HiFT: A Hierarchical Full Parameter Fine-Tuning Strategy

Yongkang Liu[1,2,5], **Yiqun Zhang**[1], **Qian Li**[3], **Tong Liu**[4,5], **Shi Feng**[1], **Daling Wang**[1]*, **Yifei Zhang**[1] **and Hinrich Schütze**[2,5]
[1]Northeastern University, China; [2]CIS, LMU Munich, Germany
[3]Shandong University, China; [4]Institute of Informatics, LMU Munich, Germany
[5]Munich Center for Machine Learning (MCML), Germany
misonsky@163.com,yiqunzhang@stumail.neu.edu.cn,TongLiu.physics@gmail.com
feiwangyuzhou@sdu.edu.cn,{fengshi,wangdaling,zhangyifei}@cse.neu.edu.cn

Abstract

Full-parameter fine-tuning (FPFT) has become the go-to choice for adapting language models (LMs) to downstream tasks due to its excellent performance. As LMs grow in size, fine-tuning the full parameters of LMs requires a prohibitively large amount of GPU memory. Existing approaches utilize zeroth-order optimizer to conserve GPU memory, which potentially compromises the performance of LMs as non-zero order optimizers tend to converge more readily on most downstream tasks. We propose a novel, memory-efficient, optimizer-independent, end-to-end hierarchical fine-tuning strategy, HiFT, which only updates a subset of parameters at each training step. HiFT significantly reduces the amount of gradients and optimizer state parameters residing in GPU memory at the same time, thereby reducing GPU memory usage. Our results demonstrate that: (1) HiFT achieves comparable performance with parameter-efficient fine-tuning and standard FPFT. (2) Results on six models show that HiFT reduces the number of trainable parameters by about 89.18% on average compared to FPFT. (3) HiFT supports FPFT of 7B models for 24G GPU memory devices under mixed precision without using any memory saving techniques. (4) HiFT supports various optimizers including AdamW, AdaGrad, SGD, etc. The source code link is https://github.com/misonsky/HiFT.

1 Introduction

Full-Parameter Fine-Tuning (FPFT) Language Models (LMs) have been a successful paradigm in various downstream tasks (Vaswani et al., 2017; Liu et al., 2020). However, as the size of LMs becomes larger, FPFT LMs require immense memory, which has become an obstacle to conducting research. One line of research to reduce memory is to use heterogeneous memory (Pudipeddi et al., 2020; Rajbhandari et al., 2021) (e.g., GPU, CPU, and NVMe memory) or distributed techniques (e.g., tensor parallelism (Shazeer et al., 2018; Shoeybi et al., 2019; Zhang et al.; Kim et al., 2023; Wu et al., 2023)). These strategies require parameter sharing across diverse devices and thus usually introduce a significant communication burden. Parameter-Efficient Fine-Tuning (PEFT) is another line of strategies for memory reduction, categorized into addition-based, selection-based, and reparametrization-based methods (Lialin et al., 2023). The addition-based methods (e.g., Prefix-Tuning (Li and Liang, 2021), AttentionFusion (Cao et al., 2022)) reduce the number of trainable parameters by only updating newly added parameters and freezing the weights of LMs. Although these methods reduce the number of parameters for fine-tuning, they expand the number of model parameters and increase the burden on forward propagation. The selection-based methods (e.g, BitFit (Zaken et al., 2022), LT-SFT (Ansell et al., 2022), FAR (Vucetic et al., 2022)), on the other hand, fine-tune a subset of model parameters, resulting in a performance gap with FPFT. The reparametrization-based methods (e.g., LoRA (Hu et al., 2022), KronA (Edalati et al., 2022), S4-model (Chen et al., 2023)) leverage low-rank decomposition to minimize the number of trainable parameters. Using low-rank representations inevitably leads to information loss and performance degradation. PEFT involves a trade-off between serving efficiency and quality. According to existing works (Raschka, 2023; Artur et al., 2023; Kourosh and Rehaan, 2023), FPFT still maintains advantages in performance on most benchmarks.

Some works have reduced memory usage for FPFT by removing the momentum state of the optimizer. LOMO (Lv et al., 2023) reduces the memory usage of the optimizer momentum and gradients by integrating gradient calculation and update. Nevertheless, LOMO requires forward propagation twice. In addition, LOMO forces the model to be 16-bit

*Corresponding Author

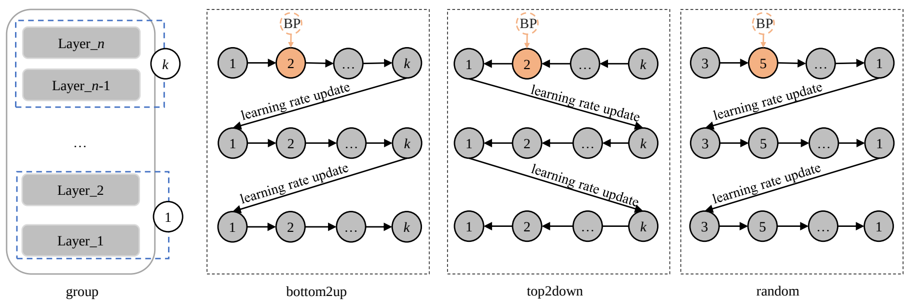

Figure 1: Schematic diagram of our HiFT. **group** represents the grouping operation of the layers. **bottom2up**, **top2down** and **random** are training strategies. Gray indicates that the corresponding parameters are in the frozen state, and brown indicates that the corresponding parameters are in the activated state. k is the number of groups, n is the number of layers of the given model, and BP denotes parameter update through back propagation.

quantized and uses the gradient checkpointing technique (Chen et al., 2016) to reduce memory usage while LOMO has limited memory savings in real-world scenarios. MeZO (Malladi et al., 2023) designs a zeroth-order optimizer to reduce memory usage. However, MeZO is unstable and performs poorly without prompts. These methods make the momentum optimizers unusable, while the momentum optimizers such as AdamW (Loshchilov and Hutter, 2017) have been proven to be superior in improving performance.

In this paper, we propose a novel memory-efficient **Hi**erarchical **F**ine-**T**uning (**HiFT**) strategy, adopting the idea of block-by-block training. HiFT divides the layers of the model into different groups (a group is a block.). At each training step, HiFT updates the parameters of one group while freezing the others. Compared to standard FPFT, HiFT leads to different groups of parameters being updated with different learning rates. This causes the model parameters to be updated in an inconsistent amplitude, which leads to a decrease in model performance. To solve this problem, we adopt to delay the learning rate update, which only updates the learning rate once when all layers of the model are updated. HiFT is also different from layer-wise training (Bengio et al., 2006), where the layer-wise training incrementally adds new layers to a pre-trained shallow model, only updating the newly added parameters at each training stage until all layers are updated. As a result, the layer-wise strategy produces accumulated errors at different training stages due to its pipeline training.

HiFT can significantly reduce the number of trainable parameters per training step. We only keep the momentum and gradients of the parameters that need to be updated on the GPU device due to only a portion of the parameters are updated at each training step. This helps to reduce the GPU memory usage of the optimizer states and gradients. HiFT supports full-parameter fine-tuning of a 7B model on devices with 24G memory. Our contributions are summarized as follows:

- We propose a novel, memory-efficient, optimizer-independent, end-to-end hierarchical fine-tuning strategy HiFT. Different from standard full parameter fine-tuning, HiFT achieves full-parameter fine-tuning in an asynchronous block-by-block manner.
- We show that the order of updates has no impact on model performance during asynchronous block-by-block updates, which provides a basis for block-by-block parallel updates of models in the future.
- Experiments show that HiFT achieves the same or even better performance than FPFT and PEFT on instruction fine-tuning, classification, generation, question answering and inference tasks with less GPU memory.

2 Related Work

Full-Parameter Fine-tuning FPFT fine-tunes the pre-trained LMs on specific tasks by updating all parameters (Sun et al., 2023; Lin et al., 2024; Ma et al., 2024), which requires massive computing power as the parameters of LMs increase. Mixed-precision training enables high-throughput computations by employing half-precision storage for parameters, activations, and gradients (Rajbhandari et al., 2020a; Narayanan et al., 2021). Staged training incrementally increases the amount of compute and reuse the compute from prior stages (Shen et al., 2022). These methods increase the parameter consumption when training precision or operators. LOMO (Lv et al., 2023) identifies the memory saving of SGD (Robbins and Monro, 1951), fuses the gradient computation and the parameter update in one step. MeZO (Malladi et al., 2023) designs a gradient-free method to update the model. Although it can reduce memory usage, its performance has a big gap than FPFT, especially when there is no prompt. These methods waste the superiority of momentum optimizers.

Parameter-Efficient Fine-tuning PEFT minimizes resource utilization from the perspective of parameters with additon, selection or decomposition methods (Lialin et al., 2023). The addition-based methods add and update new parameters with the weights of LMs frozen, such as Prefix-Tuning (Li and Liang, 2021), AttentionFusion (Cao et al., 2022), while the added parameters increase the burden on forward propagation. The selection-based methods fine-tune a subset of the parameters of LMs, such as BitFit (Zaken et al., 2022), LT-SFT (Ansell et al., 2022), FAR (Vucetic et al., 2022), but has a performance gap with FPFT. The reparametrization-based methods leverage low-rank decomposition to minimize the number of trainable parameters, such as LoRA (Hu et al., 2022), PHM (Karimi Mahabadi et al., 2021), KronA (Edalati et al., 2022), S4-model (Chen et al., 2023), while using low-rank representations inevitably leads to information loss and performance degradation. PEFT involves a trade-off between serving efficiency and quality.

Memory-Efficient Fine-tuning MEFT minimizes memory usage with heterogeneous memory (e.g., GPU, CPU and NVMe) or parallel methods (e.g., tensor and pipeline parallelism). In a layer-to-layer strategy (Pudipeddi et al., 2020), only the tensors necessary for the computation of a particular layer are transferred to GPU, while the remaining tensors are retained in CPU. ZeRO-Infinity (Rajbhandari et al., 2021) enables the partitioned states and tensors to CPU and NVMe. Tensor parallelism accelerates training by parallelizing tensor computations across different GPUs, but requires multiple global communications during each propagation (Shazeer et al., 2018; Shoeybi et al., 2019). Pipeline parallelism accelerates training by breaking the model into segments or layers and processing them sequentially in a pipeline fashion (Zhang et al.; Kim et al., 2023; Wu et al., 2023). These methods transfer massive memory to heterogeneous devices, although temporarily saving memory, still requires a large number of devices.

Different from existing works (Lv et al., 2023; Malladi et al., 2023), HiFT adopts the idea of block-by-block training to save memory of FPFT, and can be seamlessly integrated with any optimizer.

3 Approach

The training strategy of our **HiFT** is shown in Figure 1. We first present some necessary notations.

Notation Given the training dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, the goal of the training is to learn a model M with n layers, where N is the number of the training samples, (x_i, y_i) is the labeled data pair. We use P to represent the optimizer, and η_t to represent the learning rate schedule. The number of layers in each group is represented by m and the number of groups is represented by k. If m is divisible by n, then $k = n/m$, otherwise $k = \lfloor n/m \rfloor + 1$. Queue Q is used to store special identifiers that uniquely identify different layers. $S \in$ {"bottom2up","top2down","random"} represents the adopted update strategy.

