Quantized Side Tuning: Fast and Memory-Efficient Tuning of Quantized Large Language Models

Anonymous ACL submission

Abstract

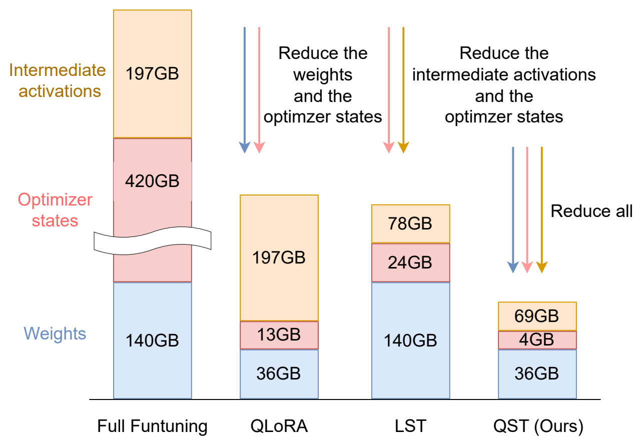
Finetuning large language models (LLMs) has been empirically effective on a variety of downstream tasks. Existing approaches to finetuning an LLM either focus on parameter-efficient finetuning, which only updates a small number of trainable parameters, or attempt to reduce the memory footprint during the training phase of the finetuning. Typically, the memory footprint during finetuning stems from three contributors: model weights, optimizer states, and intermediate activations. However, existing works still require considerable memory and none can simultaneously mitigate memory footprint for all three sources. In this paper, we present Quantized Side Tuing (QST), which enables memory-efficient and fast finetuning of LLMs by operating through a dual-stage process. First, QST quantizes an LLM's model weights into 4-bit to reduce the memory footprint of the LLM's original weights; QST also introduces a side network separated from the LLM, which utilizes the hidden states of the LLM to make task-specific predictions. Using a separate side network avoids performing backpropagation through the LLM, thus reducing the memory requirement of the intermediate activations. Furthermore, QST leverages several low-rank adaptors and gradient-free downsample modules to significantly reduce the trainable parameters, so as to save the memory footprint of the optimizer states. Experiments show that QST can reduce the total memory footprint by up to 2.3 $\times$ and speed up the finetuning process by up to 3 $\times$ while achieving competent performance compared with the state-of-the-art. When it comes to full finetuning, QST can reduce the total memory footprint up to 7 $\times$.

1 Introduction

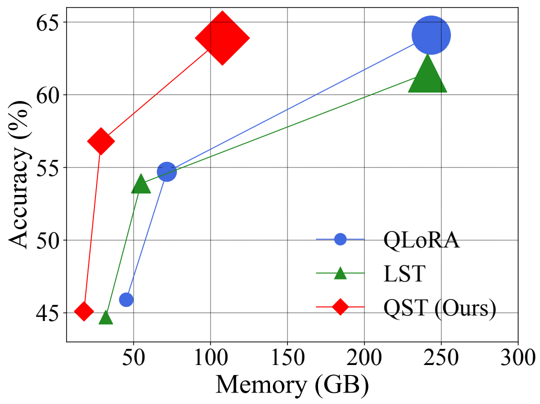
Recent advancements in large language models (LLMs), including GPT (Brown et al., 2020; Floridi and Chiriatti, 2020; OpenAI, 2023), PaLM (Chowdhery et al., 2022), OPT (Zhang et al., 2022), and LLaMA (Touvron et al., 2023), have showcased remarkable task-generalization capabilities across diverse applications (Stiennon et al., 2020; Dosovitskiy et al., 2020). The ongoing evolution of LLMs' capabilities is accompanied by exponential increases in LLMs' sizes, with some models encompassing 100 billion parameters (Raffel et al., 2020; Scao et al., 2022). Finetuning pre-trained LLMs (Min et al., 2021; Wang et al., 2022b,a; Liu et al., 2022) for customized downstream tasks provides an effective approach to introducing desired behaviors, mitigating undesired ones, and thus boosting the LLMs' performance (Ouyang et al., 2022; Askell et al., 2021; Bai et al., 2022). Nevertheless, the process of LLM finetuning is characterized by its substantial memory demands. For instance, finetuning a 16-bit LLaMA model with 65 billion parameters requires more than 780GB of memory (Dettmers et al., 2023).

To reduce the computational requirement of LLM finetuning, recent work introduces *parameter-efficient finetuning* (PEFT), which updates a subset of trainable parameters from an LLM or introduces a small number of new parameters into the LLM while keeping the vast majority of the original LLM parameters frozen (Houlsby et al., 2019; Li and Liang, 2021; Pfeiffer et al., 2020; Hu et al., 2021; He et al., 2021; Lester et al., 2021). PEFT methods achieve comparable performance as full finetuning while enabling fast adaption to new tasks without suffering from catastrophic forgetting (Pfeiffer et al., 2020). However, PEFT methods necessitate caching intermediate activations during forward processing, since these activations are needed to update trainable parameters during backward propagation. As a result, PEFT methods require saving more than 70% of activations and almost the same training time compared to full finetuning (Liao et al., 2023; Sung et al., 2022). Concisely, existing PEFT techniques cannot effectively reduce the memory footprint of LLM finetuning, restricting their applications in numerous real-world memory-constrained scenarios.

Recent work has also introduced approaches to combining PEFT and quantization. For example, QLoRA (Dettmers et al., 2023) quantizes an LLM's weights to 4-bit and leverages low-rank adaption (LoRA) (He et al., 2021) to finetune the quantized LLM. QLoRA reduces the memory footprint of an LLM's weights and optimizer states, and as a result, finetuning a 65B LLM requires less than 48 GB of memory. However, QLoRA does not consider the memory footprint of intermediate



(a) Comparing the memory requirement of different methods.



(b) Comparing the predictive performance of different finetuning techniques.

