Differentially Private Bias-Term only Fine-tuning of Foundation Models

Zhiqi Bu[1] **Yu-Xiang Wang**[1,2] **Sheng Zha**[1] **George Karypis**[1]
[1] AWS AI [2] UC Santa Barbara
`zhiqibu@amazon.com` `yuxiangw@cs.ucsb.edu`
`zhasheng@amazon.com` `gkarypis@amazon.com`

Abstract

We study the problem of differentially private (DP) fine-tuning of large pre-trained models — a recent privacy-preserving approach suitable for solving downstream tasks with sensitive data. Existing work has demonstrated that high accuracy is possible under strong privacy constraint, yet requires significant computational overhead or modifications to the network architecture.

We propose differentially private bias-term fine-tuning (DP-BiTFiT), which matches the state-of-the-art accuracy for DP algorithms and the efficiency of the standard BiTFiT. DP-BiTFiT is model agnostic (not modifying the network architecture), parameter efficient (only training about 0.1% of the parameters), and computation efficient (almost removing the overhead caused by DP, in both the time and space complexity). On a wide range of tasks, DP-BiTFiT is $2 \sim 30\times$ faster and uses $2 \sim 8\times$ less memory than DP full fine-tuning, even faster than the standard full fine-tuning. This amazing efficiency enables us to conduct DP fine-tuning on language and vision tasks with long-sequence texts and high-resolution images, which were computationally difficult using existing methods.

1 Introduction

Fine-tuning from large pre-trained neural networks is one of the most critical technique in deep learning, yielding strong performance in a variety of domains Pan & Yang (2009); Kenton & Toutanova (2019); Goyal et al. (2017). Among different methods, full fine-tuning is the most prevalent one, which trains all the model parameters on the downstream tasks and achieves high accuracy within a small number of training epochs. However, full fine-tuning on large models can be burdensome in terms of the computation and the deployment, since a full copy of fine-tuned model parameters is needed for each task.

To alleviate this issue, the parameter efficient fine-tuning only trains a substantially small portion of the model parameters, in contrast to the full fine-tuning. At high level, the parameter efficient fine-tuning methods can be divided into two categories. $\langle 1 \rangle$ Model-aware methods, meaning a relatively small number of parameters are introduced into the neural network architecture and only the new parameters are optimized. Examples include LoRA Hu et al. (2021), Adapter Houlsby et al. (2019), and Compacter Mahabadi et al. (2021). $\langle 2 \rangle$ Model-agnostic methods, meaning that only a subset of existing parameters are trainable. Examples include training only the output linear layer (linear probing, Kornblith et al. (2019)), only the layer normalization layer Houlsby et al. (2019) and bias-term fine-tuning (BiTFiT; Zaken et al. (2022)). We illustrate the differences in Equation (1): $\mathbf{W}_0, \mathbf{b}_0$ are the pre-trained weights and biases, ‘ ˆ ’ indicates trainable parameters, and $\boldsymbol{\theta}$ is the additional parameters.

$$\underbrace{f(\boldsymbol{x};\mathbf{W}_0,\mathbf{b}_0)}_{\text{pre-trained model}} \quad\longrightarrow\quad \underbrace{f(\boldsymbol{x};\hat{\mathbf{W}},\hat{\mathbf{b}})}_{\text{full fine-tuning}} \quad\text{or}\quad \underbrace{f(\boldsymbol{x};\mathbf{W}_0,\mathbf{b}_0,\hat{\boldsymbol{\theta}})}_{\text{model-aware fine-tuning}} \quad\text{or}\quad \underbrace{f(\boldsymbol{x};\mathbf{W}_0,\hat{\mathbf{b}})}_{\text{bias-term fine-tuning}} \qquad (1)$$

Empirically, these parameter efficient fine-tuning methods have achieved high accuracy that is comparable to the full fine-tuning in the standard non-private setting. For instance, linear probing of ResNet He et al. (2016) and Vision Transformer (ViT, Dosovitskiy et al. (2020)) achieves 80% accuracy on the ImageNet dataset Sun et al. (2017); Kornblith et al. (2019); LoRA and BiTFiT of RoBERTa Liu et al. (2019) and BERT Kenton & Toutanova (2019) achieve about 94% on SST2, 87% on MNLI, and on average 85% across the General Language Understanding Evaluation (GLUE) datasets He et al. (2021); Hu et al. (2021). In addition, parameter efficient methods are faster than full fine-tuning and save the communication cost significantly in the distributed learning.

Parallel to these developments, the success of deep learning models relies on the availability of large datasets, which may contain sensitive information to be protected rigorously. This privacy issue is well-known for neural networks can be vulnerable to privacy attacks: membership information can be leaked from the purchase records via Google and Amazon online services Shokri et al. (2017); sensitive texts can be reconstructed by specifically designed prefix on GPT2 Carlini et al. (2021) and so can images in CIFAR10 and MNIST Haim et al. (2022). To protect against such privacy risks, the standard technique is differential privacy (DP, formally stated in Definition 2.1), which randomizes the standard optimizers by updating with the private gradient in Equation (2).

A recent line of work has extensively studied the DP fine-tuning in both computer vision and language tasks, often achieving less than 3% accuracy drop across different settings via full fine-tuning De et al. (2022); Li et al. (2021); Bu et al. (2022b,a), linear probing Mehta et al. (2022), LoRA, Adapter, or Compacter Yu et al. (2021a). In fact, fine-tuning or pre-training from large dataset is considered necessary in the DP deep learning literature. As a matter of fact, full fine-tuning DP-GPT2 only achieves 24.2 BLEU score ($\epsilon = 8$) on E2E dataset if randomly initialized Li et al. (2021), in starking contrast to 63.2 BLEU if pre-trained; similarly, state-of-the-art (SOTA) DP accuracy on ImageNet is 48% ($\epsilon = 10$) without pre-training Kurakin et al. (2022) but 86.7% accuracy if pre-trained De et al. (2022). Specifically, parameter efficient DP fine-tuning has empirically demonstrated strong accuracy (see our Table 3) with $3 \sim 4\times$ memory saving and $2 \sim 3\times$ speedup compared to DP full fine-tuning by Opacus (c.f. Figure 3 and (Yu et al., 2021a, Table 3)). Although previous works have shed light on various DP fine-tuning methods, we are the first to study DP-BiTFiT specifically and to show two distinctive advantages of it.