Consider a pre-trained LM $f_{\theta_{pre}}$ parameterized by θ_{pre}. Let θ_{fpft} and θ_{hift} denote parameters after full fine-tuning and hierarchical full-parameter fine-tuning after *one training step*, respectively. Let $\mathcal{L}_\tau(\mathcal{D};\theta)$ be the objective to minimize during fine-tuning, with $\mathcal{D}$ being the input, θ being updated parameters, and τ being the task in fine-tuning. In the process of full fine-tuning, we optimize the model by adjusting its full parameters:

$$\theta_{fpft} = \underset{\theta_{pre}}{\operatorname{argmin}}\, \mathcal{L}_\tau(\mathcal{D};\theta_{pre}), \tag{1}$$

where the dimension of θ_{fpft}, $|\theta_{fpft}|$, equals the dimension of θ_{pre}, $|\theta_{pre}|$.

In the process of HiFT, only a subset of parameters are updated at one training step. More formally, with optimizing group $i \in \{1, ..., k\}$, we have:

$$\theta_{hift}^{(i)} = \underset{\beta_i \circ \theta_{hift}^{(i-1)}}{\operatorname{argmin}}\, \mathcal{L}(\mathcal{D}, \beta_i \circ \theta_{hift}^{(i-1)} + (1-\beta_i) \circ \theta_{hift}^{(i-1)}) \tag{2}$$

$$\theta_{hift}^{(1)} = \underset{\beta_1 \circ \theta_{pre}}{\operatorname{argmin}}\, \mathcal{L}(\mathcal{D}, \beta_1 \circ \theta_{pre} + (1-\beta_1) \circ \theta_{pre}), \tag{3}$$

where β_i denotes a fixed binary mask of parameters, with $\beta_i \in \{0,1\}^{|\theta_{pre}|}$, depending on the training strategy chosen in Figure 1. We simply denote $\theta_{hift}^{(k)}$ as θ_{hift}.

3.1 Hierarchical Training

FPFT has been proven to achieve the-state-of-art performance in most downstream tasks (Raschka,

Algorithm 1: HiFT Training Algorithm

```
Require: model $M$ with $n$ layers, number of layers
  per group $m$, batch size $B$, step budget $T$, optimizer
  $P$, parameter queue $Q$, update strategy $S$, learning
  rate schedule $\eta_t$
Initialize: Initialize queue $Q$ by layer identifier
UpdateStrategy($Q, S$)
for $t = 1, ..., T$ do
    a). Freeze all parameters of $M$;
    b). Sample batch $\mathcal{B} \subset \mathcal{D}$ with random seed $s$
    Select key features of layers to be updated
    c). $E \leftarrow$ QueueGetAndRemove($Q, m$)
    Removed elements added to tail of queue
    d). QueueAddTail($Q, E$)
    e). $\boldsymbol{\theta}_s \leftarrow$ SelectParameters($M, E$)
    f). Set requires_grad = True of parameters $\boldsymbol{\theta}_s$
    g). UpdateOptimizerParameter($P, \boldsymbol{\theta}_s$)
    h). ForwardPropagation($M, \mathcal{B}$)
    Preserve optimizer state of $\boldsymbol{\theta}_s$ within the GPU
    i). MoveOptimizerState2GPU($P, \boldsymbol{\theta}_s$)
    g). Backpropagation($P, \boldsymbol{\theta}_s, M$) & Clear
      gradients
    Keep optimizer state within the CPU
    k). MoveOptimizerState2CPU($P, \boldsymbol{\theta}_s$)
    if IsAllLayerUpdate($t, n, m$) then
        Update learning rate $\eta_t$
    end
    else
        Keep the learning rate $\eta_t$ constant
    end
end
```

2023; Artur et al., 2023; Kourosh and Rehaan, 2023). Standard FPFT updates all parameters of M at each training step, which requires a large amount of GPU memory to store forward and backward propagation parameters at the same time. Different from standard FPFT, HiFT only updates a part of the model parameters and freezes the remaining parameters at each training step, and achieves fine-tuning of all parameters through block-by-block updates. During the BP process, only the parameters that need to be updated will be stored in the GPU memory, which greatly reduces the GPU memory requirements for FPFT.

As shown in Figure 1, we divide the model into k groups and update only one group of parameters in each step. All groups are iterated in sequence until convergence. We provide three update strategies: **bottom2up** (B2U), **top2down** (T2D) and **random** (RAN). Different strategies only represent different orders of updates, e.g., bottom2up represents the update from the bottom to top. Note that random strategy only shuffles the grouping order before training, and maintains this order in the training process, which avoids the instability caused by constant changes in the update order. Here, the embedding layer is regarded as the bottom layer, and the head layer used for classification or generation is the top layer.

The detailed training process is shown in Algorithm 1. The first step is to determine the update strategy. During training, we freeze all parameters. The layers to be updated, denoted by E, are selected from the queue Q based on the parameter m. The selected layer E is removed from head of the queue Q and added to the tail of Q to wait for the next update. We select the parameter $\boldsymbol{\theta}_s$ that needs to be updated from M based on E, set the parameter $\boldsymbol{\theta}_s$ to a computable gradient state and set the update parameter group of optimizer P to $\boldsymbol{\theta}_s$. Before parameter updates, the states parameters (e.g., the gradient first moment estimation and second moment estimation of AdamW) of optimizer P related to $\boldsymbol{\theta}_s$ could be moved to GPU devices. After the completion of weight updates, the corresponding gradients are cleaned up and optimizer state parameters are moved to CPU. To update the learning rate η_t, we employ a delayed update strategy. Specifically, we adjust the learning rate once after updating all layers, which helps alleviate the instability issue arising from excessively updates in some layers, especially when fine-tuning deep models. By employing the successive update strategy, the number of parameters residing in GPU simultaneously reduces, thus lowering the GPU memory requirements of fine-tuned models.

Note that we provide a theoretical generalization bound for HiFT (Appendix A) and a theoretical memory analysis (Appendix B).

4 Experiments

Please refer to Appendix for baselines(C), datasets (D) and implementation details (F).

4.1 Results

Prompt results Table 1 reports the prompt-based fine-tuning results of the RoBERTa$_{\text{large}}$. HiFT uses the same prompt template (see Appendix G.3) as MeZO. We clearly observe that HiFT has an absolute performance advantage compared to gradient-free methods. Although gradient-free methods can reduce the memory usage of fine-tuning, there is still a huge gap in performance compared to gradient-based methods. Reducing memory usage at the expense of performance is not an ideal solution. Compared with standard FPFT and PEFT methods, HiFT still achieves competitive results. Table 2 reports the performance comparison of

	SST-2	SST-5	SNLI	MNLI	RTE	TREC
Task Type	—— sentiment ——		—— natural language inference ——			— topic —
Zero-shot†	79	35.5	50.2	48.8	51.4	32
Gradient-free methods: Num = 16						
LP†	76.0 (2.8)	40.3 (1.9)	66.0 (2.7)	56.5 (2.5)	59.4 (5.3)	51.3 (5.5)
MeZO†	90.5 (1.2)	45.5 (2.0)	68.5 (3.9)	58.7 (2.5)	64.0 (3.3)	76.9 (2.7)
MeZO(LoRA)†	91.4 (0.9)	43.0 (1.6)	69.7 (6.0)	64.0 (2.5)	64.9 (3.6)	73.1 (6.5)
MeZO(prefix)†	90.8 (1.7)	45.8 (2.0)	71.6 (2.5)	63.4 (1.8)	65.4 (3.9)	80.3 (3.6)
MeZO-Adam†	90.4 (1.4)	45.4 (1.5)	74.1 (2.7)	64.3 (0.8)	59.2 (11.1)	78.3 (1.4)
Gradient-based methods: Num = 16						
FPFT†	**91.9 (1.8)**	47.5 (1.9)	77.5 (2.6)	**70.0 (2.3)**	66.4 (7.2)	85.0 (2.5)
FT(LoRA)†	91.4 (1.7)	46.7 (1.1)	74.9 (4.3)	67.7 (1.4)	66.1 (3.5)	82.7 (4.1)
FT(prefix)†	**91.9 (1.0)**	47.7 (1.1)	**77.2 (1.3)**	66.5 (2.5)	**66.6 (2.0)**	**85.7 (1.3)**
HiFT	**91.9 (2.3)**	**47.8 (2.6)**	76.7 (3.5)	69.9 (1.9)	66.3 (4.5)	84.3 (4.1)
Gradient-free methods: Num = 512						
LP†	91.3 (0.5)	51.7 (0.5)	80.9 (1.0)	71.5 (1.1)	73.1 (1.5)	89.4 (0.5)
MeZO†	93.3 (0.7)	53.2 (1.4)	83.0 (1.0)	78.3 (0.5)	78.6 (2.0)	94.3 (1.3)
MeZO(LoRA)†	93.4 (0.4)	52.4 (0.8)	84.0 (0.8)	77.9 (0.6)	77.6 (1.3)	95.0 (0.7)
MeZO(prefix)†	93.3 (0.1)	53.6 (0.5)	84.8 (1.1)	79.8 (1.2)	77.2 (0.8)	94.4 (0.7)
MeZO-Adam†	93.3 (0.6)	53.9 (0.8)	85.3 (0.8)	79.6 (0.4)	79.2 (1.2)	95.1 (0.3)
Gradient-based methods: Num = 512						
FPFT†	93.9 (0.7)	55.9 (0.9)	**88.7 (0.8)**	**84.4 (0.8)**	82.7 (1.4)	97.3 (0.2)
FT(LoRA)†	**94.2 (0.2)**	55.3 (0.7)	88.3 (0.5)	83.9 (0.6)	**83.2 (1.3)**	97.0 (0.3)
FT(prefix)†	93.7 (0.3)	54.6 (0.7)	88.3 (0.7)	83.3 (0.5)	82.5 (0.8)	**97.4 (0.2)**
HiFT	**94.2 (0.6)**	**57.2 (0.8)**	88.1 (1.2)	83.8 (0.8)	82.6 (0.9)	96.7 (0.3)

Table 1: Performance of RoBERTa$_{\text{large}}$ based on prompt fine-tuning. LP: Linear probing; MeZO, MeZO(LoRA) and and MeZO(prefix): memory-efficient ZO-SGD with full-parameter tuning, LoRA, and prefix-tuning respectively; FPFT: fine-tuning with AdamW. All reported numbers are averaged accuracy (standard deviation). *Num* denotes the number of training examples per class. The parameter m of HiFT is set to 1. † means the result comes from Malladi et al. (2023)

TasK	**SST2**	**RTE**	**CB**	**BoolQ**	**WSC**	**WIC**	**MultiRC**	**COPA**	**ReCoRD**	**SQuAD**	**DROP**
Task type	——	——	——	classification	——	——	——	—multiple choice—		—generation—	
Zero-shot†	58.8	59.6	46.4	59.0	38.5	55.0	46.9	80.0	81.2	46.2	14.6
ICL†	87.0	62.1	57.1	66.9	39.4	50.5	53.1	87.0	82.5	75.9	29.6
LP†	93.4	68.6	67.9	59.3	63.5	60.2	63.5	55.0	27.1	3.7	11.1
MeZO†	91.4	66.1	67.9	67.6	63.5	61.1	60.1	**88.0**	81.7	84.7	30.9
MeZO (LoRA)†	89.6	67.9	66.1	73.8	64.4	59.7	61.5	84.0	81.2	83.8	31.4
MeZO (prefix)†	90.7	70.8	69.6	73.1	60.6	59.9	63.7	87.0	81.4	84.2	28.9
FPFT†	92.0	70.8	**83.9**	77.1	63.5	**70.1**	71.1	79.0	74.1	84.9	31.3
FT(LoRA)	92.4	74.5	83.3	77.8	**64.5**	68.2	70.2	86.5	**82.6**	85.2	30.8
FT(IA3)	92.5	76.7	82.4	76.5	63.2	67.7	69.1	87.3	81.7	86.4	29.6
FT(prefix)	93.6	77.8	82.9	77.4	63.2	68.3	70.4	87.2	80.4	84.2	31.7
HiFT	**94.4**	**78.7**	83.1	**78.1**	63.6	69.4	**71.9**	**88.0**	81.4	**86.1**	**32.7**