Figure 1: Figure 1a shows the memory footprint of different methods of fintuning LLaMA-2-70b. Figure 1b shows the MMLU 5-shot accuracy of different methods when tuning LLaMA-2-7B, LLaMA-2-13B, and LLaMA-2-70B. Note that we set the batch size to 16 and the sequence length to 384. Larger markers represent larger models.

activations, which can be particularly large when using a large batch size for finetuning. As a result, QLoRA only supports small-batch training (e.g. a batch size of 1), and finetuning a 65B LLM requires checkpointing gradients (Chen et al., 2016) to fit the LLM on a single 48GB GPU, resulting in long training time. Besides, our evaluation also reveals that the performance of QLoRA becomes unstable when using 16-bit floating points. Sung et al. (2022) and Zhang et al. (2020) propose to use a *side network* to reduce the memory footprint of intermediate activations by *avoiding backpropagation of the LLM* on natural language processing (NLP) and computer vision (CV) tasks, respectively. Even with the adoption of a side network, the inherent model size of the LLM remains a challenge. Meanwhile, these approaches focus on small models (i.e., less than 3 billion parameters), and their applicability and efficacy for larger models remain unexplored.

In this paper, we propose a fast, memory-efficient LLM finetuning framework, called **Q**uantized **S**ide-**T**uning (QST), which operates through a dual-stage process as shown in Figure 2. First, QST quantizes an LLM into 4-bit to reduce the memory footprint of its model weights. Second, QST introduces a side network separating from the quantized LLM to avoid performing backward propagation for the quantized LLM, thus saving the memory footprint of intermediate activations. During the training phase of QST, the input to each layer of the side network is formed by combining (1) the downsampled output of the corresponding quantized LLM layer and (2) the output of the previous layer of the side network. A larger LLM usually has a larger model depth (i.e., the number of layers) and width (the hidden size of each layer), which in turn requires more trainable parameters for the downsampling layers. Unlike Sung et al. (2022) that leverages linear layer to perform downsampling, QST uses several low-rank adapter methods (He et al., 2021; Edalati et al., 2022) such as MaxPooling (LeCun et al., 1998) and AvgPooling, significantly reducing the required trainable parameters and the memory footprint for the optimizer states. After that, we use a learnable parameter to assign weights and subsequently aggregate the hidden states of the quantized LLM and the side network. Finally, we reuse the LLM head or classifier to predict. Combined with 4-bit quantization and side tuning, QST significantly reduces all three main contributors of the memory footprint and training time during the training phase. Besides, QST does not increase inference latency since the LLM and side network can be computed in paralle. Figure 1 compares the memory footprint of QST and existing parameter-efficient fine-tuning methods, including QLoRA and LST.

To validate the effectiveness of our QST, we conduct extensive evaluations for different types of LLMs (e.g., OPT, LLaMA 2), with 1.3B to 70B parameters, on various benchmarks. Experiment results show that QST can reduce the total memory footprint by up to 2.3 $\times$ and speed up the finetuning process by up to 3 $\times$ while achieving competent performance compared with the state-of-the-art. Our codes are released to the GitHub anonymously [1].

2 Related Work

2.1 Parameter-Efficient Finetuning

Finetuning allows an LLM to adapt to specialized domains and tasks (Devlin et al., 2018; Radford et al., 2019; Brown et al., 2020). However, fully finetuning an LLM comes with high computation costs due to the rapidly increasing LLM sizes. *Parameter-efficient finetuning* (PEFT) methods are proposed to solve this issue. Drawing inspiration from the pronounced sensitivity of

[1] https://anonymous.4open.science/r/QST-0242

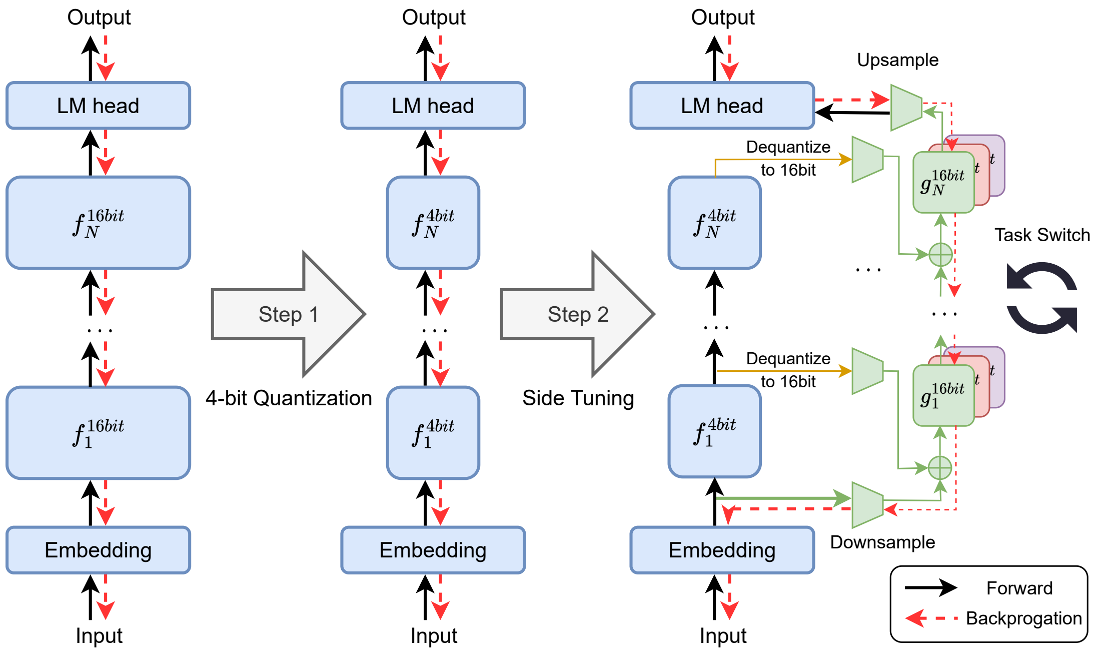


Figure 2: A overview of quantized side tuning.

LLMs to prompts as highlighted in Schick and Schütze (2020), a series of studies introduce trainable prompt embeddings prepended to the input text or attention components while preserving the original LLM parameters (Liu et al., 2023; Li and Liang, 2021; Lester et al., 2021). Rusu et al. (2016) and Houlsby et al. (2019) propose *adapter* modules to introduce new task-specific parameters, which are inserted into the Transformer layers inside the LLM. LoRA (Hu et al., 2021) leverages the low-rank decomposition concept to construct trainable parameters inserted into the original LLM weights. (IA)^3 (Liu et al., 2022) proposes to scale the pre-trained model weights of an LLM with a trainable vector. Of late, there has been a surge in the proposal of unified approaches that amalgamate various PEFT methods by leveraging human heuristics (He et al., 2021) or employing neural architecture search (Zhou et al., 2023; Zoph and Le, 2016; Mao et al., 2021). Existing PEFT approaches focus on optimizing model performance while minimizing trainable parameters. However, a reduction in the number of trainable parameters does not inherently imply a corresponding reduction in memory footprint.