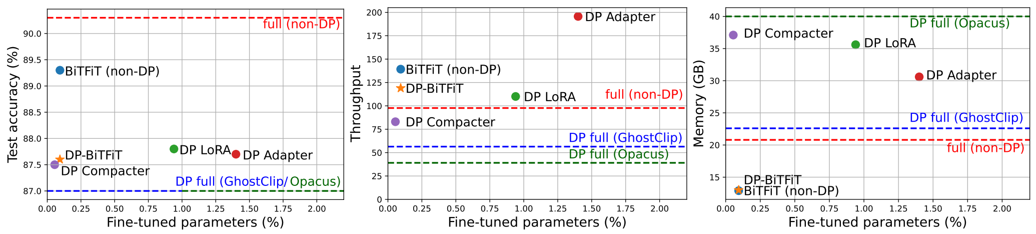

Figure 1: Performance of different fine-tuning methods on MNLI dataset with RoBERTa-large.

Firstly, DP-BiTFiT is model-agnostic and remains its parameter efficiency around 0.1% across models by Table 1. While linear probing is also model-agnostic, the parameter efficiency can be as high as 8% in ResNet50. Other methods like LoRA, Adapter and Compacter are architecture-dependent and possibly parameter inefficient, making them difficult to directly apply on arbitrary neural networks: LoRA and Adapter may need to train more than 12% on BART-large Lewis et al. (2020) to achieve high accuracy by (He et al., 2021, Figure 1& 4).

Secondly, DP-BiTFiT is computationally efficient, almost as much as the standard BiTFiT and significantly more efficient than DP full fine-tuning, particularly with large models and high-dimensional input data. For examples of DP full fine-tuning, Li et al. (2021) have reported $2 \sim 4\times$ slowdown on large language models for four advanced private codebases and up to $5\times$ memory overhead, compared to the standard fine-tuning; even on small networks, 11 codebases across Tensorflow, JAX, and Pytorch have demonstrated $0.2 \sim 5\times$ slowdown and $3 \sim 100\times$ reduction in maximum batch size in Subramani et al. (2021). See more discussion in Section 3.3.

1.1 Contributions

1. Algorithmically, we propose the Differentially Private Bias-Term Fine-Tuning (DP-BiTFiT) in Algorithm 1 that is highly accurate under DP constraint, on par with SOTA in Section 4.

Specially, we propose a two-phase training to close the accuracy gap between DP-BiTFiT and DP full-finetuning.

2. DP-BiTFiT is model-agnostic and only optimizes 0.1% of the model parameters on BERT, RoBERTa, GPT2, ViT, ResNet, and so on (see Table 1). Thus DP-BiTFiT is one of the most *parameter efficient* fine-tuning methods among DP LoRA, Adapter, linear probing, etc.
3. We open-source a *computationally efficient* implementation of DP-BiTFiT, whose time and space complexity is almost the same as the standard non-DP BiTFiT, while being faster than non-DP full fine-tuning and other DP fine-tuning (see Figure 1). This advantage is analyzed in Table 2, and demonstrated via the substantial speedup and memory-saving in Figure 3 and Figure 4.
4. DP-BiTFiT is the unique DP algorithm that has a *computation overhead independent of the feature dimension* T[1]. This is due to the activation-free forward pass that only happens in the no-weight training[2]. Therefore, DP-BiTFiT enjoys a special advantage to work efficiently on long-sequence texts and high-resolution images (see Figure 3).

Algorithm 1 Bias-Term Fine-Tuning (BiTFiT) v.s. DP-BiTFiT

Parameters: l-th layer's bias $\mathbf{b}_l$, subsampling probability p, number of iterations T, number of layers L, noise scale σ, clipping threshold R, clipping factor $C_i = 1$ (no clipping).

```
1:  for iteration $t = 1, \cdots, T$ do
2:      Subsample a batch $B_t \subseteq \{1, \ldots, n\}$ from training set with probability $p$
3:      for layer $l \in L, L-1, \cdots, 1$ do
4:          Get output gradient $\frac{\partial \mathcal{L}}{\partial \boldsymbol{s}_l}$
5:          Compute per-example gradient and its norm: $\frac{\partial \mathcal{L}_i}{\partial \mathbf{b}_l} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{s}_{l,i}}^\top \mathbf{1} \implies \|\frac{\partial \mathcal{L}_i}{\partial \mathbf{b}_l}\|_F^2$
6:      Aggregate gradient norms across all layers: $\|\frac{\partial \mathcal{L}_i}{\partial \mathbf{b}}\|_F^2 = \sum_l \|\frac{\partial \mathcal{L}_i}{\partial \mathbf{b}_l}\|_F^2$
7:      Compute clipping factor: $C_i = C(\|\frac{\partial \mathcal{L}_i}{\partial \mathbf{b}}\|_F; R)$
8:      Compute sum of clipped gradients $\mathbf{G} = \sum_i C_i \frac{\partial \mathcal{L}_i}{\partial \mathbf{b}}$
9:      Add Gaussian noise $\mathbf{G} = \mathbf{G} + \sigma R \cdot \mathcal{N}(0, \mathbf{I})$
10:     Descend on bias terms with the gradient $\mathbf{G}$ by SGD/Adam/...
```

2 Preliminaries

2.1 Deep learning with differential privacy

We recall the classic (ϵ, δ)-DP, under which we learn deep neural networks.

Definition 2.1 (Dwork et al. (2006))**.** A randomized algorithm M is (ε, δ)-differentially private if, for any two neighboring datasets S, S' that differ by one datapoint and for any event E, we have $\mathbb{P}[M(S) \in E] \leqslant \mathrm{e}^{\varepsilon} \mathbb{P}[M(S') \in E] + \delta$.

In deep learning, DP can be achieved through applying an off-the-shelf optimizer (SGD or Adam) with a privately released stochastic gradient in place of the regular $\sum_i \boldsymbol{g}_i$. The private stochastic gradient is computed by first getting a minibatch $\mathcal{I}$ via Poisson sampling, then compute

$$\text{Private gradient} \quad \sum\nolimits_{i \in \mathcal{I}} \boldsymbol{g}_i \cdot C(\|\boldsymbol{g}_i\|; R) + \sigma R \cdot \mathcal{N}(0, \mathbf{I}), \tag{2}$$

where C is any function[3] $\mathbb{R}^+ \to \mathbb{R}$ subject to $C(x) \leq R/x$, $\boldsymbol{g}_i$ is the i-th per-sample gradient, R is the clipping threshold, and σ is the noise multiplier. The private gradient is guaranteed to be DP

[1]As summarized in Table 2 and Table 5, the computation overhead to get the per-sample weight gradient norm is linear (by instantiating per-sample gradints) or quadratic in T (if using the ghost norm trick Goodfellow (2015); Li et al. (2021)), for DP full and parameter efficient fine-tuning.