Table 2: Experiments on OPT-13B (with 1000 examples). ICL: in-context learning; LP: linear probing; FPFT: full fine-tuning; Prefix: prefix-tuning. All experiments use prompts in Appendix G.3. † means the result comes from Malladi et al. (2023)

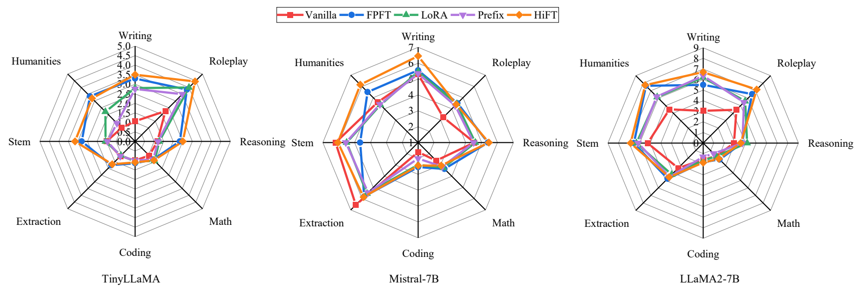

Figure 2: Category-wise scores of different fine-tuning methods on MT-bench. The detailed results are shown in Table 7 (Appendix G).

OPT-13B using different fine-tuning methods on different tasks. We observe that among the 11 tasks, HiFT enjoys performance advantages in 7 tasks. This fully demonstrates the universal effectiveness of HiFT fine-tuning method.

Instruction Fine-tuning Figure 2 and Table 7 (Appendix G) report the results of instruction fine-

	E2E NLG Challenge					
Model & Method	**BLUE**	**NIST**	**MET**	**ROUGE-L**	**CIDEr**	**AVE**
GPT-2 M (FPFT)†	68.20	8.62	46.20	71.00	2.47	39.30
GPT-2 M (Adapter^{L})†	66.30	8.41	45.00	69.80	2.40	38.38
GPT-2 M (Adapter^{H})†	67.30	8.50	46.00	70.70	2.44	38.99
GPT-2 M (FPFT^{Top2})†	68.10	8.59	46.00	70.80	2.41	39.18
GPT-2 M (PreLayer)†	69.70	**8.81**	46.10	71.40	2.49	39.70
GPT-2 M (LoRA)†	68.90	8.76	46.60	**71.50**	**2.53**	39.66
HiFT	**69.40**	8.67	**46.77**	71.26	2.46	**39.71**
GPT-2 L (FPFT)†	68.50	8.78	46.00	69.90	2.45	39.13
GPT-2 L (Adapter^{L})†	69.10	8.68	46.30	71.40	**2.49**	39.59
GPT-2 L (PreLayer)†	**70.30**	8.85	46.20	71.70	2.47	39.90
GPT-2 L (LoRA)†	70.10	8.83	**46.80**	72.00	2.47	40.04
HiFT	**70.30**	**8.86**	46.64	**72.22**	2.48	**40.10**

Table 3: GPT-2 medium (M) and large (L) with different fine-tuning methods on the E2E NLG Challenge. † indicates numbers published in prior works (Gao et al., 2024; Hu et al., 2022).

Models	**Mehtods**	**ViGGO**	**SQL Generation**	**GSM8k**
LLaMA2-7B	Vanilla†	0.93	3.50	14.00
	FPFT†	94.86	86.60	**30.00**
	LoRA†	92.05	85.93	22.87
	HiFT	**94.88**	**87.15**	29.85
LLaMA2-13B	Vanilla†	2.34	22.20	28.00
	FPFT†	**95.79**	89.20	47.00
	LoRA†	95.32	87.94	35.94
	HiFT	95.66	**90.33**	**48.01**

Table 4: Performance comparison of different fine-tuning methods for LLaMA-7B and 13B. The best result is in bold and the second best result is underlined.

tuning for TinyLLaMA, Mistral-7B, and LLaMA2-7B on MT-bench (Zheng et al., 2024). We fine-tune these models on Alpaca GPT-4 dataset (Taori et al., 2023). Compared with standard FPFT and PEFT fine-tuning, HiFT has performance advantages in 5 of 8 dimensions on TinyLlaMa, 4 of 8 dimensions on Mistral-7B, and 5 of 8 dimensions on LLaMA2-7B. In terms of overall performance, HiFT achieves the best results among the three models compared to other fine-tuning methods.

No prompt results Figure 5 (Appendix G) shows the performance of RoBERTa$_{\text{base}}$ and RoBERTa$_{\text{large}}$ using different fine-tuning strategies on eight tasks. The HiFT performances of RoBERTa$_{\text{base}}$ have competitive advantages with standard FPFT on datasets such as SST-2, MNLI, QNLI and QQP, and HiFT has achieved a weak performance advantage on the MRPC dataset. We observe that HiFT has certain performance advantages on most datasets compared to most PEFT methods such as BitFit, Prefix and Adapter. We get similar conclusions on RoBERTa$_{\text{large}}$. The number of layers of model RoBERTa$_{\text{large}}$ is about twice that of RoBERTa$_{\text{base}}$, which reflects to a certain extent that HiFT is not affected by the depth of the model. Table 3 reports the results of GPT-2 including medium and large on the E2E dataset. Compared with standard FPFT and PEFT methods, HiFT achieves competitive results on GPT-2 medium and large. To verify the generalizability of HiFT, we conduct experiments on more complex tasks such as ViGGO (Juraska et al., 2019), SQL generation (b mc2, 2023), and GSM8K (Cobbe et al., 2021). Table 4 reports the performance comparison of different fine-tuning methods on these benchmarks. We can observe that HiFT significantly outperforms standard FPFT and LoRA on these three benchmarks. This fully demonstrates the universal effectiveness of HiFT. Another phenomenon is that the performance of LoRA is significantly inferior to standard FPFT and HiFT. To a certain extent, this demonstrates that full parameter fine-tuning is more effective in capturing data characteristics for complex tasks and offers better performance advantages compared to LoRA. u

4.2 Memory Efficiency

To evaluate the effectiveness of HiFT in reducing memory, we compare HiFT with most PEFT methods in terms of memory and speed. Table 5 reports the memory and speed comparison of different fine-tuning methods on RoBERTa$_{\text{base}}$, RoBERTa$_{\text{large}}$ and LLaMA2-7B models. We can observe that HiFT has an absolute advantage in GPU memory usage. HiFT reduces memory usage from three aspects: **gradients**, **optimizer states**, and **residual states**. Since HiFT only updates a small number of parameters in each step, this directly reduces the amount of trainable parameters in each training step, and the corresponding gradient parameters and optimizer state parameters also be reduced in the same proportion. When only some layer parameters are updated in each step, the amount of parameters tracking gradients in the calculation graph is reduced, including the amount of parameters in the activations, so HiFT also reduces the amount of parameters in residual states. This is why HiFT is memory efficient. These PEFT methods introduce new parameters as trainable parameters while freezing the weights of the original LLMs, which reduces the usage of GPU memory by reducing the trainable parameters. Introducing new parameters results in larger memory requirements for the forward computation of fine-tuning. Besides, reducing the number of trainable parameters will reduce the representation ability of models and make them unable to fit complex tasks well.

Methods	RoBERTa-Base				RoBERTa-large				LLaMA2-7B			
	AdamW		SGD		AdamW		SGD		AdamW		SGD	
	Memory(GB)	Speed(step/s)	Memory(GB)	Speed(step/s)	Memory(GB)	Speed(step/s)	Memory(GB)	Speed(step/s)	Memory(GB)	Speed(step/s)	Memory(GB)	Speed(step/s)
FPFT	5.67	10.28	4.73	11.51	15.25	5.76	12.60	6.49	OOM	----	OOM	----
LoRA(r=8)	<u>2.63</u>	13.09	2.62	13.41	6.95	8.19	6.94	8.71	43.24	1.31	43.21	1.31
IA3	2.70	<u>14.17</u>	2.70	14.93	7.13	8.61	7.12	8.95	43.22	1.33	43.22	1.33
Prefix	2.66	**16.34**	<u>2.61</u>	<u>16.78</u>	<u>6.64</u>	**9.24**	<u>6.56</u>	<u>11.22</u>	<u>40.69</u>	1.37	<u>40.24</u>	<u>1.38</u>
HiFT	**2.62**	13.53	**2.58**	**18.4**	**6.62**	<u>8.99</u>	**6.55**	**11.71**	**40.11**	**2.31**	**40.01**	**2.40**

Table 5: Memory and speed comparison of different fine-tuning methods with mixed precision. The batch size and sequence length are set to 8 and 512. Dataset used by RoBERTa$_{\text{base}}$ and RoBERTa$_{\text{large}}$ is CoLA, and that used by LLaMA2-7B is E2E. All tests were performed on A100 with 80G memory.

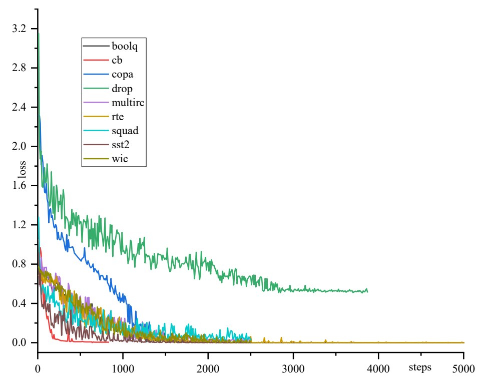

Figure 3: Loss curves of OPT-13B on different datasets. The parameter m of HiFT is set to 1.

We compare LOMO (Lv et al., 2023) and MeZO (Malladi et al., 2023) based on LLaMA2-7B. Following the settings in Table 5, LOMO reports running out of memory on an A100 with 80GB. The memory used by MeZO is about 30GB. MeZO has a memory usage advantage over HiFT due to it being a gradient-free method. Nevertheless, HiFT significantly outperforms MeZO in terms of performance. Among gradient-based methods, HiFT has advantages in memory.

To evaluate the universality of HiFT in reducing memory, we conduct extensive experiments on different optimizers (i.e., AdamW, SGDM, SGD, Adafactor and Adagrad) based on multiple LMs including RoBERTa$_{\text{base}}$, RoBERTa$_{\text{large}}$, GPT-2$_{\text{large}}$, GPT-Neo (2.7B) and LLaMA-2 (7B). Table 8 to Table 12 (Appendix G) reports the memory usage of the parameters, gradients, optimizer states and residual states under FPFT and HiFT. When using mixed precision, HiFT can save about 44.82%-53.69% of memory on RoBERTa$_{\text{base}}$, about 48.04%-56.60% of memory on RoBERTa$_{\text{large}}$, about 48.20%-54.27% of memory on GPT-2$_{\text{large}}$, about 28.99%-50.69% of memory on GPT-Neo and about 65.31%-76.65% of memory on LLaMA compared with FPFT.