2.2 Memory-Efficient Training and Finetuning

Memory-efficient training and finetuning aims to reduce the memory footprint during the LLM training and/or finetuning phase. Reversible neural networks (Gomez et al., 2017; Kitaev et al., 2020; Mangalam et al., 2022) allow the intermediate activations of each layer to be recomputed from the activation of its next layer, thus exempting the need to save intermediate activations. *Gradient checkpointing* (Chen et al., 2016) offers an optimization strategy that balances computational resources against memory footprint. Specifically, it reduces memory requirement by selectively discarding certain intermediate activations, which are subsequently recomputed through an additional forward pass when needed. Another line to enhancing memory efficiency involves network compression, that is, the original LLM is reduced to a more compact form, thereby making both the training and inference phases more computationally economical. Network pruning and distillation are the most prevalent strategies for network compression. Network distillation (Hinton et al., 2015; Koratana et al., 2019) involves the creation of a student network that is trained to approximate the output distribution of a teacher network across a specified dataset. Network pruning (Frankle and Carbin, 2018; Frankle et al., 2020) aims to streamline models by ascertaining the significance of individual parameters and subsequently eliminating those deemed non-essential. Compared with PEFT methods, network compression yields models optimized for expedited inference, whereas PEFT methods may achieve superior performance by updating a small set of trainable parameters.

Recently, QLoRA (Dettmers et al., 2023) quantizes the LLM to 4-bit and then adds LoRA to finetune the quantized LLM. QLoRA significantly reduces the memory footprint of weights and optimizer states compared with full finetuning while retaining similar performance. QLoRA does not consider the memory footprint of intermediate activations, and thus falls short in finetuning the LLM with a large batch size, resulting in a long training time. In the context of NLP and CV tasks, the studies by (Sung et al., 2022) and (Zhang et al., 2020) introduce the concept of employing a side network. The side network aims to obviate the need for backpropagation through the LLM, thereby reducing the memory footprint associated with intermediate activations. Despite incorporating the side network, the inherent model size (i.e., the memory footprint of weights) of the LLM still poses computational challenges. Hence, both methods can only focus exclusively on models with fewer than 3 billion parameters, and fail to finetune models with more parameters.

3 Quantized Side Tuning

In this section, we first describe the process of quantizing an LLM into 4-bit, and then introduce our design of the side network for side tuning.

3.1 4-bit Quantization

Quantization is the process of converting a data type with more bits (e.g., 32- or 16-bit floating points) into another data type with fewer bits (e.g., 8-bit integers or 4-bit floating points). QST first quantizes an LLM from 16-bit into 4-bit, formulated as follows.

$$X^{4bit} = \text{round}\left(\frac{M_{4bit}}{\text{Absmax}(X^{16bit})} X^{4bit}\right) \quad (1)$$

$$= \text{round}\left(c^{16bit} \cdot X^{16bit}\right), \quad (2)$$

where X^{4bit} and X^{16bit} are tensors in 4- and 16-bit, respectively. M_{4bit} is the maximum value of the 4-bit data type. For example, $M_{NF4} = 1$, where NF4 is an information-theoretically optimal data type that ensures each quantization bin has an equal number of values assigned from the input tensor. QST considers both NF4 and FP4 to quantize an LLM. We empirically demonstrate that NF4 performs the best in our experiments (see Section 4.6). c^{16bit} is the quantization constant (or quantization scale) of the 16-bit data type. Correspondingly, dequantization is given by

$$dequant(c^{16bit}, X^{4bit}) = \frac{X^{4bit}}{c^{16bit}} = X^{16bit}. \quad (3)$$

The key limitation of this method arises when the input tensor contains values with very large magnitudes, commonly referred to as *outliers*. Such outliers can result in under-utilization of the quantization bins, leading to sparsely populated or even empty bins in some instances. To address this issue, a prevalent strategy involves partitioning the input tensor into discrete blocks, each subjected to independent quantization with its own associated quantization constant. As a result, the input tensor $X \in \mathbb{R}^{b \times h}$ is decomposed into n contiguous blocks, each comprising B elements. This decomposition is facilitated by flattening X into a 1-dimensional array, which is then partitioned into $n = \frac{(b \times h)}{B}$ individual blocks. Then, we can leverage E.q. (1) to independently quantize these n blocks using different quantization constants. Typically, minimizing the error associated with 4-bit quantization would necessitate the utilization of smaller block sizes. This is attributed to the reduced influence of outliers on other weights. However, using a small block size leads to high memory overhead since we need to allocate more memory for these quantization

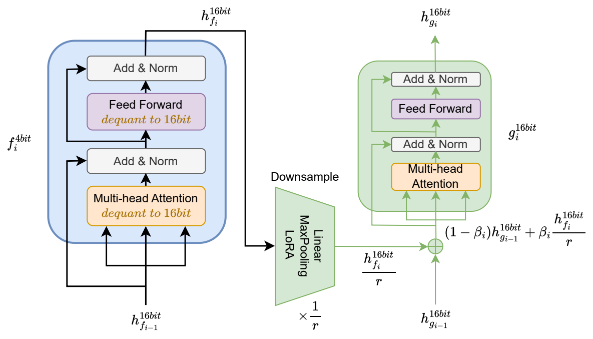


Figure 3: Illustration of i^{th} layer of QST.

constants. To reduce the memory footprint of quantization constants, we can use the same quantization strategy to quantize these quantization constants (Dettmers et al., 2023). In this paper, we use 8-bit float points to quantize the quantization constants, and the forward pass of a single linear layer in the LLM is defined as $Y^{16bit} = dequant(dequant(c_2^{16bit}, c_1^{8bit}), W^{4bit})X^{16bit}$. 4-bit quantization can significantly reduce the memory footprint of weights, facilitating easier storage and deployment of LLMs. Besides, low-precision floating numbers are faster to execute on modern accelerators such as GPUs, leading to faster model training and inference. Nonetheless, the high to low precision data type conversion process during quantization can lead to accuracy degradation, attributable to the inherent information loss.