[2]We distinguish the weight training and bias training in Section 2.2 using the chain rules. Note that activation-free means memory-saving, which is not leveraged by DP full, LoRA, Adapter, Compacter, etc.

[3]Examples of gradient clipping include but not limited to Abadi's clipping $\min(R/\|\boldsymbol{g}_i\|, 1)$ Abadi et al. (2016) and automatic clipping (AUTO-S) $R/(\|\boldsymbol{g}_i\| + 0.01)$ Bu et al. (2022b); Yang et al. (2022).

through the *sampled-Gaussian mechanism* and the associated tight privacy accounting to compose over the iterations (see, e.g., Abadi et al., 2016; Wang et al., 2019; Mironov et al., 2019; Koskela et al., 2020; Bu et al., 2020; Gopi et al., 2021, and the references therein.).

2.2 Backward propagation

We briefly introduce the back-propagation, which reveals a simple yet important difference between the gradients of weights and those of biases. We consider a linear layer, indexed as the l-th layer, with weight $\mathbf{W}_l \in \mathbb{R}^{d\times p}$ and bias as $\mathbf{b}_l \in \mathbb{R}^p$. We leave the derivation of other layers such as normalization and convolution in Appendix A. We denote the mini-batched input of this layer as $\boldsymbol{a}_l \in \mathbb{R}^{B\times T\times d}$ and the immediate output as $\boldsymbol{s}_l \in \mathbb{R}^{B\times T\times p}$, where B is the batch size and T is the feature dimension[4]: $\boldsymbol{a}_{l+1} = \phi(\boldsymbol{s}_l), \boldsymbol{s}_l = \boldsymbol{a}_l\mathbf{W}_l + \mathbf{b}_l$. Here ϕ is any non-parametric inter-layer operation, e.g. the non-linear activation (like ReLU), pooling, padding, and so on.

We write $\mathcal{L} = \sum_{i=1}^n \mathcal{L}_i$ as the total loss and $\mathcal{L}_i$ as the per-sample loss of the i-th sample. During a standard back-propagation of L layers, the chain rule keeps track of the *output gradient* at each layer in a just-in-time fashion:

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{s}_l} = \frac{\partial\mathcal{L}}{\partial\boldsymbol{a}_L}\circ\frac{\partial\boldsymbol{a}_L}{\partial\boldsymbol{s}_{L-1}}\cdot\frac{\partial\boldsymbol{s}_{L-1}}{\partial\boldsymbol{a}_{L-1}}\circ\cdots\frac{\partial\boldsymbol{a}_{l+1}}{\partial\boldsymbol{s}_l} = \frac{\partial\mathcal{L}}{\partial\boldsymbol{s}_{l+1}}\mathbf{W}_{l+1}\circ\phi'(\boldsymbol{s}_l). \tag{3}$$

This output gradient $\frac{\partial\mathcal{L}}{\partial\boldsymbol{s}_l}$ is used to compute per-sample gradient of weights and biases,

$$\frac{\partial\mathcal{L}_i}{\partial\mathbf{W}_l}^\top = \sum_j \frac{\partial\mathcal{L}_i}{\partial\boldsymbol{s}_{l,j}}^\top\frac{\partial\boldsymbol{s}_{l,j}}{\partial\mathbf{W}_l} = \frac{\partial\mathcal{L}}{\partial\boldsymbol{s}_{l,i}}^\top\boldsymbol{a}_{l,i}, \quad \frac{\partial\mathcal{L}_i}{\partial\mathbf{b}_l}^\top = \sum_j\frac{\partial\mathcal{L}_i}{\partial\boldsymbol{s}_{l,j}}^\top\frac{\partial\boldsymbol{s}_{l,j}}{\partial\mathbf{b}_l} = \frac{\partial\mathcal{L}}{\partial\boldsymbol{s}_{l,i}}^\top\mathbf{1}. \tag{4}$$

Notably, the weight gradient needs the activation tensor $\boldsymbol{a}_l$ to compute an expensive $O(BTpd)$ tensor multiplication. Memory-wise, $\{\boldsymbol{a}_l\}_l$ across all layers is very costly to store (taking more than 95% memory across VGG, ResNet, DenseNet, RoBERTa, etc. by (Jain et al., 2020, Figure 3)). In sharp contrast, the computation of bias gradient does not need $\boldsymbol{a}_l$, and the multiplication with $\mathbf{1}$ in Equation (4) is actually a cheap $O(BTp)$ summation on $\frac{\partial\mathcal{L}}{\partial\boldsymbol{s}_l}: B\times T\times p\rightarrow B\times p$.

3 Differentially private Bias-Term Fine-Tuning

We propose DP-BiTFiT, to privately train only the bias terms in a neural network by combining Equation (4) and Equation (2). We use shaded lines to represent the additional DP operations in Algorithm 1, and add DP-related variables and operations in red in the computation graph by Figure 2. We provide the implementation details of DP-BiTFiT in Appendix B.

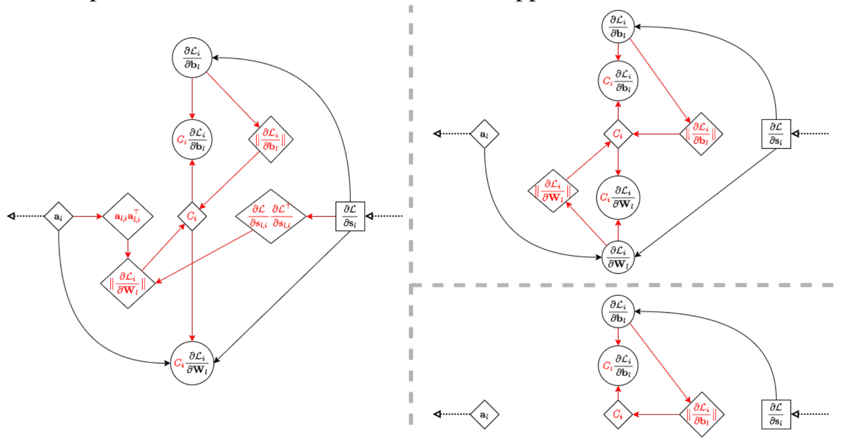

Figure 2: Back-propagation for DP (red&black) and non-DP (black) algorithms. Left: full fine-tuning with GhostClip (ghost clipping; Goodfellow (2015); Li et al. (2021); Bu et al. (2022a)). Upper right: full fine-tuning with Opacus Yousefpour et al. (2021). Lower right: BiTFiT.