4.3 Wallclock Time Efficiency

In this section, we measure the wallclock time efficiency of HiFT compared to standard FPFT and PEFT methods, with respect to different model sizes. We conduct our experiments on A100 with 80GB GPU memory. Table 5 reports the wallclock time results for different fine-tuning methods using different optimizers. We can observe that as the number of model parameters increases, the wallclock speed of HiFT gradually gains an advantage. When using the AdamW optimizer, although HiFT is slower than prefix on the RoBERTa$_{\text{base}}$ model, it is nearly as fast as the prefix method on RoBERTa$_{\text{large}}$ and faster than PEFT methods on the LLaMA2-7B model. Specifically, on LLaMA2-7B model, HiFT is 1.76× that of LoRA, 1.73× that of IA3, and 1.68× that of prefix. When using the SGD optimizer, HiFT outperforms PEFT and the standard FPFT approach across all models. For LLaMA2-7B model, HiFT is 1.83× that of LoRA, 1.80× that of IA3, and 1.74× that of prefix.

When using the AdamW optimizer, each step of HiFT has a communication cost between the CPU and GPU. The peak communication parameters are shown as the #Sta values in Table 8 to Table 12. The communication cost has limited impact on the speed of HiFT. There are several main reasons: i) The number of communication parameters is small even zero. HiFT is an optimizer-independent method that supports various optimizers. When using SGD, the peak communication parameter is zero. When using Adafactor, the peak communication parameter is 0.19MB for RoBERTa$_{\text{base}}$, 0.21MB for RoBERTa$_{\text{large}}$, and 0.33MB for LLaMA2-7B. ii) when the required amount of computation reaches the bottleneck of the device, the number of parameters processed per second by the device will no longer increase. Even if the GPU memory is large enough to load parameters, the training speed will not be greatly improved because the computing capability of the device per second is limited. iii) HiFT updates only a subset of parameters at each step, reducing the number of trainable parameters and cutting off gradient propagation to shallow layers. This significantly decreases the computation needed for fine-tuning, thereby increasing the speed. This is why HiFT still has a speed advantage over LLaMA2-7B even with the AdamW optimizer.

4.4 Stability of Training

In order to explore the stability of HiFT training, we report the loss curves of OPT-13B on different datasets. As shown in Figure 3, we can observe that during the training process, the loss curve fluctuates within a reasonable range and converges steadily on different datasets. This fully demonstrates that HiFT strategy does not affect the convergence of models. HiFT adopts a delayed learning rate update strategy, which ensures that the update amplitude of parameters in different blocks is consistent and avoids oscillation during the update process.

4.5 Trainable Parameter

Figure 6 (e) reports the changes in the amount of peak fine-tuning parameters under HiFT at different model sizes. We observe that as the number of model parameters increases, the proportion of peak trainable parameters gradually decreases. When fine-tuning the 13B model, the peak amount of fine-tunable parameters is only 2.44% of the original model parameter amount.

Figure 6 shows the percentage of memory used by the parameters of each part when fine-tuning LLaMA2-7B under FPFT and HiFT with the AdamW optimizer. Under FPFT, the optimizer states occupy the most memory. When fine-tuning 32-bit precision (Figure 6 (a)), the memory occupied by residual states is second only to the optimizer state. When mixed precision fine-tuning (Figure 6 (c)), the memory used by model parameters exceeds the memory used by residual states is secondary to the optimizer states. The main reason is that in mixed precision training, both 32-bit and half-precision parameters exist at the same time. Therefore, model parameters occupy more memory in mixed precision. HiFT significantly reduces the memory usage of gradients and optimizer states. Therefore, when using HiFT for full-parameter fine-tuning, the main memory-consuming parts are model parameters and residual states.

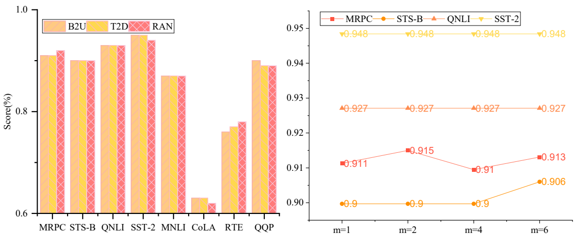

Figure 4: The left shows the performance of HiFT of RoBERTa$_{\text{base}}$ under **B2U**, **T2D** and **RAN** strategies, respectively. The right shows the performance of HiFT of RoBERTa$_{\text{base}}$ under different grouping settings, where m is the number of layers in each group.

4.6 Impact of Strategy

The left plot of Figure 4 reports the performance of RoBERTa$_{\text{base}}$ using B2U, T2D and RAN strategies. We observe that the order of updates has almost no effect on the performance of the model. It is an interesting phenomenon that the model still achieves competitive results even when updated in a random order. Changing the update order does not affect the position of the corresponding layer in the model, which is the reason why the performance is not affected. We believe that this phenomenon provides support for hierarchical parallel fine-tuning of large-scale models in the future.

4.7 Impact of Grouping

The right plot of Figure 4 reports the impact of different grouping settings on model performance. Although different grouping settings can cause fluctuations in model performance, the overall impact is negligible. We use the learning rate delayed update strategy, which updates the learning rate only after all layers are updated once. This strategy ensures that the learning rate used to update the parameters of each layer is the same in each training step, which helps to prevent the model performance from decreasing due to the update of some parameters being too fast in the hierarchical update process.

Conclusion

We propose an end-to-end hierarchical full-parameter fine-tuning strategy, HiFT, which groups the model parameters and updates a single group of parameters per training step. The number of trainable parameters per training step greatly reduce, which lowers the GPU memory usage of the corresponding gradients, optimizer state parameters, and activations. HiFT lowers the barrier of full-parameter fine-tuning of language models and supports full-parameter fine-tuning of a 7B model on a 24G memory device.

Limitations

Although HiFT achieves the performance of standard full-parameter fine-tuning at a lower GPU memory cost, there are still some shortcomings. HiFT divides the model by layers, and the maximum division limit is the number of layers of the model. Due to the limitation of the number of layers, HiFT cannot break through the number of model layers for finer-grained division. When the model width is large, it limits HiFT's capabilities. On the other hand, after dividing the model, the number of parameters in each group is different, and the GPU memory usage fluctuates during the fine-tuning process. The peak memory occupied by the fine-tuned model is the decisive factor that determines whether the model is able to be fine-tuned on a certain device. This fluctuation in memory usage during fine-tuning prevents us from fully utilizing resources.

Acknowledgement

We would like to thank reviewers for their constructive comments. The project is supported by the National Natural Science Foundation of China (62172086,62272092) and DFG (grant SCHU 2246/14-1). The project is also supported by China Scholarship Council.

References

Alan Ansell, Edoardo Ponti, Anna Korhonen, and Ivan Vulić. 2022. Composable sparse fine-tuning for cross-lingual transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1778–1796.

Niederfahrenhorst Artur, Hakhamaneshi Kourosh, and Ahmad Rehaan. 2023. *Fine-Tuning LLMs: LoRA or Full-Parameter? An in-depth Analysis with Llama-2*.

b mc2. 2023. sql-create-context dataset.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. 2006. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19.

Jin Cao, Chandana Satya Prakash, and Wael Hamza. 2022. Attention fusion: a light yet efficient late fusion mechanism for task adaptation in nlu. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 857–866.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14.

Jiaao Chen, Aston Zhang, Xingjian Shi, Mu Li, Alex Smola, and Diyi Yang. 2023. Parameter-efficient fine-tuning design spaces. *arXiv preprint arXiv:2301.01821*.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *ArXiv*, abs/1604.06174.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Marie-Catherine De Marneffe, Mandy Simons, and Judith Tonhauser. 2019. The commitmentbank: Investigating projection in naturally occurring discourse. In *proceedings of Sinn und Bedeutung*, volume 23, pages 107–124.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2368–2378.

John C. Duchi, Elad Hazan, and Yoram Singer. 2010. Adaptive subgradient methods for online learning and stochastic optimization. In *COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel, June 27-29, 2010*, pages 257–269. Omnipress.

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*.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2021. Making pre-trained language models better few-shot learners. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3816–3830, Online. Association for Computational Linguistics.

Ziqi Gao, Qichao Wang, Aochuan Chen, Zijing Liu, Bingzhe Wu, Liang Chen, and Jia Li. 2024. Parameter-efficient fine-tuning with discrete fourier transform. *arXiv preprint arXiv:2405.03003*.

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. 2022. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Juraj Juraska, Kevin Bowden, and Marilyn Walker. 2019. Viggo: A video game corpus for data-to-text generation in open-domain conversation. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 164–172.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34:1022–1035.

Daniel Khashabi, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. 2018. Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 252–262.

Taebum Kim, Hyoungjoo Kim, Gyeong-In Yu, and Byung-Gon Chun. 2023. Bpipe: Memory-balanced pipeline parallelism for training large language models. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 16639–16653. PMLR.

Hakhamaneshi Kourosh and Ahmad Rehaan. 2023. *Fine-Tuning Llama-2: A Comprehensive Case Study for Tailoring Models to Unique Applications*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059.

Hector Levesque, Ernest Davis, and Leora Morgenstern. 2012. The winograd schema challenge. In *Thirteenth international conference on the principles of knowledge representation and reasoning*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597.

Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. 2023. Scaling down to scale up: A guide to parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.15647*.

Peiqin Lin, Shaoxiong Ji, Jörg Tiedemann, André FT Martins, and Hinrich Schütze. 2024. Mala-500: Massive language adaptation of large language models. *arXiv preprint arXiv:2401.13303*.

Zhaojiang Lin, Andrea Madotto, and Pascale Fung. 2020. Exploring versatile generative language model via parameter-efficient transfer learning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 441–459.

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.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020. Multilingual denoising pretraining for neural machine translation. *Trans. Assoc. Comput. Linguistics*, 8:726–742.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Kai Lv, Yuqing Yang, Tengxiao Liu, Qinghui Gao, Qipeng Guo, and Xipeng Qiu. 2023. Full parameter fine-tuning for large language models with limited resources. *CoRR*, abs/2306.09782.

Bolei Ma, Ercong Nie, Shuzhou Yuan, Helmut Schmid, Michael Färber, Frauke Kreuter, and Hinrich Schütze. 2024. Topro: Token-level prompt decomposition for cross-lingual sequence labeling tasks. *arXiv preprint arXiv:2401.16589*.

Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D. Lee, Danqi Chen, and Sanjeev Arora. 2023. Fine-tuning language models with just forward passes. *CoRR*, abs/2305.17333.

Vaishnavh Nagarajan and J. Zico Kolter. 2019. Uniform convergence may be unable to explain generalization in deep learning. *CoRR*, abs/1902.04742.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 2021. Efficient large-scale language model training on GPU clusters using megatron-lm. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2021, St. Louis, Missouri, USA, November 14-19, 2021*, page 58. ACM.

Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. 2017. The e2e dataset: New challenges for end-to-end generation. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 201–206.