3.2 Side Tuning

We now analyze the memory footprint of LLM training and then introduce the neural architecture of the side network, which reduces the inherent information loss and minimizes accuracy drop during quantization.

Memory footprint during the training phase. For a given LLM with N layers, let $y_i = f_i(W_i, x_i)$ denotes the i^{th} transformer layer of the LLM, where x_i is the input to the i^{th} layer (i.e., $x_i = y_{i-1}$). The memory required during the training phase of the LLM predominantly comprises three main contributors: **M1**) weights of the LLM $\{W_i\}_{i=1}^N$, **M2**) the optimizer state, which is threefold the size of the trainable parameters when employing the Adam optimizer (Kingma and Ba, 2014) (one for gradient and two for moments), and **M3**) the intermediate activations $\{y'_i\}_{i=1}^N$. The memory footprint of intermediate activations is related to model depth, width, and several training settings, e.g., batch size and sequence length. QLoRA reduces the memory footprint of an LLM's weights and optimizer states (M1 and M2) but fails to reduce intermediate activations (M3). When finetuning an LLM with a large batch size and/or long sequence length, the memory footprint of QLoRA increases significantly. However, using a small batch size results in long training time. Sung et al. (2022) only reduces the memory footprint of intermediate activations (M3), thus it struggles to finetune a model with more than 3 billion parameters.

Side network. Our side network g serves as a lightweight version of the quantized LLM f. The hidden state and weight dimension of g are r times smaller than those of f, where r is the reduction factor. During the forward pass, the hidden state of the i^{th} layer of the side network h_{g_i} is formulated by $h_{g_i}^{16bit} = (1-\beta_i) * downsample_i(h_{f_i}^{16bit}) + \beta_i * h_{g_{i-1}}^{16bit}$, where $h_{f_i}^{16bit}$ is the hidden state of the i^{th} layer of f and can be computed using E.q. (3). The illustration of i^{th} layer of our QST is shown in Figure 3. Note that we use the output of the embedding layer and the downsampled embedding layer as $h_{f_0}^{16bit}$ and $h_{g_0}^{16bit}$. $\beta_i = sigmoid(\alpha_i)$ is a learned gate parameter of i^{th} layer, where α_i is a learnable zero-initialized scalar. $downsample_i$ is the downsample module of the i^{th} layer to reduce the hidden state dimension of f by r times. Prior work leverages linear projections to downsample (i.e., $\times\frac{1}{r}$) the high-dimensional hidden states of f to the low-dimensional hidden states of g. However, an LLM typically comprises plenty of layers with substantially high-dimensional hidden states, particularly when the number of parameters exceeds 3 billion. Using linear projections to downsample involves a significant amount of trainable parameters, requiring a high memory footprint for the parameters and their optimizer states. For example, if the LLM has 24 layers, the dimension of its hidden state is 2048 and the reduction factor r is 4, the downsample module consumes about 50% of the overall trainable parameters.

To address this problem, we leverage several different downsample methods, including LoRA (He et al., 2021), Adapter (Edalati et al., 2022), MaxPooling (LeCun et al., 1998) and AvgPooling. LoRA augments a linear projection through an additional factorized projection, which can be formulated as $W = L_1L_2$, where $W \in \mathbb{R}^{d_{in}\times d_{out}}$, $L_1 \in \mathbb{R}^{d_{in}\times d_r}$ and $L_2 \in \mathbb{R}^{d_r\times d_{out}}$. Adapter is similar to LoRA but introduces an extra nonlinear function between L_1 and L_2. Using LoRA or Adapter can reduce the ratio of the trainable parameters of these downsample modules from 56% to 8%. MaxPooling and AvgPooling do not introduce extra trainable parameters. We empirically demonstrate that the Adapter performs the best in our experiments. Finally, we upsample (i.e., $\times r$) from low-dimensional hidden states of g to high-dimensional hidden states of f, and subsequently leverage the LLM's head or classifier to perform task-specific predictions. When switching across different downstream tasks, QST alters the side network and therefore eliminates the necessity to redeploy the LLM.

QST only updates the parameters of the side network g, but not the 4-bit weights in the LLM f. Unlike QLoRA, the calculation of the gradient $\frac{\partial L}{\partial g}$ does not entail the calculation of $\frac{\partial L}{\partial f}$, thus avoiding the extensive computational costs of performing backpropagation on f, which ultimately reduces the memory footprint of intermediate activations and speeds up finetuning.

In summary, QST leverages a 4-bit data type to store an LLM's model weights, thus reducing the memory footprint of weights (M1). In addition, QST leverages a 16-bit computation data type for the forward pass and backpropagation computation and only computes the gradient of weights in g (M3). Finally, QST leverages several factorized projection and gradient-free downsample methods to reduce the trainable parameters (M2). These techniques together allow QST to reduce the memory requirement for all three factors, resulting in fast and memory-efficient finetuning with a nearly 1% performance drop.

4 Evaluation

In this section, we empirically validate the effectiveness of our QST method by examining its performance for LLMs with different types (e.g., OPT and LLaMA 2), sizes (from 1.3B to 70B), and benchmarks.

4.1 Experimental Setup

Datasets. We evaluate the performance of QST and several baselines on natural language understanding (NLU) and natural language generation tasks. For NLU experiments, we use the GLUE (Wang et al., 2018) (General Language Understanding Evaluation) and MMLU (Hendrycks et al., 2020) (Massively Multitask Language Understanding) benchmarks. The GLUE benchmark provides a comprehensive evaluation of models across a range of linguistic tasks. These tasks encompass linguistic acceptability as examined in CoLA (Warstadt et al., 2019), sentiment analysis as portrayed in SST2 (Socher et al., 2013), tasks probing similarity and paraphrase distinctions such as MRPC (Dolan and Brockett, 2005), QQP (Iyer, 2017), and STS-B (Cer et al., 2017), in addition to natural language inference tasks including MNLI (Williams et al., 2017), QNLI (Rajpurkar et al., 2016), and RTE (Bentivogli et al., 2009). We report accuracy on MNLI, QQP, QNLI, SST-2, MRPC, and RTE, Pearson correlation coefficients on SST-B, and Mathews correlation coefficients (Matthews, 1975) on CoLA. The MMLU benchmark consists of 57 tasks including elementary mathematics, US history, computer science, law, and more. We report the average 5-shot test accuracy on the 57 tasks.