[4]In sequential data such as text, T is the sequence length; in vision data, T is the product of input dimensions (e.g. for images, T is the product of height and width). We refer to a high-dimensional input when T is large.

3.1 Parameter efficiency

DP-BiTFiT enjoys exactly the same parameter efficiency as the standard BiTFiT, training merely about 0.1% of the total parameters in large models. We demonstrate that DP-BiTFiT is one of the most parameter-efficient fine-tuning through a list of models in Table 1, extended in Table 9.

An advantage of this parameter efficiency is reflected in the computation efficiency, given that most parameters do not require gradients to be computed: we show in Table 2 and Section 3.3 that DP-BiTFiT is much more efficient than full fine-tuning (DP and even non-DP). Additionally, the parameter efficiency also translates to the communication efficiency in the distributed learning. For example, the 64-bit communication cost of DP full fine-tuning is $64MD$ where M is the number of worker and D is the total number of parameters, which can be improved to $0.064MD$ by DP-BiTFiT.

Dataset	Model	# of params	% of params
ImageNet	VGG16	138M	0.009
	ResNet18	11.7M	0.043
	ResNet50	25.6M	0.113
	ViT-small-patch16	21.7M	0.238
	ViT-base-patch16	85.8M	0.120
	ViT-large-patch16	303M	0.090
E2E	GPT2-small	124M	0.082
	GPT2-medium	355M	0.076
	GPT2-large	774M	0.066
GLUE	RoBERTa-base	125M	0.083
	RoBERTa-large	355M	0.077

Table 1: Parameter efficiency of (DP) BiTFiT.

3.2 Complexity of weight and bias training

We present in Table 2 the complexity of DP training on weights and biases, for one layer mapping $B \times T_l \times d_l$ to $B \times T_l \times p_l$. To elaborate on Footnote 4, for text data, T_l is the sequence length, d_l is input dimension, and p_l is output dimension; for image data and specially in a convolution layer, T_l is height times width, d_l is the input channels times kernel sizes, p_l is the output channels (c.f. (Bu et al., 2022a, Section 2.3)). Notice that the total complexity of training a network is summed across all layers, e.g. the time complexity of standard full training is $6B\sum_l T_l p_l d_l$, DP full fine-tuning is over $8B\sum_l T_l p_l d_l$, and DP-BiTFiT is about $4B\sum_l T_l p_l d_l$. Therefore, our complexity analysis indicates that DP-BiTFiT is $6/4 = 1.5\times$ faster than non-private full fine-tuning and over $8/4 = 2\times$ faster than DP full fine-tuning.

	forward &output grad	weight training				bias training	
		non-DP	Opacus	GhostClip	MixGhostClip	non-DP	DP (ours)
Time complexity	$4BTpd$	$2BTpd$	$+2BTpd$	$+2BTpd$ $+2BT^2(p+d)$	$+2BTpd$ $+\langle 2BT^2(p+d), 2BTpd\rangle$	BTp	$+3Bp$
Space complexity	$pd+$ $BT(p+d)$	$BT(p+d)$	$+Bpd$	$+2BT^2$	$+\min\{2BT^2, 2Bpd\}$	p	$+Bp$
# back-prop		1	1	2	2	1	1
forward hook		✗	✓	✓	✓	✗	✗

Table 2: Per-layer time and space complexity of training on weights (full fine-tuning) and biases. '$+$' means additional overhead to non-DP training, and '$\langle\rangle$' means between two values.

Here, the DP weight training (full fine-tuning) uses three efficient implementations that are equivalent mathematically but have different complexity: Opacus Yousefpour et al. (2021), GhostClip Goodfellow (2015); Li et al. (2021), and MixGhostClip Bu et al. (2022a). The first two implementations are illustrated in Figure 2, of which MixGhostClip is a hybridization that reduces to GhostClip when T is small. These implementations have been thoroughly analyzed in (Bu et al., 2022a, Appendix C), and we take the complexity result from (Bu et al., 2022a, Table 1). For the complexity of bias training in Table 2, it suffices to analyze Line 5 of Algorithm 1. We refer the interested readers to Table 5 for details, where we also apply the complexity analysis of weight training on other methods beyond full fine-tuning, including DP LoRA and DP Adapter.

3.3 Scalability of DP algorithms

From the complexity analysis in Table 2, we observe that DP training on weights can be memory costly, especially when the models are large and the data is high-dimensional. In sharp contrast, DP-BiTFiT is amazingly scalable since its computational overhead is negligible and independent of T (though the total complexity, mainly due to forward and output gradient, is still linear in T).

Efficiency of DP training v.s. feature dimension To empirically evaluate the computation efficiency of DP fine-tuning methods, we measure the time and GPU memory for a fixed batch size. We depict the high-dimensional data issue in Figure 3, in which the memory saving and speedup by DP-BiTFiT is substantial. We expect to observe greater efficiency advantage of DP-BiTFiT on higher dimensional data, e.g. in document-level language tasks with $T \approx 20000$ by Beltagy et al. (2020), and in high-resolution image tasks, such as 1024×1024 CelebA-HQ Karras et al. (2018) and Flickr-Faces-HQ Karras et al. (2019) where T can be of order 10^5 in the convolution layers.

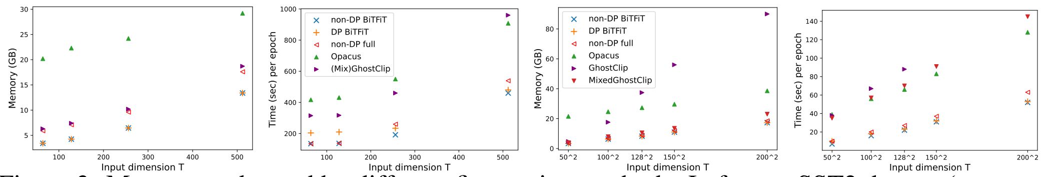

Figure 3: Memory and speed by different fine-tuning methods. Left two: SST2 dataset (sequence length T; MixGhostClip is equivalent to GhostClip for this small T) with RoBERTa-base and batch size 20. Right two: 50000 images of $\sqrt{T} \times \sqrt{T}$ pixels with ResNet50 and batch size 200.

Efficiency of DP training v.s. model size To stress-test the computation efficiency of DP-BiTFiT with large models, we apply the maximum batch size with respect to each fine-tuning method, instead of using a fixed one across different methods. Therefore, DP-BiTFiT can further leverage its memory efficiency to achieve the best throughput. Here we consider a setting of high-dimensional data ($T = 512^2$) but small ResNet ($11.7 \sim 58.2$M parameters) and the other setting of low-dimensional data ($T = 100$) but large GPT2 ($125 \sim 774$M parameters).