Abhishek Panigrahi, Nikunj Saunshi, Haoyu Zhao, and Sanjeev Arora. 2023. Task-specific skill localization in fine-tuned language models. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 27011–27033. PMLR.

Mohammad Taher Pilehvar and Jose Camacho-Collados. 2019. Wic: the word-in-context dataset for evaluating context-sensitive meaning representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1267–1273.

Bharadwaj Pudipeddi, Maral Mesmakhosroshahi, Jinwen Xi, and Sujeeth Bharadwaj. 2020. Training large neural networks with constant memory using a new execution algorithm. *CoRR*, abs/2002.05645.

Ning Qian. 1999. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020a. Zero: memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*, page 20. IEEE/ACM.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020b. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE.

Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. 2021. Zero-infinity: breaking the GPU memory wall for extreme scale deep learning. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2021, St. Louis, Missouri, USA, November 14-19, 2021*, page 59. ACM.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392.

Sebastian Raschka. 2023. *Finetuning LLMs with LoRA and QLoRA: Insights from Hundreds of Experiments*.

Herbert Robbins and Sutton Monro. 1951. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. 2011. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*.

Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, Ryan Sepassi, and Blake A. Hechtman. 2018. Mesh-tensorflow: Deep learning for supercomputers. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 10435–10444.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4603–4611. PMLR.

Sheng Shen, Pete Walsh, Kurt Keutzer, Jesse Dodge, Matthew E. Peters, and Iz Beltagy. 2022. Staged training for transformer language models. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 19893–19908. PMLR.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053.

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.

Xianghui Sun, Yunjie Ji, Baochang Ma, and Xiangang Li. 2023. A comparative study between full-parameter and lora-based fine-tuning on chinese instruction data for instruction following large language model. *CoRR*, abs/2304.08109.

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.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Danilo Vucetic, Mohammadreza Tayaranian, Maryam Ziaeefard, James J Clark, Brett H Meyer, and Warren J Gross. 2022. Efficient fine-tuning of bert models on the edge. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1838–1842. IEEE.

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. In *BlackboxNLPEMNLP*.

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. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of NAACL-HLT*, pages 1112–1122.

Shaohua Wu, Xudong Zhao, Shenling Wang, Jiangang Luo, Lingjun Li, Xi Chen, Bing Zhao, Wei Wang, Tong Yu, Rongguo Zhang, Jiahua Zhang, and Chao Wang. 2023. YUAN 2.0: A large language model with localized filtering-based attention. *CoRR*, abs/2311.15786.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*.

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.

Sheng Zhang, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, Kevin Duh, and Benjamin Van Durme. 2018. Record: Bridging the gap between human and machine commonsense reading comprehension. *arXiv preprint arXiv:1810.12885*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: open pre-trained transformer language models. *CoRR*, abs/2205.01068.

Zheng Zhang, Donglin Yang, Yaqi Xia, Liang Ding, Dacheng Tao, Xiaobo Zhou, and Dazhao Cheng. Mpipemoe: Memory efficient moe for pre-trained models with adaptive pipeline parallelism. In *IEEE International Parallel and Distributed Processing Symposium, IPDPS 2023, St. Petersburg, FL, USA, May 15-19, 2023*, pages 167–177. IEEE.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2024. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

A Generalization Bound for HiFT

In this section, we establish the generalization bound for HiFT, first building upon a quantization assumption as in Panigrahi et al. (2023). It is important to note that quantization is a common practical consideration; for instance, in our work, we implement a 32-bit quantization precision.

Assumption 1. (Quantization bound) *Given model parameters θ, we denote $\bar{q}(\theta)$ to be the parameter that quantizes every parameter into the q given values. Then there exist $\varepsilon > 0$ s.t. for any sample x_i with label y_i at any training step, we have*

$$|\mathcal{L}((x_i, y_i); \bar{q}(\theta)) - \mathcal{L}((x_i, y_i); \theta)| \leq \varepsilon. \tag{4}$$

Assumption 2. (Uniform convergence generalization bound for subset parameter fine-tuning) *Following Panigrahi et al. (2023), we deviate from the classical uniform convergence generalization bound (Nagarajan and Kolter, 2019) to get a tighter uniform convergence generalization bound for HiFT:*

$$\begin{aligned}&\mathcal{L}_{test}(\theta^{(i)}_{hift}) - \mathcal{L}_{train}(\theta^{(i)}_{hift})\\ &\qquad \leq \sup_{\widetilde{\theta}^{(i)}_{hift}\in\Theta} |\mathcal{L}_{test}(\widetilde{\theta}^{(i)}_{hift}) - \mathcal{L}_{train}(\widetilde{\theta}^{(i)}_{hift})|,\end{aligned} \tag{5}$$

where Θ denotes the subset of parameter space, $\theta^{(i)}_{hift}$ being the parameter after i-th optimizing step at one training step.

Theorem 3. (HiFT generalization bound) *Under Assumption 1 and 2, we have the following generalization bound for HiFT:*

$$\begin{aligned}&\mathcal{L}_{test}(\theta^{(k)}_{hift}) - \mathcal{L}_{test}(\theta^*)\\ &\quad \leq 4k\epsilon + 2\sum_{i=1}^{k} \sup_{\widetilde{\theta}^{(i)}} |\mathcal{L}_{test}(\bar{q}(\widetilde{\theta}^{(i)})) - \mathcal{L}_{train}(\bar{q}(\widetilde{\theta}^{(i)}))|\\ &\quad + \mathcal{L}_{test}(\theta^{(k)*}) - \mathcal{L}_{test}(\theta^*),\end{aligned} \tag{6}$$

where θ^* denotes the parameter with the best test performance, $\widetilde{\theta}^{(i)}$ is in the space of $\beta_i \circ \theta_{pre}$ and $\theta^{(i)*}$ denotes the parameter with the best test performance when only changing the subset parameter $\beta_i \circ \theta_{pre}$. With probability at least $1-\delta$, the second term $2\sum_{i=1}^{k}\sup_{\widetilde{\theta}^{(i)}} |\mathcal{L}_{test}(\bar{q}(\widetilde{\theta}^{(i)})) - \mathcal{L}_{train}(\bar{q}(\widetilde{\theta}^{(i)}))|$ can be further bounded:

$$\begin{aligned}&2\sum_{i=1}^{k} \sup_{\widetilde{\theta}^{(i)}} |\mathcal{L}_{test}(\bar{q}(\widetilde{\theta}^{(i)})) - \mathcal{L}_{train}(\bar{q}(\widetilde{\theta}^{(i)}))|\\ &\qquad \leq 2\sum_{i=1}^{k}\sqrt{\frac{s_i \log q + \log(1/\delta)}{N}},\end{aligned} \tag{7}$$

where s_i denotes the number of parameters in each optimizing group i.

Proof. We first derive HiFT generalization bound between the objective with parameters after a first step of optimization at one training step $\mathcal{L}_{test}(\theta^{(1)}_{hift})$ and the objective with parameters that has the best test performance $\mathcal{L}_{test}(\theta^*)$:

$$\begin{aligned}&\mathcal{L}_{test}(\theta^{(1)}_{hift}) - \mathcal{L}_{test}(\theta^*)\\ &\quad \leq 4\epsilon\\ &\quad + 2\sup_{\widetilde{\theta}^{(1)}} |\mathcal{L}_{test}(\bar{q}(\widetilde{\theta}^{(1)})) - \mathcal{L}_{train}(\bar{q}(\widetilde{\theta}^{(1)}))|\\ &\quad + \mathcal{L}_{test}(\theta^{(1)*}) - \mathcal{L}_{test}(\theta^*),\end{aligned} \tag{8}$$

with probability at least $1-\delta$, the second term can be bounded:

$$\begin{aligned}&2\sup_{\widetilde{\theta}^{(1)}} |\mathcal{L}_{test}(\bar{q}(\widetilde{\theta}^{(1)})) - \mathcal{L}_{train}(\bar{q}(\widetilde{\theta}^{(1)}))|\\ &\qquad \leq 2\sqrt{\frac{s_1 \log q + \log(1/\delta)}{N}}\end{aligned} \tag{9}$$

The above inequality can be shown by considering Theorem D.2 in Panigrahi et al. (2023) and taking $\Theta_N = 1$.

Similarly, we can have:

$$\begin{aligned}&\mathcal{L}_{test}(\theta^{(i)}_{hift}) - \mathcal{L}_{test}(\theta^{(i-1)}_{hift})\\ &\quad \leq 4\epsilon\\ &\quad + 2\sup_{\widetilde{\theta}^{(i)}} |\mathcal{L}_{test}(\bar{q}(\widetilde{\theta}^{(i)})) - \mathcal{L}_{train}(\bar{q}(\widetilde{\theta}^{(i)}))|\\ &\quad + \mathcal{L}_{test}(\theta^{(i)*}) - \mathcal{L}_{test}(\theta^{(i-1)}_{hift})\end{aligned} \tag{10}$$

Summing over the above terms with $i = \{1, ..., k\}$ completes the proof of this theorem.

B Memory Analysis

According to previous work (Lv et al., 2023; Malladi et al., 2023), the main components that consume GPU memory during the fine-tuning process include the weight parameter, optimizer states, gradients, and calculation of residual states (i.e, activations, temporary buffers and fragmented memory) (Rajbhandari et al., 2020b). In this section, we give theoretical analysis on the GPU memory advantages of HiFT strategy from the perspectives of **weight parameter**, **optimizer states** and **gradients** [1]. Assuming the model is fine-tuned using the AdamW optimizer with 32-bit precision, we employ ζ_1, ζ_2, ζ_3 to represent the GPU memory used by weight parameter, optimizer states and gradients

[1]Since the GPU memory occupied by forward activations is related to the model implementation, batch size and sentence length, we analyze the GPU memory requirements of internal variables through experiments.

respectively. AdamW optimizer stores the gradient first moment estimation and second moment estimation, which means that the optimizer state parameter ζ_2 is two times larger than weight parameter ζ_1 (i.e., $\zeta_2 = 2 * \zeta_1$). The gradient parameters typically correspond to the parameters updated in the model (i.e., $\zeta_3 = \zeta_1$). Therefore, the number of gradient parameters ζ_3 is the same as the number of parameters ζ_1 that need to be updated in the model. Therefore, for standard FPFT, the GPU memory required for these parameters is as follows:

$$\begin{aligned}\zeta_{fpft} &= \zeta_1 + \zeta_2 + \zeta_3 \\ &= \zeta_1 + 2\zeta_1 + \zeta_1 \\ &= 4\zeta_1\end{aligned} \tag{11}$$

Taking the fine-tuning of a 7B model at 32 precision using the AdamW optimizer as an example, the ζ_1 is about 26.08G. Theoretically, the GPU memory required for fine-tuning these three parts of the 7B model is approximately 104.32 GB. If considering GPU memory occupied by forward activations and the impact of batch size and sentence length, the actual scenario FPFT requires more GPU memory than 104.32 GB. Under the HiFT training strategy, since only one group of parameters is updated for each training step, only the gradients of the updated parameters and the corresponding optimizer states are stored in the GPU according to Algorithm 1. The weight parameter need to reside in the GPU memory for forward propagation. Therefore, the average GPU memory required for each training step is as follows:

$$\begin{aligned}\zeta_{hift} &= \zeta_1 + \frac{\zeta_2}{k} + \frac{\zeta_3}{k} \\ &= \frac{k+3}{k} * \zeta_1\end{aligned} \tag{12}$$

Compared with FPFT, the memory saved by HiFT in model parameters, gradients and optimizer states is:

$$\begin{aligned}\Delta\zeta &= \zeta_{fpft} - \zeta_{hift} \\ &= \frac{3k-3}{k} * \zeta_1\end{aligned} \tag{13}$$

In addition to these computable fixed parameters, HiFT can reduce the number of activation-related parameters that simultaneously reside in memory, which is discussed in the experimental section. Considering the embedding layer, task-related head layer and 32 hidden layers, LLaMA-7B has $n = 34$ layers. When $m = 1$, it can be deduced that $k = 34$, and the required GPU memory can be inferred to be $\zeta_{hift} \approx 31.13G$, the GPU memory saving is about 73.19G compared with FPFT.