Models. We use decoder-only LLMs such as the OPT series (OPT-1.3B, OPT-2.7B, OPT-6.7B, OPT-13B, OPT-30B, and OPT-66B) and the LLaMA-2 series (LLaMA-2-7B, LLaMA-2-13B, and LLaMA-2-70B).

Baselines. We compare QST with QLoRA (Dettmers et al., 2023), LST (Sung et al., 2022), LoRA (He et al., 2021), and Adapter (Houlsby et al., 2019). Note that we only compare LST, LoRA, and Adapter when the model size is less than 3B since their memory footprint of weights can be excessively huge beyond that.

Implementation. We set the reduction factor r to 16 by default. We use Adapter as the downsample module, a linear layer as the upsample module, and set the rank of the Adapter to 16. We use the NF4 data type to store the weights of the LLM and bfloat16 as the data type for

Method	# Param. (%)	Memory (GB)	RTE	MRPC	STS-B	CoLA	SST-2	QNLI	QQP	MNLI	Avg.
					OPT-1.3B (batchsize=16, sequence length=512)						
QLoRA	4.41%	31.3	$81.3_{\pm 1.6}$	$\underline{83.3}_{\pm 1.1}$	$\mathbf{89.9}_{\pm 0.5}$	$\underline{62.1}_{\pm 2.3}$	$\mathbf{94.9}_{\pm 0.1}$	$\mathbf{86.3}_{\pm 0.2}$	$\mathbf{87.1}_{\pm 0.1}$	$76.0_{\pm 0.3}$	**82.6**
LST	2.39%	$\underline{20.9}$	$82.0_{\pm 2.2}$	$83.1_{\pm 1.3}$	$88.6_{\pm 0.4}$	$59.5_{\pm 3.1}$	$\underline{94.4}_{\pm 0.3}$	$\underline{86.1}_{\pm 0.3}$	$86.4_{\pm 0.6}$	$77.8_{\pm 0.5}$	82.2
LoRA	2.36%	32.9	$\mathbf{82.7}_{\pm 1.9}$	$\mathbf{83.4}_{\pm 0.9}$	$89.3_{\pm 0.2}$	$\mathbf{62.5}_{\pm 1.7}$	$93.7_{\pm 0.7}$	$81.4_{\pm 9.3}$	$\underline{86.9}_{\pm 0.3}$	$\mathbf{81.2}_{\pm 0.1}$	**82.6**
Adapter	$\underline{0.48\%}$	32.5	$\underline{82.2}_{\pm 0.8}$	$82.7_{\pm 1.4}$	$\underline{89.7}_{\pm 1.6}$	$60.6_{\pm 3.0}$	$93.8_{\pm 0.2}$	$83.6_{\pm 0.1}$	$86.3_{\pm 0.4}$	$\underline{80.5}_{\pm 0.1}$	$\underline{82.4}$
QST	**0.45%**	**17.7**	$79.5_{\pm 2.5}$	$81.7_{\pm 1.1}$	$88.4_{\pm 1.1}$	$59.7_{\pm 2.9}$	$94.3_{\pm 0.3}$	$85.7_{\pm 0.5}$	$84.3_{\pm 0.7}$	$77.1_{\pm 0.6}$	81.3
					OPT-2.7B (batchsize=16, sequence length=512)						
QLoRA	3.57%	47.0	$83.6_{\pm 1.5}$	$\mathbf{84.8}_{\pm 1.2}$	$\underline{91.2}_{\pm 0.6}$	$\underline{63.7}_{\pm 2.6}$	$\mathbf{95.6}_{\pm 0.2}$	$\mathbf{88.7}_{\pm 0.1}$	$89.5_{\pm 0.2}$	$78.3_{\pm 0.4}$	$\underline{84.4}$
LST	2.39%	$\underline{30.7}$	$82.5_{\pm 2.9}$	$83.9_{\pm 1.5}$	$89.1_{\pm 0.9}$	$60.7_{\pm 3.5}$	$95.3_{\pm 0.4}$	$\underline{87.3}_{\pm 0.2}$	$88.8_{\pm 1.0}$	$80.4_{\pm 0.7}$	83.5
LoRA	1.90%	50.4	$\mathbf{84.7}_{\pm 1.4}$	$\underline{84.6}_{\pm 0.8}$	$90.9_{\pm 0.1}$	$\mathbf{64.5}_{\pm 2.4}$	$95.3_{\pm 0.6}$	$83.0_{\pm 7.4}$	$\mathbf{90.7}_{\pm 0.1}$	$\mathbf{82.6}_{\pm 0.2}$	**84.5**
Adapter	**0.37%**	49.9	$\underline{84.4}_{\pm 0.7}$	$83.7_{\pm 1.4}$	$\mathbf{91.5}_{\pm 1.9}$	$63.4_{\pm 3.8}$	$\underline{95.4}_{\pm 0.3}$	$83.6_{\pm 0.2}$	$\underline{90.2}_{\pm 0.3}$	$\underline{81.1}_{\pm 0.1}$	84.2
QST	$\underline{0.43\%}$	**24.4**	$80.1_{\pm 2.1}$	$83.7_{\pm 1.2}$	$88.9_{\pm 1.4}$	$62.0_{\pm 3.4}$	$95.2_{\pm 0.8}$	$86.6\ _{\pm 0.9}$	$86.5_{\pm 0.9}$	$80.4_{\pm 0.6}$	83.0
					OPT-6.7B (batchsize=16, sequence length=512)						
QLoRA	2.33%	63.6	$84.5_{\pm 1.2}$	$85.9_{\pm 0.7}$	$92.0_{\pm 0.8}$	$64.3_{\pm 2.8}$	$96.2_{\pm 0.1}$	$90.2_{\pm 0.2}$	$90.7_{\pm 0.2}$	$79.8_{\pm 0.3}$	85.5
QST	0.42%	27.5	$80.8_{\pm 1.4}$	$85.2_{\pm 1.0}$	$89.6_{\pm 0.7}$	$62.8_{\pm 2.6}$	$96.4_{\pm 0.6}$	$87.3_{\pm 1.1}$	$89.4_{\pm 0.8}$	$81.6_{\pm 0.5}$	84.2

Table 1: Experiments results on GLUE benchmark (using BF16 data type).

computation. We adopt the same parameters reported in QLoRA, LST, LoRA, and Adapter to construct the baselines. Other hyperparameters are specified in Appendix A and Appendix B. We run each experiment three times under different random seeds and report the average performance. We conduct all the experiments using Pytorch (Paszke et al., 2017) and HuggingFace library (Wolf et al., 2019) on 4 NVIDIA RTX A5000 GPUs, each with 24GB memory.