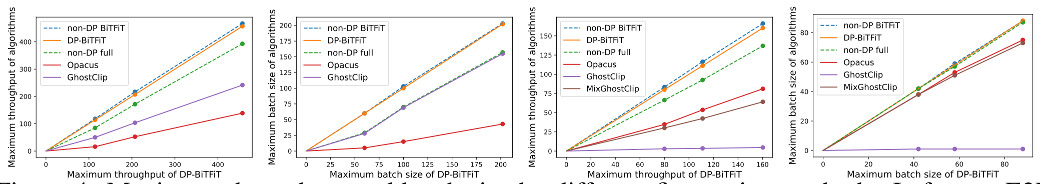

Figure 4: Maximum throughput and batch size by different fine-tuning methods. Left two: E2E dataset with GPT2-small/medium/large (MixGhostClip is equivalent to GhostClip for this small T). Right two: 50000 images of 512×512 pixels with ResNet 50/101/152.

4 Experiments

We now test the accuracy of DP-BiTFiT on natural language and computer vision tasks, with the settings in Appendix D. For DP full fine-tuning algorithms, we use GhostClip Li et al. (2021) on texts, and MixedGhostClip Bu et al. (2022a) on images, which achieve SOTA efficiency and accuracy on these datasets respectively. We report ϵ using RDP Mironov (2017).

4.1 Language tasks

We experiment on MNLI-m(mismatch) Williams et al. (2018), QQP Iyer et al. (2017), QNLI Rajpurkar et al. (2016), and SST2 datasets Socher et al. (2013). Competitive algorithms include reparameterized gradient perturbation (RGP, Yu et al. (2021b)), LoRA, Adapter and Compacter Yu et al. (2021a). We use the same setup as Li et al. (2021) on RoBERTa models, only increasing the learning rate for DP-BiTFiT. Additional results with different clipping functions and under a stronger privacy guarantee $\epsilon = 3$ can be found in Table 10.

In Table 3, DP-BiTFiT is highly parameter efficiency and on-par with other DP fine-tuning in terms of accuracy. As indicated by Figure 1 and Figure 3, over $2\times$ speedup and over $3\times$ memory saving is observed, when switching from DP full fine-tuning to DP-BiTFiT across datasets.

We also provide the evidence that DP-BiTFiT is performant, even outperforming DP full fine-tuning, on the natural language generation. We refer interested readers to Table 11 where we apply GPT2 to E2E dataset, under the same setting as Li et al. (2021); Bu et al. (2022b).

	Full Li et al. (2021)		RGP Yu et al. (2021a)	Adapter Yu et al. (2021a)	LoRA Yu et al. (2021a)		BiTFiT Ours		Compacter Yu et al. (2021a)
Additional params to networks	✗		✗	✓	✓		✗		✓
Forward caching activations	✓		✓	✓	✓		✗		✓
RoBERTa-base (125M)									
% of trainable params	100%		100%	1.4%	0.94%		0.083%		0.055%
	standard	DP	DP	DP	standard	DP	standard	DP	DP
Accuracy SST2	94.5	92.1	91.6	92.5	95.1	92.2	93.5	92.4	92.3
Accuracy QNLI	91.4	87.9	87.2	87.5	93.3	87.3	87.3	86.5	85.1
Accuracy QQP	87.3	86.1	85.5	85.6	90.8	85.7	86.1	83.4	84.7
Accuracy MNLI-m	85.9	83.2	80.1	83.4	87.5	83.5	83.4	82.6	82.6
RoBERTa-large (355M)									
% of trainable params	100%		100%	1.4%	0.94%		0.077%		0.053%
	standard	DP	DP	DP	standard	DP	standard	DP	DP
Accuracy SST2	96.2	93.8	93.0	93.9	96.2	95.3	95.5	94.5	94.2
Accuracy QNLI	93.6	91.1	90.0	90.7	94.9	90.8	92.2	91.0	90.2
Accuracy QQP	87.9	87.5	86.7	86.3	91.6	87.4	87.9	86.5	86.2
Accuracy MNLI-m	90.3	87.0	86.1	87.7	90.6	87.8	89.3	87.6	87.5

Table 3: Accuracy of fine-tuning methods with RoBERTa, under $\epsilon = 8$. More non-private fine-tuning results (similar to here) can be found in Yu et al. (2021a); Hu et al. (2021); Zaken et al. (2022). Note that linear probing of RoBERTa-base only gets 87.2% on SST2 and 77.3% on QNLI.

4.2 Image classification

We further experiment with DP-BiTFiT on CIFAR10/CIFAR100 (32×32 pixels, resized to 224×224) and CelebA (218×178 pixels, not resized) after pre-training on ImageNet (224×224 pixels). Unlike language tasks, DP-BiTFiT can be less satisfactory, e.g. inducing 30% test accuracy gap in CelebA [Smiling] classification in Table 12, though this gap can often be closed by a *two-phase training*, denoted as X+BiTFiT: we firstly apply DP full fine-tuning for X epochs then DP-BiTFiT for the rest of training. Hence, X+BiTFiT becomes DP full fine-tuning when X equals total epochs, and reduces to DP-BiTFiT when $X = 0$. Empirically speaking, it suffices to use $X \leq 2$ to achieve comparable accuracy to full fine-tuning, while still enjoying the speedup. The effectiveness of two-phase training is verified in Table 4 and Appendix E.3. 1+BiTFiT outperforms previous SOTA by DP full fine-tuning Bu et al. (2022a) that used BEiT-large: CIFAR10 $97.1\% \to 98.8\%$; CIFAR100 $86.2\% \to 88.7\%$, under $\epsilon = 2$. 2+BiTFiT is comparable to previous SOTA, $87.05/87.58\% \to 86.54/86.71\%$ on CelebA in Table 12, under $\epsilon = 3/8$.

CIFAR10	DP-BiTFiT	1+BiTFiT	2+BiTFiT	DP full
$\epsilon = 1$	11.7	98.2	97.9	97.2
$\epsilon = 2$	10.0	98.3	98.0	97.3
$\epsilon = 4$	13.8	98.2	98.0	97.5
$\epsilon = 8$	10.1	98.5	98.0	97.8
CIFAR100	DP-BiTFiT	1+BiTFiT	2+BiTFiT	DP full
$\epsilon = 1$	1.0	86.9	87.8	87.0
$\epsilon = 2$	1.0	88.7	89.3	88.7
$\epsilon = 4$	1.0	89.7	89.7	89.6
$\epsilon = 8$	1.0	90.3	90.7	90.0

Table 4: Accuracy of two-phase training (ours) and full fine-tuning by Bu et al. (2022a) with BEiT-large (Bao et al., 2021) (previous SOTA).