C Baselines

Language Models include **RoBERTa** (Liu et al., 2019) with **base** and **large** versions, **GPT-2** (Radford et al., 2019) with **medium** and **large** versions, **LLaMA** (Touvron et al., 2023) with **7B** and **13B** versions, and **OPT-13B** (Zhang et al., 2022).

Fine-Tuning strategies include **BitFit** (Zaken et al., 2022), **Adapter** (Houlsby et al., 2019), **Prefix** (Lester et al., 2021), **LoRA** (Hu et al., 2022), **MeZO** (Malladi et al., 2023), **S4** (Chen et al., 2023), **Adapter$^{\mathbf{L}}$** (Lin et al., 2020), **PreLayer** (Hu et al., 2022), IA3 (Liu et al., 2022), and **FPFT**. **Optimizers** include **AdamW** (Loshchilov and Hutter, 2017), **SGDM** (Qian, 1999), **SGD**, **Adafactor** (Shazeer and Stern, 2018), **Adagrad** (Duchi et al., 2010). Some baselines might only appear in certain experiments.

D Datasets

We conduct experiments on the following datasets: **GLUE** (Wang et al., 2018) (SST-2 (Socher et al., 2013), CoLA (Warstadt et al., 2019), MNLI (Williams et al., 2018), MRPC (Warstadt et al., 2019), QNLI (Rajpurkar et al., 2018), QQP[2], RTE and STS-B (Cer et al., 2017)); **SuperGLUE** (CB (De Marneffe et al., 2019), BoolQ (Clark et al., 2019), COPA (Roemmele et al., 2011), MultiRC (Khashabi et al., 2018), RTE, WiC (Pilehvar and Camacho-Collados, 2019), WSC (Levesque et al., 2012), ReCoRD (Zhang et al., 2018)), **SQuAD** (Rajpurkar et al., 2016), **E2E** (Novikova et al., 2017), **DROP** (Dua et al., 2019), **ViGGO** (Juraska et al., 2019), **SQL Generation** (Yu et al., 2018; Zhong et al., 2017) and **GSM8K** (Cobbe et al., 2021).

E Difference from Splitting Optimization

The purpose of splitting optimization is to serve parallel computing. For example, matrix $C = A \cdot B$, matrix A can be divided into A$_1$ and A$_2$ by row, then $C = [A_1 \cdot B; A_2 \cdot B]$. We can put $A_1 \cdot B$ and $A_2 \cdot B$ on different devices and calculate them in parallel. The purpose of HiFT is full-parameter model fine-tuning on low-resource devices. HiFT only updates a subset of parameters at each training

[2]https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs

step. Reduce the number of trainable parameters in each step through layer-by-layer asynchronous updates, thereby reducing the memory usage of fine-tuning models. Both the algorithm process and the purpose of the algorithm are different.

Besides, the theory behind splitting optimization is the matrix block principle. This principle states that a large matrix can be divided into smaller submatrices or blocks. These blocks can then be manipulated independently. The result of each block is a subset of the original matrix multiplication result. Megatron-LM applies the splitting optimization principle to conduct large-scale parallel training of language models. However, HiFT does not rely on the matrix block principle. HiFT's updates are independent at each step, not a subset of standard fine-tuning, and is a new approach independent of standard fine-tuning. The relationship between HiFT's update process and standard fine-tuning cannot be described using splitting optimization.

F Implementation Details

The performance results of the experiment are based on training with the AdamW optimizer. For RoBERTa$_{\text{base}}$ and RoBERTa$_{\text{large}}$ models, we follow Chen et al. (2023) for the hyperparameter setting of no-prompt fine-tuning such as batch size and learning rate. For GPT-2$_{\text{medium}}$and GPT-2$_{\text{large}}$, we follow Hu et al. (2022) for the hyperparameter setting for no-prompt fine-tuning such as batch size and learning rate. For RoBERTa$_{\text{large}}$ model, we follow Malladi et al. (2023) for the hyperparameter setting of prompt fine-tuning such as prompt template, batch size and learning rate. The specific model layering principle is that all embedding layers are treated as a single layer including position coding, all head layer parameters are treated as a single layer, and the remaining layers are divided according to the construction structure of the model. For example, RoBERTa$_{\text{base}}$ has 12 hidden layers, thus are divided into 12 layer units. Then we group them according to the divided layers. Table 6 reports hyperparameter used for HiFT. For instruction fine-tuning, we fine-tune these languages models on Alpaca dataset (Taori et al., 2023). Alpaca contains 51K instruction-following demonstrations generated from text-davinci-003 (GPT-3.5). For evaluation, we use the fine-tuned models to generate responses for the pre-defined questions, which are from the MT-bench (Zheng et al., 2024). GPT-4 takes these answers as input and evaluates them with scores within 10. Repository FastChat[3] provides the detailed evaluation process.

G More Experiment Results

G.1 Proportion of Parameters

Figure 6 (a, b, c, d) shows the percentage of memory used by the parameters of each part when fine-tuning LLaMA-2 (7B) under standard FPFT and HiFT with the AdamW optimizer. Figure 6 (e) repotrs the changes in the amount of peak fine-tuning parameters under HiFT at different model sizes.

G.2 Mixing Precision

We observe an interesting phenomenon when fine-tuning the GPT-Neo (2.7B) (Table 11 in Appendix G) and LLaMA-2 (7B) (Table 12) using mixed precision, the memory usage is higher than FPFT. We find that when using mixed precision fine-tuning, both single-precision and half-precision parameters of the model exist simultaneously. Therefore, the model parameters use more memory in mixed precision than in standard FPFT. Mixed precision mainly focuses on reducing the memory usage of activation states (i.e., residual states). When the model's own parameter size is large, the memory increase of the model parameters may be greater than the memory reduction of mixed precision (when the batch size is not large enough). Therefore, it may appear that the memory usage of mixed precision is greater than standard FPFT. Due to the large number of parameters of LLMs (large language models), it is difficult to use larger batch sizes, so it is difficult to bring out the advantages of mixed precision in the context of large models. HiFT is an optional, more efficient solution that maintains single-precision full-parameter fine-tuning while greatly reducing memory usage. We would like to emphasize that the current mixed precision does not support hierarchical operations, so it cannot take advantage of HiFT.

To fully exploit the advantages of HiFT, we have adapted mixed precision to HiFT. That is, each step only moves the single-precision weight corresponding to the parameter that needs to be updated to the GPU (Mixed precision makes a single-precision backup of the weights of the half-precision model.). Table 12 reports the memory profiling for LLaMA2-7B using adapted mixed precision. When using the AdamW optimizer, the adapted mixed precision for HiFT saves approximately 76.65% of GPU

[3]https://github.com/lm-sys/FastChat

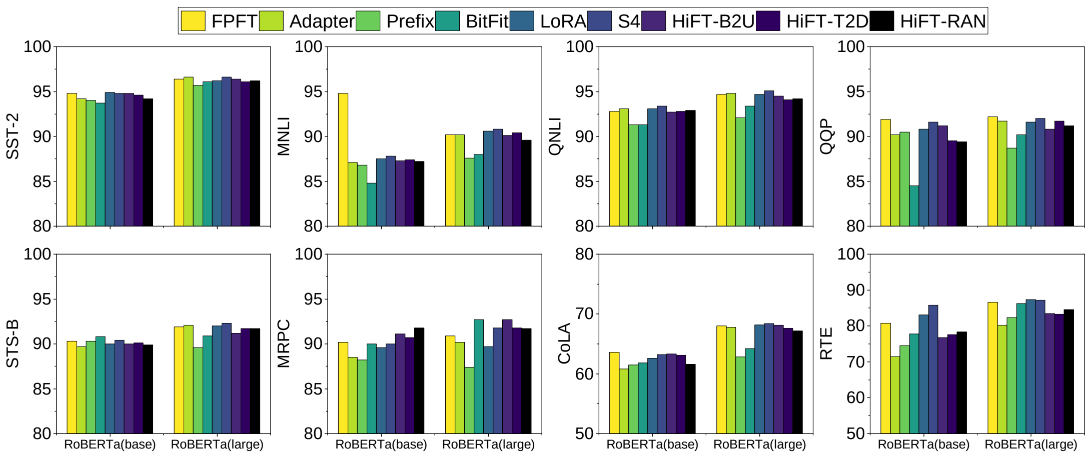

Figure 5: RoBERTa results on different fine-tuning strategies. We report accuracy metrics for the SST-2, QNLI, QQP, MRPC and RTE, mean accuracy for MNLI, spearman coefficient for STS-B and matthews correlation coefficient for CoLA. The m of HiFT is set to 1. B2U, T2D and RAN are bottom2up, top2down and random strategies.

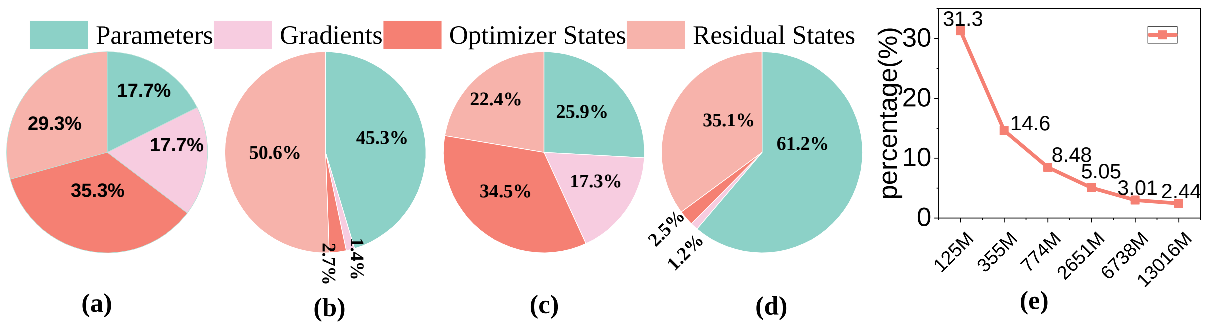

Figure 6: (a), (b), (c) and (d) represent the proportion of parameters occupied by different parts when fine-tuning LLaMA-2 (7B). The sequence length and batch size are set to 512 and 6. (a): 32-bit precision FPFT; (b): 32-bit precision HiFT; (c) mixed precision FPFT; (d): mixed precision HiFT. Fine-tuning uses the AdamW optimizer. The m is set to 1 for HiFT. (e) represents the change in the proportion of the peak trainable parameters to the total model parameters during the HiFT training under different size models.

memory. When the batch size is 1, fine-tuning the LLaMA-7B model on the E2E data set requires approximately **16.87G** of GPU memory, and fine-tuning the LLaMA-13B model requires approximately 31G of memory. This means that HiFT supports FPFT of a 7B model on a device with 24G GPU memory.