4.2 Experiments on GLUE Benchmark

Table 1 shows the performance of different methods on the GLUE benchmark. Overall, QST achieves the lowest memory footprint among all methods while attaining competent accuracy. Particularly, for relatively small models (i.e., OPT-1.3B and OPT-2.7B), QST reduces the memory footprint by around 2× compared with QLoRA, LoRA, and Adapter, while achieving comparable accuracy. Compared with LST, QST reduces the memory requirement by 3.2GB and 6.3GB for finetuning OPT-1.3B and OPT-2.7B. QST also reduces the trainable parameters by around 10× and 5× compared with QLoRA and the other baselines, respectively.

For larger models such as OPT-6.7B, we focus on comparing QST with QLoRA. This is because QLoRA has similar accuracy with the other baselines, but LoRA, Adapter, and LST all have excessively huge memory footprints of weights when it comes to finetuning OPT-6.7B[2]. Compared with QLoRA, QST reduces the memory footprint and trainable parameters by 2.3× and 5.5×, while only introducing a 1.3% accuracy drop.

4.3 Experiments on MMLU Benchmark

The experiment results of the MMLU benchmark are shown in Table 2. We set the batch size to 4 and the sequence length to 384. We use the Alpaca dataset (Taori et al., 2023) to finetune both QLoRA and QST. We compare QST with QLoRA on accuracy and memory requirement over OPT-1.3B, OPT-2.7B, OPT-6.7B, OPT-13B, OPT-30B, OPT-66B, LLaMA-2-7B, LLaMA-2-13B, and LLaMA-2-70B. QST improves the accuracy by 0.1% on average while reducing the memory footprint by 1.8× compared with QLoRA. Particularly, QST yields an enhancement of 2.1% in accuracy over QLoRa when finetuning LLaMA-2-13B. When finetuning the OPT-2.7B, OPT-6.7B, and OPT-13B models, QST achieves 0.3%, 0.6%, and 0.3% accuracy improvements, respectively.

[2]QLoRA can leverage gradient accumulation to finetune with a batch size of 16 while guaranteeing an affordable memory footprint.

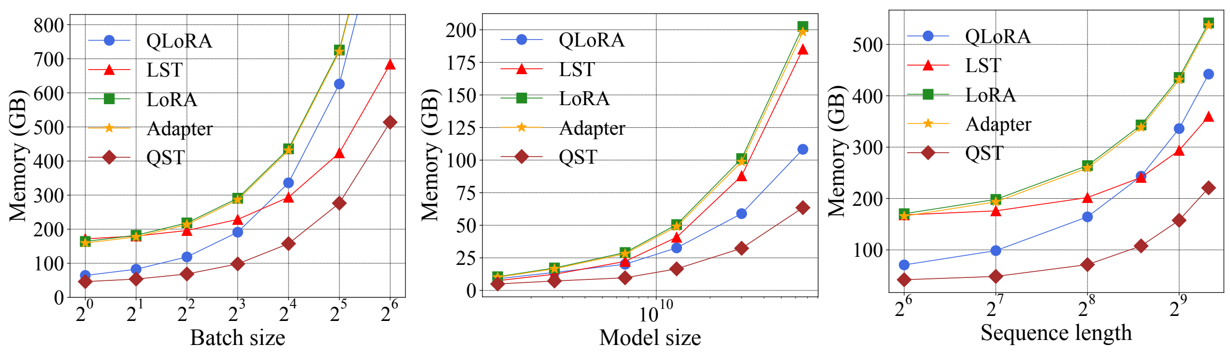
4.4 Memory Footprint Analysis

Effects of batch size. Figure 5(a) illustrates the effects of batch size for different methods. We use LLaMA-2-70B as the LLM and set the sequence length to 512. While the memory footprint of all methods increases as the batch size increases, QST achieves the lowest memory footprint among all, regardless of the batch size. Particularly, the memory footprint of QST is only one-third of LoRA and Adapter. Besides, the memory footprint of both QST and LST grows less drastically than QLoRa, Adapter, and LoRa as the batch size increases. This is because both LST and QST use side tuning to reduce the hidden dimension of the intermediate activations, thereby alleviating the growth of memory footprint induced by intermediate activations. QST also achieves an additional reduction of approximately 100GB in memory footprint compared to LST, thanks to the 4-bit quantization design that effectively compresses the memory footprint of the weights and well design of the downsample modules to reduce the optimizer states.

Effects of the model size. Figure 5(b) shows the effects of the total model bits on different methods. We use the OPT model series and set the batch size to 4. Due to the 4-bit quantization, QST and QLoRA reduce the memory footprint compared with the other baselines. The memory footprint gap further widens as the model size increases. Besides, QST achieves around 2 times reduction in memory footprint compared with QLoRA

Method	OPT-1.3B	OPT-2.7B	OPT-6.7B	OPT-13B	OPT-30B	OPT-66B	LLaMA-2-7B	LLaMA-2-13B	LLaMA-2-70B	Avg.
QLoRA	25.0/6.3	25.2/10.1	25.6/15.5	26.5/25.4	27.7/46.8	36.4/87.5	45.9/15.6	54.7/25.4	64.1/95.5	36.8/36.5
QST	24.3/3.2	25.5/4.8	26.2/7.2	26.8/12.6	27.3/25.7	36.0/52.3	45.1/7.3	56.8/12.6	63.9/56.0	36.9/20.2

Table 2: Experiment results (accuracy/memory) on MMLU 5-shot.



(a) Effect of batch size. (b) Effect of total model bits. (c) Effect of sequence length.

Figure 4: Effects of the batch size, total model bits, and sequence length on memory footprint.

Method	FLOPS per token (10^{-5})		
	LLaMA-2-7B	LLaMA-2-13B	LLaMA-2-70B
QLoRA	11.7	16.0	38.1
LST	11.0	19.0	80.7
LoRA	11.3	15.6	37.2
Adapter	11.2	15.6	27.2
QST	4.4	6.1	15.3

Table 3: Experiments on FLOPS per token of different methods.