5 Discussion

In this work, we study DP-BiTFiT to privately train only the bias terms of neural networks. The highlight of DP-BiTFiT is the accuracy, the parameter efficiency and the computation efficiency, which is realized by not forward caching the activation tensors, and not back-propagating the gradient of weights. This consequently allows DP-BiTFiT to be as fast and memory-saving as its non-private counterpart, thus particularly suitable for large models and high-dimension data.

For future directions, DP-BiTFiT can be readily combined with prefix-based tuning and *weights*-based fine-tuning, e.g. DP Adapter+BiTFiT and DP LoRA+BiTFiT, via $f(\boldsymbol{x}; \mathbf{W}_0, \hat{\mathbf{b}}, \hat{\boldsymbol{\theta}})$ using the notation in Equation (1). Specifically, such combination can decide whether weight and/or bias should be fine-tuned in a **layer-wise manner**. For instance, we can optimize only the embedding layer (which has no bias terms) and all bias terms in other layers. We expect this interpolating approach between full fine-tuning and BiTFiT, in parallel to our two-phase training, to circumvent the limitation that DP-BiTFiT is sometimes sub-optimal on small models or difficult tasks.

References

Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 308–318, 2016.

Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. Beit: Bert pre-training of image transformers. In *International Conference on Learning Representations*, 2021.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Zhiqi Bu, Jinshuo Dong, Qi Long, and Weijie J Su. Deep learning with gaussian differential privacy. *Harvard data science review*, 2020(23), 2020.

Zhiqi Bu, Jialin Mao, and Shiyun Xu. Scalable and efficient training of large convolutional neural networks with differential privacy. *arXiv preprint arXiv:2205.10683*, 2022a.

Zhiqi Bu, Yu-Xiang Wang, Sheng Zha, and George Karypis. Automatic clipping: Differentially private deep learning made easier and stronger. *arXiv preprint arXiv:2206.07136*, 2022b.

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2633–2650, 2021.

Soham De, Leonard Berrada, Jamie Hayes, Samuel L Smith, and Borja Balle. Unlocking high-accuracy differentially private image classification through scale. *arXiv preprint arXiv:2204.13650*, 2022.

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

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pp. 265–284. Springer, 2006.

Ian Goodfellow. Efficient per-example gradient computations. *arXiv preprint arXiv:1510.01799*, 2015.

Sivakanth Gopi, Yin Tat Lee, and Lukas Wutschitz. Numerical composition of differential privacy. *Advances in Neural Information Processing Systems*, 34, 2021.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Niv Haim, Gal Vardi, Gilad Yehudai, Ohad Shamir, and Michal Irani. Reconstructing training data from trained neural networks. *arXiv preprint arXiv:2206.07758*, 2022.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

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

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

Shankar Iyer, Nikhil Dandekar, and Kornel Csernai. First quora dataset release: Question pairs, 2017. URL `https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs`.

Paras Jain, Ajay Jain, Aniruddha Nrusimha, Amir Gholami, Pieter Abbeel, Joseph Gonzalez, Kurt Keutzer, and Ion Stoica. Checkmate: Breaking the memory wall with optimal tensor rematerialization. *Proceedings of Machine Learning and Systems*, 2:497–511, 2020.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *International Conference on Learning Representations*, 2018.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4401–4410, 2019.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pp. 4171–4186, 2019.

Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2661–2671, 2019.

Antti Koskela, Joonas Jälkö, and Antti Honkela. Computing tight differential privacy guarantees using fft. In *International Conference on Artificial Intelligence and Statistics*, pp. 2560–2569. PMLR, 2020.

Alexey Kurakin, Steve Chien, Shuang Song, Roxana Geambasu, Andreas Terzis, and Abhradeep Thakurta. Toward training at imagenet scale with differential privacy. *arXiv preprint arXiv:2201.12328*, 2022.

Jaewoo Lee and Daniel Kifer. Scaling up differentially private deep learning with fast per-example gradient clipping. *arXiv preprint arXiv:2009.03106*, 2020.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 7871–7880, 2020.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Sasko, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 175–184, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. URL `https://aclanthology.org/2021.emnlp-demo.21`.

Xuechen Li, Florian Tramer, Percy Liang, and Tatsunori Hashimoto. Large language models can be strong differentially private learners. *arXiv preprint arXiv:2110.05679*, 2021.

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

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. *arXiv preprint arXiv:2106.04647*, 2021.

Harsh Mehta, Abhradeep Thakurta, Alexey Kurakin, and Ashok Cutkosky. Large scale transfer learning for differentially private image classification. *arXiv preprint arXiv:2205.02973*, 2022.

Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th computer security foundations symposium (CSF)*, pp. 263–275. IEEE, 2017.

Ilya Mironov, Kunal Talwar, and Li Zhang. Rényi differential privacy of the sampled gaussian mechanism. *arXiv preprint arXiv:1908.10530*, 2019. URL `http://arxiv.org/abs/1908.10530`.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992.

Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. Micro-batch training with batch-channel normalization and weight standardization. *arXiv preprint arXiv:1903.10520*, 2019.

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

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pp. 3–18. IEEE, 2017.

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

Pranav Subramani, Nicholas Vadivelu, and Gautam Kamath. Enabling fast differentially private sgd via just-in-time compilation and vectorization. *Advances in Neural Information Processing Systems*, 34, 2021.

Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pp. 843–852, 2017.

Yu-Xiang Wang, Borja Balle, and Shiva Prasad Kasiviswanathan. Subsampled rényi differential privacy and analytical moments accountant. In *International Conference on Artificial Intelligence and Statistics*, pp. 1226–1235. PMLR, 2019.

Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 1112–1122. Association for Computational Linguistics, 2018. URL `http://aclweb.org/anthology/N18-1101`.

Xiaodong Yang, Huishuai Zhang, Wei Chen, and Tie-Yan Liu. Normalized/clipped sgd with perturbation for differentially private non-convex optimization. *arXiv preprint arXiv:2206.13033*, 2022.

Ashkan Yousefpour, Igor Shilov, Alexandre Sablayrolles, Davide Testuggine, Karthik Prasad, Mani Malek, John Nguyen, Sayan Ghosh, Akash Bharadwaj, Jessica Zhao, Graham Cormode, and Ilya Mironov. Opacus: User-friendly differential privacy library in PyTorch. *arXiv preprint arXiv:2109.12298*, 2021.