G.3 Prompts

Tables 13 and 14 gives detailed prompts of different datasets.

Experiment	Hyperparameters	Values
RoBERTa-base	Total Batch size	64
	Learning rate	$\{1e{-}5, 2e{-}5, 3e{-}5\}$
	warmup	{0.0, 0.02, 0.06}
	Device	8*GTX 1080Ti (11G)
	Weight Decay	0
RoBERTa-large	Total Batch size	32
	Learning rate	$\{1e{-}5, 2e{-}5, 3e{-}5\}$
	warmup	{0.0, 0.02, 0.06}
	Device	8*GTX 1080Ti (11G)
	Weight Decay	0
GPT-2 (M)	Batch size	32
	Learning rate	$\{5e{-}5\}$
	warmup	{0.0}
	Device	RTX A6000 (48G)
	Temperature	0.75
	Beam size	16
	repetition penalty	4
	length penalty	0.9
GPT-2 (L)	Batch size	32
	Learning rate	$\{5e{-}5\}$
	warmup	{0.0}
	Device	RTX A6000 (48G)
	Temperature	0.75
	Beam size	16
	repetition penalty	4
	length penalty	0.9
RoBERTa-large	Batch size ($k = 16$)	$\{2, 4, 8\}$
	Batch size ($k = 512$)	$\{8, 16, 32\}$
	Learning Rates	$\{1e{-}5, 3e{-}5, 5e{-}5, 8e{-}5\}$
	Device	8*GTX 1080Ti (11G)
	Weight Decay	0
OPT-13B	Batch size	$\{2, 4, 8\}$
	Learning Rates	$\{1e{-}5, 2e{-}5, 5e{-}5, 8e{-}5\}$
	Device	A100 (80G)
	Weight Decay	0
Mistral-7B	Batch size	$\{2, 4, 8\}$
	Learning Rates	$\{1e{-}5, 2e{-}5, 5e{-}5\}$
	Device	A100 (80G)
	Weight Decay	0
TinyLLaMA	Batch size	$\{2, 4, 8\}$
	Learning Rates	$\{2e{-}5, 5e{-}5, 8e{-}5\}$
	Device	A100 (80G)
	Weight Decay	0
LLaMA2-7B	Batch size	$\{2, 4, 8\}$
	Learning Rates	$\{1e{-}5, 2e{-}5, 5e{-}5, 8e{-}5\}$
	Device	A100 (80G)
	Weight Decay	0
LLaMA2-13B	Batch size	$\{2, 4, 8\}$
	Learning Rates	$\{1e{-}5, 2e{-}5, 5e{-}5, 8e{-}5\}$
	Device	A100 (80G)
	Weight Decay	0

Table 6: The hyperparameter grids used for HiFT experiments.

Model	Method	Writing	Roleplay	Reasoning	Math	Extraction	Stem	Humanities	AVG
TinyLLaMA	Vanilla	1.06	2.25	1.17	1.05	1.10	1.50	1.00	1.30
	FPFT	3.30	3.85	1.40	1.35	**1.77**	2.70	2.35	2.39
	LoRA	2.80	4.00	1.27	**1.45**	1.05	1.55	2.20	2.05
	Prefix	2.75	3.50	1.20	1.35	1.10	1.45	1.35	1.81
	HiFT	**3.50**	**4.45**	**2.50**	1.40	1.70	**3.15**	**3.20**	**2.84**
Mistral-7B	Vanilla	5.30	3.25	4.55	2.60	**6.55**	**6.20**	4.60	4.72
	FPFT	5.55	4.50	5.40	**3.35**	5.80	4.65	5.50	4.96
	LoRA	5.45	4.45	4.60	3.25	5.55	5.50	4.35	4.74
	Prefix	5.35	4.30	4.50	3.25	5.45	5.55	4.40	4.69
	HiFT	**6.45**	**5.40**	**5.45**	3.05	5.85	6.05	**6.15**	**5.49**
LLaMA2-7B	Vanilla	3.05	4.45	2.90	1.75	3.35	5.25	4.50	3.61
	FPFT	5.50	6.55	3.65	2.10	**4.75**	6.55	7.65	5.25
	LoRA	6.20	5.60	**4.15**	1.55	4.20	6.30	6.15	4.88
	Prefix	6.35	5.45	3.70	1.40	4.50	6.15	6.20	4.82
	HiFT	**6.70**	**7.15**	3.55	**2.20**	4.55	**6.85**	**7.85**	**5.55**

Table 7: Performance comparison of different fine-tuning methods on the MT-Bench. The rank of LoRA is 64, and the number of virtual words of prefix is 128.

Optimizer	#Dtype	#FType	#Trainable Parameters	#Para(MB)	#Gra(MB)	#Sta(MB)	#PGS(GB)	Residual States(GB)	Total(GB)
AdamW	fp32	FPFT	124.65M	475.49	475.49	950.98	1.86	5.02	6.88
		HiFT	39.00M	475.49	148.77	297.54	0.90	3.61	4.52
	mixed	FPFT	124.65M	713.25	475.49	950.98	2.09	3.58	5.67
		HiFT	39.00M	713.25	148.77	297.54	1.13	2.58	3.71
	Mixed^{Hi}	HiFT	39.00M	386.52	148.77	297.54	0.81	1.81	**2.62**
SGDM	fp32	FPFT	124.65M	475.49	475.49	475.49	1.39	5.00	6.39
		HiFT	39.00M	475.49	148.77	148.77	0.75	3.76	4.52
	mixed	FPFT	124.65M	713.25	475.49	475.49	1.63	3.57	5.20
		HiFT	39.00M	713.25	148.77	148.77	0.99	2.70	3.69
	Mixed^{Hi}	HiFT	39.00M	386.52	148.77	148.77	0.67	1.93	**2.60**
SGD	fp32	FPFT	124.65M	475.49	475.49	0.00	0.93	4.97	5.90
		HiFT	39.00M	475.49	148.77	0.00	0.61	3.91	4.52
	mixed	FPFT	124.65M	713.25	475.49	0.00	1.16	3.57	4.73
		HiFT	39.00M	713.25	148.77	0.00	0.84	2.87	3.71
	Mixed^{Hi}	HiFT	39.00M	386.52	148.77	0.00	0.52	2.06	**2.58**
Adafactor	fp32	FPFT	124.65M	475.49	475.49	0.98	0.93	4.98	5.91
		HiFT	39.00M	475.49	148.77	0.19	0.61	3.91	4.52
	mixed	FPFT	124.65M	713.25	475.49	0.98	1.16	3.57	4.73
		HiFT	39.00M	713.25	148.77	0.19	0.84	2.87	3.71
	Mixed^{Hi}	HiFT	39.00M	386.52	148.77	0.19	0.52	2.09	**2.61**
Adagrad	fp32	FPFT	124.65M	475.49	475.49	475.49	1.39	5.00	6.39
		HiFT	39.00M	475.49	148.77	148.77	0.75	3.76	4.52
	mixed	FPFT	124.65M	713.25	475.49	475.49	1.63	3.57	5.20
		HiFT	39.00M	713.25	148.77	148.77	0.99	2.70	3.69
	Mixed^{Hi}	HiFT	39.00M	386.52	148.77	148.77	0.67	1.96	**2.62**

Table 8: The GPU memory usage of fine-tuning RoBERTa$_{\text{base}}$ on the CoLA dataset. The sequence length and batch size are set to 512 and 8, respectively. **#Dtype** represents the data type used for training, where **FP32** represents fully parameter fine-tuning the model with 32-bit precision, and **mixed** represents fine-tuning with mixed precision. **#Trainable parameters** represents the maximum number of trainable parameters that appear in a single step during the fine-tuning process. **#Para** represents the memory occupied by the model parameters, **#Gra** represents the memory occupied by the gradient, and **#Sta** represents the memory occupied by the optimizer state. **#PGS** represents the sum of memory occupied by model parameters (i.e.,**#Para**), gradients (i.e.,**#Gra**) and optimizer state (i.e.,**#Sta**). **Residual states** mainly includes activation, temporary buffers and unusable fragmented memory. **Total** represents the total memory used during fine-tuning. The parameter m of HiFT is set to 1.

Optimizer	#Dtype	#FType	#Trainable Parameter	#Para(MB)	#Gra(MB)	#Sta(MB)	#PGS(GB)	Residual States(GB)	Total(GB)
AdamW	fp32	FPFT	355.36M	1355.60	1355.60	2711.20	5.30	13.08	18.38
		HiFT	52.00M	1355.60	198.38	396.73	1.90	9.97	11.88
	mixed	FPFT	355.36M	2033.40	1355.60	2711.20	5.96	9.30	15.25
		HiFT	52.00M	2033.40	198.38	396.73	2.57	7.17	9.74
	Mixed^{Hi}	HiFT	52.00M	876.18	198.38	396.73	1.44	5.18	**6.62**
SGDM	fp32	FPFT	355.36M	1355.60	1355.60	1355.60	3.97	13.08	17.05
		HiFT	52.00M	1355.60	198.38	198.38	1.71	10.20	11.91
	mixed	FPFT	355.36M	2033.40	1355.60	1355.60	4.63	9.30	13.93
		HiFT	52.00M	2033.40	198.38	198.38	2.37	7.37	9.74
	Mixed^{Hi}	HiFT	52.00M	876.18	198.38	198.38	1.24	5.38	**6.62**
SGD	fp32	FPFT	355.36M	1355.60	1355.60	0.00	2.65	13.08	15.73
		HiFT	52.00M	1355.60	198.38	0.00	1.52	10.36	11.88
	mixed	FPFT	355.36M	2033.40	1355.60	0.00	3.31	9.30	12.60
		HiFT	52.00M	2033.40	198.38	0.00	2.18	7.50	9.68
	Mixed^{Hi}	HiFT	52.00M	876.18	198.38	0.00	1.05	5.50	**6.55**
Adafactor	fp32	FPFT	355.36M	1355.60	1355.60	3.14	2.65	13.08	15.73
		HiFT	52.00M	1355.60	198.38	0.21	1.52	10.36	11.88
	mixed	FPFT	355.36M	2033.40	1355.60	3.14	3.31	9.30	12.61
		HiFT	52.00M	2033.40	198.38	0.21	2.18	7.51	9.69
	Mixed^{Hi}	HiFT	52.00M	876.18	198.38	0.21	1.05	5.50	**6.55**
Adagrad	fp32	FPFT	355.36M	1355.60	1355.60	1355.60	3.97	13.08	17.05
		HiFT	52.00M	1355.60	198.38	198.38	1.71	10.20	11.91
	mixed	FPFT	355.36M	2033.40	1355.60	1355.60	4.63	9.30	13.93
		HiFT	52.00M	2033.40	198.38	198.38	2.37	7.37	9.74
	Mixed^{Hi}	HiFT	52.00M	876.18	198.38	198.38	1.24	5.38	**6.62**

Table 9: The GPU memory usage of fine-tuning $\text{RoBERTa}_{\text{large}}$ on the CoLA dataset. The sequence length and batch size are set to 512 and 8, respectively.