Method	LLaMA-2-7B	LLaMA-2-13B	LLaMA-2-70B	Avg.
FP4	44.5	55.4	63.5	54.5
NF4	45.1	56.8	63.9	55.3

Table 4: Experiments on 4-bit data types.

thanks to its small volume of trainable parameters and intermediate activations.

Effects of sequence length. Figure 5(c) shows the effects of sequence length on different methods. We use LLaMA-2-70B and set the batch size to 4. Similar to the effect of batch size, LST and QST alleviate the growth rate of memory footprint of intermediate activations, while QST further achieves around 100GB reduction in memory footprint compared with LST.

4.5 Experiments on Training Throughput

Table 3 shows the training throughput of different methods, measured by FLOPS per token (the lower the better), on LLaMA-2-7B, LLaMA-2-13B, and LLaMA-2-70B. While the FLOPS per token of all methods increases as the model size grows, QST achieves the lowest FLOPS per token among all. Particularly, QST achieves around 2.5$\times$ speed up compared with the baselines. LST suffers from the highest FLOPS per token. The FLOPS per token of QLoRA is slightly higher than LoRA and Adapter since QLoRA adds more LoRA components.

4.6 Sensitive Analysis

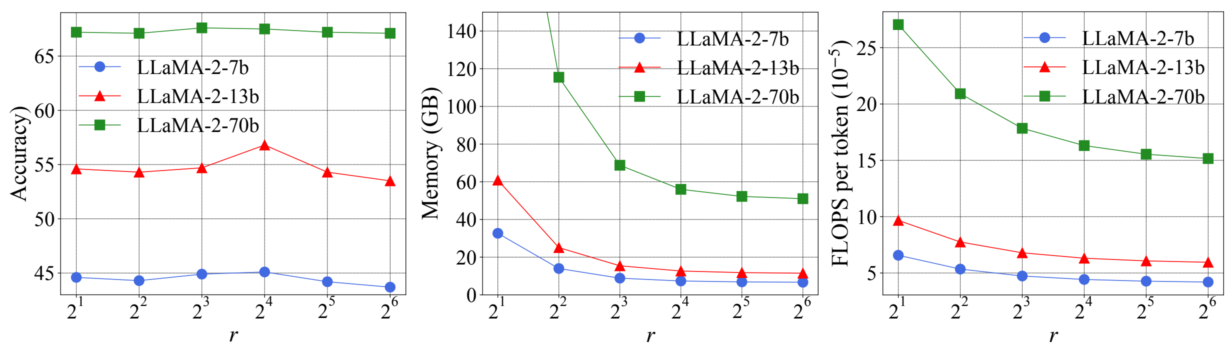
Effects of reduction factor r. We conduct experiments using LLaMA-2-7B, LLaMA-2-13B, and LLaMA-2-70B to verify the effects of reduction factor r (from 2 to 64) on memory footprint, MMLU accuracy, and throughput. We set the batch size to 4 and the sequence length to 384. The MMLU accuracy changes slightly as r varies as shown in Figure 5a. QST achieves the best accuracy of finetuning LLaMA-2-7B and LLaMA-2-13B when r is set to 16. As shown in Figure 5b and 5c, the memory footprint and the FLOPS per token decrease drastically when r varies from 2 to 16 for finetuning all the models. The memory footprint and the FLOPS per token decrease slightly when r varies from 16 to 64. Therefore, we use r to 16 in our experiments as default.

Effects of 4-bit data types. We evaluate two 4-bit data types: FP4 and NF4 using the LLaMA-2 model series and the MMLU benchmark. As shown in Table 4, NF4 improves the average accuracy by about 0.8% compared with FP4. Therefore, we use NF4 as the default 4-bit data type in our experiments.

Effects of computation data types. We analyze the effects of two computation data types: BF16 (results shown in Table 1) and FP16 (results shown in Table 5). As can be seen, QST retains similar results using FP16 and BF16. On the other hand, QLoRA is unstable using FP16 as the computation data type. We finetune OPT-6.7B on the GLUE benchmark and discover that QLoRA fails to finetune on the MRPC and QNLI datasets. We run each

Method	MRPC	QNLI
QLoRA	68.0	60.3
QST	85.6	87.2

Table 5: Experiments of QLoRA and QST using FP16.



(a) Effect of r on MMLU accuracy. (b) Effect of r on memory footprint. (c) Effect of r on training throughput.

Figure 5: Effects of the reduction factor r on MMLU accuracy, memory footprint, and training throughput.

Method	# Param. (%)	Ratio	Memory	Accuracy
Linear	0.85%	56.0%	7.8	44.9
LoRA	0.41%	7.8%	7.3	44.7
Adapter	0.41%	7.8%	7.3	45.1
MaxPooling	0.38%	0%	7.3	43.7
AvgPooling	0.38%	0%	7.3	42.5

Table 6: Experiments on downsample modules. Note that the ratio represents the ratio of downsample modules trainable parameter in all trainable parameters.

Method	Training Time	Memory	Score
QLoRA-70B	~80h	96.3	6.61
QST-70B	~25h	56.1	6.60

Table 7: Chatbot performance on QLoRA and QST.

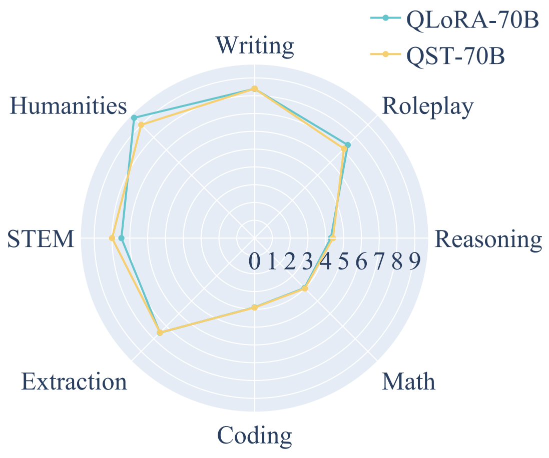


Figure 6: MT-Bench scores of QLoRA and QST in different categories.

dataset under three different random seeds and QLoRA fails on two of them.

Effects of downsample modules. We conduct experiments on different downsample modules: Linear, LoRA, Adapter, MaxPooling, and AvgPooling using LLaMA-2-7B and the MMLU benchmark. As shown in Table 6, using Adapter as the downsample module achieves the best performance among all baselines, and reduces the trainable parameters and memory footprint.