Da Yu, Saurabh Naik, Arturs Backurs, Sivakanth Gopi, Huseyin A Inan, Gautam Kamath, Janardhan Kulkarni, Yin Tat Lee, Andre Manoel, Lukas Wutschitz, et al. Differentially private fine-tuning of language models. *arXiv preprint arXiv:2110.06500*, 2021a.

Da Yu, Huishuai Zhang, Wei Chen, Jian Yin, and Tie-Yan Liu. Large scale private learning via low-rank reparametrization. In *International Conference on Machine Learning*, pp. 12208–12218. PMLR, 2021b.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1–9, 2022.

A Detailed analysis of back-propagation

We rigorously analyze the neural network represented in Section 2.2: for sample index $i \in [B]$,

$$\underbrace{\boldsymbol{a}_{l+1,i}}_{\mathbb{R}^{T\times d'}} = \phi(\underbrace{\boldsymbol{s}_{l,i}}_{\mathbb{R}^{T\times p}}), \qquad \boldsymbol{s}_{l,i} = \underbrace{\boldsymbol{a}_{l,i}}_{\mathbb{R}^{T\times d}} \underbrace{\mathbf{W}_l}_{\mathbb{R}^{d\times p}} + \underbrace{\mathbf{1}}_{\mathbb{R}^{T\times 1}} \cdot \underbrace{\mathbf{b}_l}_{\mathbb{R}^{1\times p}}, \tag{5}$$

Then the per-sample weight gradient is given by the chain rule as

$$\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}_l}^\top = \sum_j \frac{\partial \mathcal{L}_i}{\partial \boldsymbol{s}_{l,j}}^\top \frac{\partial \boldsymbol{s}_{l,j}}{\partial \mathbf{W}_l} = \frac{\partial \mathcal{L}_i}{\partial \boldsymbol{s}_{l,i}}^\top \frac{\partial \boldsymbol{s}_{l,i}}{\partial \mathbf{W}_l} = \frac{\partial \mathcal{L}_i}{\partial \boldsymbol{s}_{l,i}}^\top \boldsymbol{a}_{l,i} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{s}_{l,i}}^\top \boldsymbol{a}_{l,i}$$

in which the second equality holds when there is no parameter sharing (so that each per-sample loss only depends on i-th input and output). The last equality holds for the same reason.

Similarly, we have the per-sample bias gradient as

$$\frac{\partial \mathcal{L}_i}{\partial \mathbf{b}_l}^\top = \sum_j \frac{\partial \mathcal{L}_i}{\partial \boldsymbol{s}_{l,j}}^\top \frac{\partial \boldsymbol{s}_{l,j}}{\partial \mathbf{b}_l} = \frac{\partial \mathcal{L}_i}{\partial \boldsymbol{s}_{l,i}}^\top \frac{\partial \boldsymbol{s}_{l,i}}{\partial \mathbf{b}_l} = \frac{\partial \mathcal{L}_i}{\partial \boldsymbol{s}_{l,i}}^\top \mathbf{1} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{s}_{l,i}}^\top \mathbf{1}.$$

We additionally demonstrate that bias gradient is independent of the input $\boldsymbol{a}_l$, on the convolution (1d/2d/3d) and the normalization layers. For the convolution, $\boldsymbol{s}_l$ is the inversely folded output and $\boldsymbol{a}_l$ is the unfolded input, then the forward pass is the same as that of linear layer in Equation (5). Notice that T is the product of hidden feature dimension (c.f. Bu et al. (2022a)), which depends on the padding, kernel sizes, strides, etc. For the batch, layer, group, and instance normalization, the forward pass is

$$\boldsymbol{s}_{l,i} = \frac{\boldsymbol{a}_{l,i} - \mathbb{E}(\boldsymbol{a}_l)}{\sqrt{\text{Var}(\boldsymbol{a}_l) + 0.00001}} \cdot \mathbf{W}_l + \mathbf{1} \cdot \mathbf{b}_l$$

which can be analyzed similarly to that of Equation (5).

B Implementation of DP-BiTFiT

In this section we describe the implementation of DP-BiTFiT, which only uses Pytorch backward hook but not the forward hook, and thus is different from existing packages such as FastGradClip Lee & Kifer (2020), Opacus Yousefpour et al. (2021), Private Transformers Li et al. (2021), Private CNN Bu et al. (2022a). Notice that in these packages, the forward hook is used to store the activation tensor $\boldsymbol{a}_l$ for all layers, which incurs huge memory burden.

The Pytorch backward hook is a function, to be registered on a torch Module (or a layer in the neural network), that will be executed in the backward propagation. The backward hook automatically extracts the input gradient $\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}_l}$ and the output gradient $\frac{\partial \mathcal{L}}{\partial \boldsymbol{s}_l}$ of the layer.

In DP-BiTFiT, we call `register_backward_hook` to register a backward hook for Line 5 of Algorithm 1. An example for a linear layer: $\mathbb{R}^{B\times T\times d} \to \mathbb{R}^{B\times T\times p}$ looks like

```
def hook(linear_layer, grad_input, grad_output):
    linear_layer.bias.grad_sample = grad_output.sum(dim=1)
    linear_layer.bias.norm_sample = linear_layer.bias.grad_sample.norm(2,dim=1)
```

Here the attribute `norm_sample` stores the per-sample gradient norm $\left\| \frac{\partial \mathcal{L}_i}{\partial \mathbf{b}_l} \right\|_F$, and the attribute `grad_sample` stores the $\mathbb{R}^{B\times p}$ per-sample gradient of bias.

Then the implementation of DP-BiTFiT for one iteration looks like

```
output=model(input)
loss=F.cross_entropy()(output,label)
torch.autograd.grad(loss,biases)
all_layer_norm_sample = torch.stack([param.norm_sample for param in biases],dim=0).norm(2, dim=0)
clipping_factor=1/(all_layer_norm_sample+0.01)
for layer in model.modules():
    layer.bias.grad=torch.einsum("i,i...->...", clipping_factor,layer.bias.grad_sample)
optimizer.step()
optimizer.zero_grad()
```

where `biases` is the collection of all bias terms in all layers.

C Complexity analysis

We provide more details on analyzing the time and space complexity. The analysis for full fine-tuning has been presented in (Bu et al., 2022a, Appendix C) and is adapted here for the parameter efficient fine-tuning: for example, Adapter Houlsby et al. (2019) uses two matrices $W_{down} \in \mathbb{R}^{p\times r}, W_{up} \in \mathbb{R}^{r\times p}$ that constitute

$$x \longleftarrow x + \text{GeLU}(x \cdot W_{down})W_{up}$$

Hence the complexity, in comparison to full-finetuning, changes by replacing $d \to 2r$.

