 2020. Bignas: Scaling up neural architecture search

with big single-stage models. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII 16*, pages 702–717. Springer.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pages 36–39. IEEE.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*.

Shaokun Zhang, Xiawu Zheng, Chenyi Yang, Yuchao Li, Yan Wang, Fei Chao, Mengdi Wang, Shen Li, Jun Yang, and Rongrong Ji. 2021. You only compress once: Towards effective and elastic bert compression via exploit-explore stochastic nature gradient. *arXiv preprint arXiv:2106.02435*.

Hui Zhu, Zhulin An, Chuanguang Yang, Kaiqiang Xu, Erhu Zhao, and Yongjun Xu. 2019. Eena: efficient evolution of neural architecture. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. 2023. A survey on model compression for large language models. *arXiv preprint arXiv:2308.07633*.

Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

Appendix

A Details of Experimental Set-up

Following (Hu et al., 2022), we use the fine-tuned MNLI checkpoint to initialize the model weights for experiments on small data sets such as RTE and MRPC. In MLFS, the Low rank matrices are added on the QKV vectors and the intermediate size of feed-forward network (FFN) layers. We set $\beta_l = 0.1\,\forall l$ in feature distillation loss and choose distillation factor $\alpha = 0.9$. For training, we use a maximum sequence length of 128; effective batch size of 128 for QQP, MNLI, QNLI, and 64 for the other data sets. Training is done for a maximum of 8 epochs for all GLUE data sets except SST-2 for which we allocate maximum 3 epochs. We set an initial learning rate of $5e^{-4}$ for QNLI & MNLI, and $1e^{-3}$ for other GLUE data sets. We use rank $r = 8$ for the low rank matrices A, B unless mentioned otherwise. We choose gradient scaling hyper-parameter $\gamma = 1$ for SST-2 and $\gamma = 2$ for all other data sets.

B Additional Experimental Results

First, we present additional results on distilling Santacoder-1.1B model. In Fig. 6, we compare HumanEval performance of a 0.7B Santacoder model fine-tuned through full fine-tuning (FT) from random initialisation vs. full-rank (non-LoRA) MLFS with ($\alpha = 0.9$) and without ($\alpha = 0$) distillation. The improvement in the evaluation numbers is remarkable even after fine-tuning on up to only 5% of the examples. In Fig. 7, we also show better convergence of validation loss on the Santacoder 0.7B for MLFS with distillation loss ($\alpha > 0$). This demonstrates the benefit of MLFS distillation as compared to full MLFS fine tuning of the sliced model.

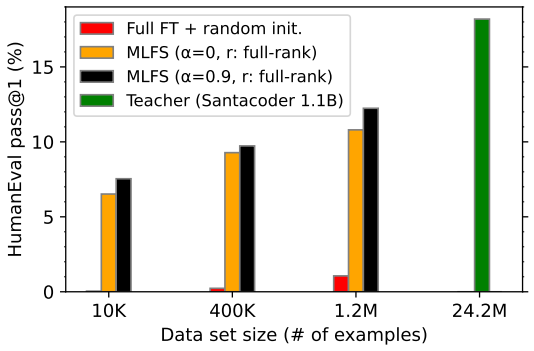

Figure 6: Superior performance of supernet training compared to other full fine-tuning based approaches on three data sets with 10K/400K/1.2M examples.

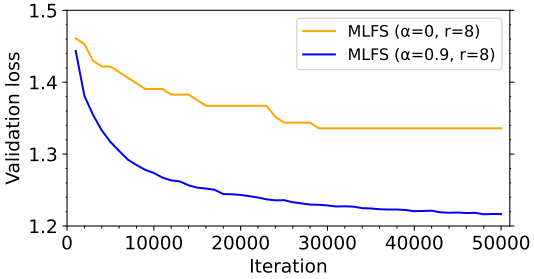

Figure 7: Convergence comparison of validation loss while fine-tuning a custom model using MLFS with/without distillation.

Finally, in Fig. 8, we show performance of a spectrum of models distilled from BERT$_{base}$ using MLFS on 3 more GLUE data sets: QNLI, QQP, and MNLI.

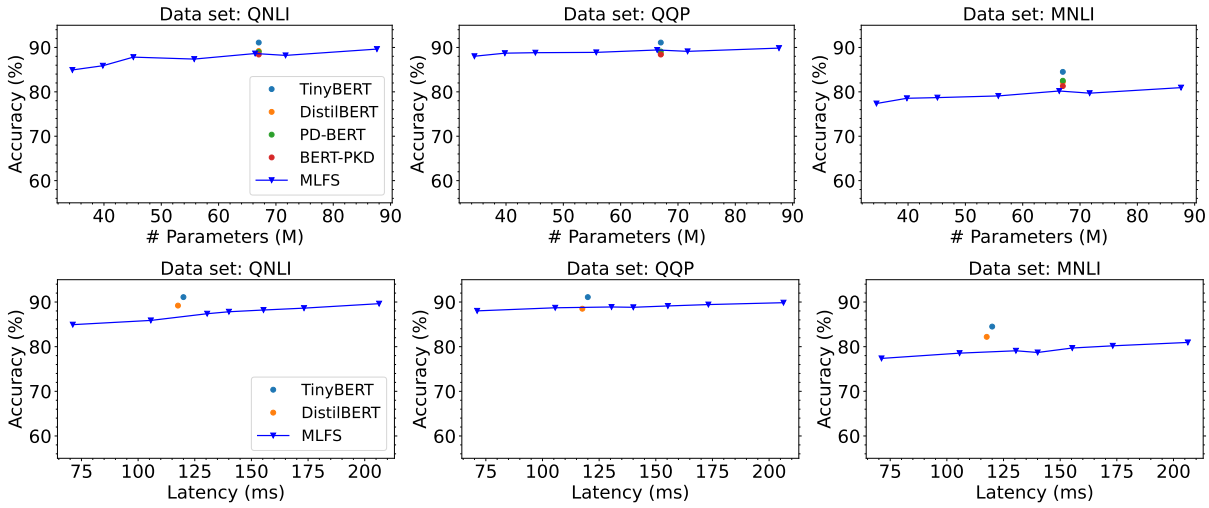

Figure 8: Performance of task-specific BERT models produced by MLFS vs. other methods on 3 GLUE data sets.