4.7 Experiments on Chatbot Performance

We conduct experiments on Chatbot performance using MT-benchmark (Zheng et al., 2023). MT-benchmark is a set of challenging multi-turn open-ended questions for evaluating the chat assistant's performance in writing, roleplay, reasoning, math, coding, extraction, STEM, and humanities categories. In our experiments, we use GPT-4 to act as judges and assess the quality of the responses of the model finetuned by QLoRA and QST. We finetune LLaMA-2-70B using a variant of OASST1 (Dettmers et al., 2023). Table 6 shows the experiment results of QLoRA and QST on the total training time, memory footprint, and the average MT-Bench score over 8 categories. QST speeds up the training by 3.2 $\times$ and reduces memory footprint by 1.7 $\times$, with just a slight score drop of 0.16 compared withe QLoRA. As shown in Figure 6, QST performs better than QLoRA in STEM, Extraction, Coding, and reasoning. This may contribute to the transformer block of the side network, which can reconstruct the inherent information loss of context. QST performs slightly worse than QLoRA in other categories.

5 Conclusion

In this paper, we propose Quantized Side Tuing (QST), a novel fast and memory-efficient finetuning framework. QST operates through a dual-stage process: first, QST quantizes the LLM into 4-bit to reduce the memory footprint of the weights in LLM; then QST introduces a side network separated from the LLM, which utilizes the hidden states of the LLM to make task-specific predictions. QST can significantly reduce the memory footprint of LLM finetuning compared to existing approaches. In particular, experiments show that QST can reduce the total memory footprint by up to 2.3 $\times$ and speed up the finetuning process by up to 3 $\times$ while achieving comparable performance compared with the state-of-the-art.

References

Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, et al. 2021. A general language assistant as a laboratory for alignment. *arXiv preprint arXiv:2112.00861*.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. 2022. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*.

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth pascal recognizing textual entailment challenge. *TAC*, 7:8.

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.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Bill Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Third International Workshop on Paraphrasing (IWP2005)*.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Ali Edalati, Marzieh Tahaei, Ivan Kobyzev, Vahid Partovi Nia, James J Clark, and Mehdi Rezagholizadeh. 2022. Krona: Parameter efficient tuning with kronecker adapter. *arXiv preprint arXiv:2212.10650*.

Luciano Floridi and Massimo Chiriatti. 2020. Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30:681–694.

Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. 2020. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, pages 3259–3269. PMLR.

Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. 2017. The reversible residual network: Backpropagation without storing activations. *Advances in neural information processing systems*, 30.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.

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

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Shankar Iyer. 2017. First quora dataset release: Question pairs. `https://quoradata.quora.com/First-Quora-Dataset-Release-\Question-Pairs`.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*.

Animesh Koratana, Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. Lit: Learned intermediate representation training for model compression. In *International Conference on Machine Learning*, pages 3509–3518. PMLR.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Baohao Liao, Shaomu Tan, and Christof Monz. 2023. Make your pre-trained model reversible: From parameter to memory efficient fine-tuning. *arXiv preprint arXiv:2306.00477*.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2023. Gpt understands, too. *AI Open*.

Karttikeya Mangalam, Haoqi Fan, Yanghao Li, Chao-Yuan Wu, Bo Xiong, Christoph Feichtenhofer, and Jitendra Malik. 2022. Reversible vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10830–10840.

Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Wen-tau Yih, and Madian Khabsa. 2021. Unipelt: A unified framework for parameter-efficient language model tuning. *arXiv preprint arXiv:2110.07577*.

Brian W Matthews. 1975. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451.

Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2021. Metaicl: Learning to learn in context. *arXiv preprint arXiv:2110.15943*.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

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.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2022. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*.

Timo Schick and Hinrich Schütze. 2020. Exploiting cloze questions for few shot text classification and natural language inference. *arXiv preprint arXiv:2001.07676*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021.

Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. 2022. Lst: Ladder side-tuning for parameter and memory efficient transfer learning. *Advances in Neural Information Processing Systems*, 35:12991–13005.

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

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022a. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, et al. 2022b. Supernaturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. *arXiv preprint arXiv:2204.07705*.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2017. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Jeffrey O Zhang, Alexander Sax, Amir Zamir, Leonidas Guibas, and Jitendra Malik. 2020. Side-tuning: a baseline for network adaptation via additive side networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 698–714. Springer.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena.

Han Zhou, Xingchen Wan, Ivan Vulić, and Anna Korhonen. 2023. Autopeft: Automatic configuration search for parameter-efficient fine-tuning. *arXiv preprint arXiv:2301.12132*.

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

A Hyperparameters of QST on the GLUE benchmark

The hyperparameters of QST on GLUE benchmark are shown in Table 7.

B Hyperparameters of QST on MMLU benchmark

The hyperparameters of QST on the MMLU benchmark are shown in Table 8.

Model	Dataset	RTE	MRPC	STS-B	CoLA	SST-2	QNLI	QQP	MNLI
	Optimizer	AdamW							
	Warmup Ratio	0.06							
	LR Schedule	Linear							
OPT-1.3B	Batch Size	32	8	32	32	32	8	8	32
	# Epochs	20							
	Learning Rate	2E-04							
	r	16							
	the rank of downsamples	16							
OPT-2.7B	Batch Size	16	8	16	16	16	8	8	16
	# Epochs	15							
	Learning Rate	2E-04							
	r	16							
	the rank of downsamples	16							
OPT-6.7B	Batch Size	8	4	8	8	8	4	4	8
	# Epochs	10							
	Learning Rate	2E-04							
	r	16							
	the rank of downsamples	16							

Table 8: The hyperparameters of QST on the GLUE benchmark.

	OPT-1.3B	OPT-2.7B	OPT-6.7B	OPT-13B	OPT-30B	OPT-66B	LLaMA-2-7B	LLaMA-2-13B	LLaMA-2-70B
Optimizer	AdamW								
Warmup Ratio	0.03								
LR Schedule	Constant								
Batch Size	8	8	4	2	1	1	4	2	1
# Epochs	5	5	3	3	2	2	3	2	2
Learning Rate	2E-04	2E-04	2E-04	1E-04	1E-04	1E-04	2E-04	2E-04	1E-04
r	16								
the rank of downsamples	16								

Table 9: The hyperparameters of QST on the MMLU benchmark.