			#Trainable					Residual	
Optimizer	#Dtype	#FType	Parameters	#Para(MB)	#Gra(MB)	#Sta(MB)	#PGS(GB)	States(GB)	Total(GB)
AdamW	fp32	FPFT	774.03M	2952.69	2952.69	5905.39	11.53	37.26	48.79
		HiFT	65.64M	2952.69	250.40	500.79	3.62	31.73	35.35
	mixed	FPFT	774.03M	4429.05	2952.69	5905.39	12.98	28.13	41.11
		HiFT	65.64M	4429.05	250.40	500.79	5.06	24.97	30.03
	Mixed^{Hi}	HiFT	65.64M	1726.75	250.40	500.79	2.42	16.38	**18.80**
SGDM	fp32	FPFT	774.03M	2952.69	2952.69	2952.69	8.65	37.26	45.91
		HiFT	65.64M	2952.69	250.40	250.40	3.37	31.98	35.35
	mixed	FPFT	774.03M	4429.05	2952.69	2952.69	10.09	28.14	38.23
		HiFT	65.64M	4429.05	250.40	250.40	4.81	25.22	30.03
	Mixed^{Hi}	HiFT	65.64M	1726.75	250.40	250.40	2.18	16.62	**18.80**
SGD	fp32	FPFT	774.03M	2952.69	2952.69	0.00	5.77	37.25	43.02
		HiFT	65.64M	2952.69	250.40	0.00	3.13	32.22	35.35
	mixed	FPFT	774.03M	4429.05	2952.69	0.00	7.21	28.12	35.33
		HiFT	65.64M	4429.05	250.40	0.00	4.57	25.46	30.03
	Mixed^{Hi}	HiFT	65.64M	1726.75	250.40	0.00	1.93	16.32	**18.25**
Adafactor	fp32	FPFT	774.03M	2952.69	2952.69	5.31	5.77	37.26	43.03
		HiFT	65.64M	2952.69	250.40	0.21	3.13	32.22	35.35
	mixed	FPFT	774.03M	4429.05	2952.69	5.31	7.21	28.12	35.33
		HiFT	65.64M	4429.05	250.40	0.21	4.57	25.46	30.03
	Mixed^{Hi}	HiFT	65.64M	1726.75	250.40	0.21	1.93	16.37	**18.30**
Adagrad	fp32	FPFT	774.03M	2952.69	2952.69	2952.69	8.65	37.26	45.91
		HiFT	65.64M	2952.69	250.40	250.40	3.37	31.98	35.35
	mixed	FPFT	774.03M	4429.05	2952.69	2952.69	10.09	28.13	38.22
		HiFT	65.64M	4429.05	250.40	250.40	4.81	25.22	30.03
	Mixed^{Hi}	HiFT	65.64M	1726.75	250.40	250.40	2.18	16.62	**18.80**

Table 10: The GPU memory usage of fine-tuning GPT-2$_{\text{large}}$ on the E2E dataset. The sequence length and batch size are set to 512 and 8, respectively.

			#Trainable					Residual	
Optimizer	#Dtype	#FType	Parameters	#Para(MB)	#Gra(MB)	#Sta(MB)	#PGS(GB)	States(GB)	Total(GB)
AdamW	fp32	FPFT	2651.31M	10113.95	10113.95	20227.89	39.51	22.69	62.20
		HiFT	133.9M	10113.95	510.79	1021.58	11.37	16.96	**28.33**
	mixed	FPFT	2651.31M	15170.93	10113.95	20227.89	44.45	19.56	64.01
		HiFT	133.9M	15170.93	510.79	1021.58	16.31	15.92	32.23
	Mixed^{Hi}	HiFT	133.9M	5567.77	510.79	1021.58	6.93	24.63	31.56
SGDM	fp32	FPFT	2651.31M	10113.95	10113.95	10113.95	29.63	22.69	52.32
		HiFT	133.9M	10113.95	510.79	510.79	10.87	17.46	**28.33**
	mixed	FPFT	2651.31M	15170.93	10113.95	10113.95	34.57	19.56	54.13
		HiFT	133.9M	15170.93	510.79	510.79	15.81	16.33	32.14
	Mixed^{Hi}	HiFT	133.9M	5567.77	510.79	510.79	6.43	25.13	31.56
SGD	fp32	FPFT	2651.31M	10113.95	10113.95	0.00	19.75	22.69	42.44
		HiFT	133.9M	10113.95	510.79	0.00	10.38	17.95	**28.33**
	mixed	FPFT	2651.31M	15170.93	10113.95	0.00	24.69	19.57	44.26
		HiFT	133.9M	15170.93	510.79	0.00	15.31	16.83	32.14
	Mixed^{Hi}	HiFT	133.9M	5567.77	510.79	0.00	5.94	25.49	31.43
Adafactor	fp32	FPFT	2651.31M	10113.95	10113.95	8.99	19.76	22.69	42.45
		HiFT	133.9M	10113.95	510.79	0.22	10.38	17.95	**28.33**
	mixed	FPFT	2651.31M	15170.93	10113.95	8.99	24.70	19.56	44.26
		HiFT	133.9M	15170.93	510.79	0.22	15.31	16.83	32.14
	Mixed^{Hi}	HiFT	133.9M	5567.77	510.79	0.22	5.94	25.49	31.43
Adagrad	fp32	FPFT	2651.31M	10113.95	10113.95	10113.95	29.63	22.69	52.32
		HiFT	133.9M	10113.95	510.79	510.79	10.87	17.46	**28.33**
	mixed	FPFT	2651.31M	15170.93	10113.95	10113.95	34.57	19.57	54.14
		HiFT	133.9M	15170.93	510.79	510.79	15.81	16.33	32.14
	Mixed^{Hi}	HiFT	133.9M	5567.77	510.79	510.79	6.43	25.13	31.56

Table 11: The GPU memory usage of fine-tuning GPT-Neo on the E2E dataset. The sequence length and batch size are set to 512 and 8, respectively.

			#Trainable					Residual	
Optimize	**#Dtype**	**#FType**	**Parameters**	**#Para(MB)**	**#Gra(MB)**	**#Sta(MB)**	**#PGS(GB)**	**States(GB)**	**Total(GB)**
AdamW	FP32	FPFT	6738.42M	25705.04	25705.04	51410.08	100.41	41.7	142.11
		HiFT	202.38M	25705.04	772.03	1544.06	27.36	28.04	55.41
	Mixed	FPFT	6738.42M	38557.56	25705.04	51410.08	112.96	32.54	145.50
		HiFT	202.38M	38557.56	772.03	1544.06	39.92	21.62	61.54
	Mixed^{Hi}	HiFT	202.38M	13624.53	772.03	1544.06	15.57	18.40	**33.96**
SGDM	FP32	FPFT	6738.42M	25705.04	25705.04	25705.04	75.31	41.71	117.01
		HiFT	202.38M	25705.04	772.03	772.03	26.61	28.8	55.41
	Mixed	FPFT	6738.42M	38557.56	25705.04	25705.04	87.86	32.54	120.40
		HiFT	202.38M	38557.56	772.03	772.03	39.16	22.37	61.54
	Mixed^{Hi}	HiFT	202.38M	13624.53	772.03	772.03	14.81	19.15	**33.96**
SGD	FP32	FPFT	6738.42M	25705.04	25705.04	0.00	50.21	41.72	91.93
		HiFT	202.38M	25705.04	772.03	0.00	25.86	29.55	55.41
	Mixed	FPFT	6738.42M	38557.56	25705.04	0.00	62.76	32.54	95.30
		HiFT	202.38M	38557.56	772.03	0.00	38.41	23.13	61.54
	Mixed^{Hi}	HiFT	202.38M	13624.53	772.03	0.00	14.06	19.00	**33.06**
Adafactor	FP32	FPFT	6738.42M	25705.04	25705.04	10.82	50.22	41.72	91.94
		HiFT	202.38M	25705.04	772.03	0.33	25.86	29.55	55.41
	Mixed	FPFT	6738.42M	38557.56	25705.04	10.82	62.77	32.54	95.31
		HiFT	202.38M	38557.56	772.03	0.33	38.41	23.13	61.54
	Mixed^{Hi}	HiFT	202.38M	13624.53	772.03	0.33	14.06	19.00	**33.06**
Adagrad	FP32	FPFT	6738.42M	25705.04	25705.04	25705.04	75.31	41.72	117.01
		HiFT	202.38M	25705.04	772.03	772.03	26.61	28.80	55.41
	Mixed	FPFT	6738.42M	38557.56	25705.04	25705.04	87.86	32.54	120.40
		HiFT	202.38M	38557.56	772.03	772.03	39.16	22.37	61.54
	Mixed^{Hi}	HiFT	202.38M	13624.53	772.03	772.03	14.81	19.15	**33.96**

Table 12: The GPU memory usage of fine-tuning LLaMA (7B) on the E2E dataset. The sequence length and batch size are set to 512 and 6, respectively.

Dataset	C	**Type**	**Prompt**	**Label words**
SST-2	2	sentiment cls.	<S_1> It was `[MASK]` .	{great, terrible}
SST-5	5	sentiment cls.	<S_1> It was `[MASK]` .	{great, good, okay, bad, terrible}
TREC	6	topic cls.	`[MASK]` : <S_1>	{Description, Expression, Entity, Human, Location, Number}
MNLI	3	NLI	<S_1> ? `[MASK]` , <S_2>	{Yes, Maybe, No}
SNLI	3	NLI	<S_1> ? `[MASK]` , <S_2>	{Yes, Maybe, No}
RTE	2	NLI	<S_1> ? `[MASK]` , <S_2>	{Yes, No}

Table 13: The prompts of the datasets we used in our RoBERTa-large experiments (i.e., Table 1). The prompts are adapted from (Gao et al., 2021) and include a template and a set of label words that can fill in the `[MASK]`token. <S_1> and <S_2> refer to the first and the second (if any) input sentence. C is the number of labels.

Dataset	Type	Prompt
SST-2	cls.	`<text>` It was terrible/great
RTE	cls.	`<premise>` Does this mean that "`<hypothesis>`" is true? Yes or No? Yes/No
CB	cls.	Suppose `<premise>` Can we infer that "`<hypothesis>`"? Yes, No, or Maybe? Yes/No/Maybe
BoolQ	cls.	`<passage>` `<question>`? Yes/No
WSC	cls.	`<text>` In the previous sentence, does the pronoun "`<span2>`" refer to `<span1>`? Yes or No? Yes/No

WIC	cls.	Does the word "`<word>`" have the same meaning in these two sentences? Yes, No?
		`<sent1>`
		`<sent2>`
		Yes/No

MultiRC	cls.	`<paragraph>` Question: `<question>` I found this answer "`<answer`". Is that correct? Yes or No? Yes/No
COPA	mch.	`<premise>` so/because `<candidate>`

ReCoRD	mch.	`<passage>`
		`<query>.replace("@placeholder", <candidate>)`

SQuAD	QA	Title: `<title>` Context: `<context>` Question: `<question>` Answer:
DROP	QA	Passage: `<context>` Question: `<question>` Answer:

Table 14: The prompts of the datasets we used in our OPT experiments. There are three types of tasks: classification (cls.), multiple-choice (mch.), and question answering (QA). `<text>` represents input from the dataset and Yes represents label words. For inference on multiple choice tasks, we put in different candidates in the prompt and calculate the average log-likelihood for each candidate, and choose the candidate with the highest score. For inference on QA tasks, we use greedy decoding to generate the answer. All prompts configurations are consistent with Malladi et al. (2023)