LoRA Hu et al. (2021) also uses two matrices $W_{down} \in \mathbb{R}^{d\times r}, W_{up} \in \mathbb{R}^{r\times p}$ that constitute

$$x \longleftarrow x \cdot W + x \cdot W_{down}W_{up}$$

Hence the complexity, in comparison to full-finetuning, changes by replacing $pd \to r(p + d)$.

	forward &output grad	weight training				bias training	
		non-DP	DP full (Opacus)	DP LoRA	DP Adapter	non-DP	DP (ours)
Time complexity	$4BTpd$	$2BTpd$	$+2BTpd$	$+2BT(pr + dr)$	$+4BTpr$	BTp	$+3Bp$
Space complexity	$pd + BTd$	$BT(p + d)$	$+Bpd$	$+B(pr + dr)$	$+2Bpr$	p	$+Bp$
# back-prop		1	1	1	1	1	1
forward hook		✗	✓	✓	✓	✗	✗

Table 5: Per-layer time and space complexity of training on weights (full and parameter efficient fine-tuning) and biases. '+' means additional overhead to non-DP training.

For per-sample bias gradient clipping, we need $\frac{\partial \mathcal{L}_i}{\partial \mathbf{b}_l}^\top = \frac{\partial \mathcal{L}}{\partial \boldsymbol{s}_{l,i}}^\top \mathbf{1}$ in Equation (4), which consists of the *per-sample gradient instantiation* (i.e. summation along the feature dimension, from $\mathbb{R}^{Tp} \to \mathbb{R}^p$, $\frac{\partial \mathcal{L}}{\partial \boldsymbol{s}_{l,i}} \to \frac{\partial \mathcal{L}_i}{\partial \mathbf{b}_l}$), and computing the per-sample gradient norm (i.e. *taking the square* at each index and *summing all indices*). Here each operation in italic takes Bp time complexity, meaning the total time complexity is $3Bp$, but the space complexity is Bp if operated in-place.

D Experiment details

D.1 Language tasks

Throughout this work, the text datasets are processed and loaded from Huggingface Lhoest et al. (2021). We follow the same setup as Li et al. (2021); Bu et al. (2022b), such as $\delta = 0.5\times$sample size. The full fine-tuning is implemented by Private Transformers codebase, version 0.2.0 (i.e. GhostClip algorithm Li et al. (2021)).

For text classification, we experiment on four datasets: **MNLI(m)**, the matched splits from Multi-Genre Natural Language Inference Corpus; **QQP**, the Quora Question Pairs2 dataset; **QNLI** The Stanford Question Answering dataset; **SST2** The Stanford Sentiment Treebank dataset.

To give a fair comparison, we use the same optimizer as in Li et al. (2021), i.e. DP-Adam with Abadi's clipping.

Dataset	MNLI	QQP	QNLI	SST2
epoch	18	18	6	3
batch size	6000	6000	2000	1000
clipping threshold R	0.1			
DP learning rate	full 5e-4 / BiTFiT 5e-3			
non-DP learning rate	full 5e-5 / BiTFiT 1e-3			
max sequence length	256			

Table 6: Hyperparameters of text classification in Table 3 and Table 10, using RoBERTa (base/large).

For E2E generation task, we experiment GPT2 models using the same optimizer as in Bu et al. (2022b), using DP-AdamW with automatic clipping.

Model	GPT2-small	GPT2-medium	GPT2-large
epoch		10	
batch size		1024	
DP learning rate (full)	2e-3	2e-3	2e-3
non-DP learning rate (full)	2e-4	1e-4	1e-4
DP learning rate (BiTFiT)		1e-2	
non-DP learning rate (BiTFiT)		2e-3	
learning rate decay		No	
max sequence length		100	

Table 7: Hyperparameters of E2E generation task in Table 11, using GPT2.

D.2 Image tasks

We give the experiments settings for image classification. For CIFAR10 and CIFAR100, we use the same setting as Bu et al. (2022a), e.g. 5 epochs for CrossViT/ViT and 3 epochs for BEiT-large. For CelebA, we use the same setting as Bu et al. (2022b), e.g. 10 epochs.

We use DP-Adam with Abadi's clipping. We do not apply tricks such as random data augmentation, weight standardization Qiao et al. (2019), or parameter averaging Polyak & Juditsky (1992). Our experiments are heavily based on Private CNN (i.e. MixGhostClip algorithm Bu et al. (2022a)) and TIMM codebases.

Dataset	CIFAR10	CIFAR10	CIFAR100	CelebA
Model	CrossViT	BEiT-large	BEiT-large	ResNet18
epoch	5	3	3	10
batch size	1000	1000	1000	500
clipping threshold		0.1		
DP learning rate (full)		1e-3		
DP learning rate (BiTFiT)		5e-3		
learning rate decay		No		
normalizing data	Yes	Yes	Yes	No

Table 8: Hyperparameters of image classification.

E Additional tables and figures

E.1 Parameter efficiency of DP-BiTFiT

Model	Number of params	% of params
VGG11	133M	0.009
VGG16	138M	0.009
VGG19	144M	0.010
ResNet18	11.7M	0.043
ResNet34	21.8M	0.044
ResNet50	25.6M	0.113
ResNet101	44.5M	0.121
ResNet152	60.2M	0.127
wide_resnet50_2	68.9M	0.051
wide_resnet101_2	126.9M	0.055
convnext_base	88.6M	0.148
convnext_large	197.8M	0.099
ViT-small-patch16	22.0M	0.238
ViT-base-patch16	86.6M	0.120
ViT-large-patch16	304M	0.090
beit_base_patch16_224	86.5M	0.088
deit_base_patch16_224	86.4M	0.120
GPT2-small	124M	0.082
GPT2-medium	355M	0.076
GPT2-large	774M	0.066
RoBERTa-base	125M	0.083
RoBERTa-large	355M	0.077
BERT-base-uncased	109M	0.094
BERT-large-uncased	335M	0.081
BART-large	406M	0.082
longformer-base-4096	149M	0.088
longformer-large-4096	435M	0.080

Table 9: Parameter efficiency of (DP) BiTFiT on various models.

E.2 More results on DP-BiTFiT and language tasks

	full (Li et al., 2021; Bu et al., 2022b)					BiTFiT (ours)				
RoBERTa-base										
	standard $\epsilon=\infty$	DP_{Abadi} $\epsilon=8$	DP_{AUTO} $\epsilon=8$	DP_{Abadi} $\epsilon=3$	DP_{AUTO} $\epsilon=3$	standard $\epsilon=\infty$	DP_{Abadi} $\epsilon=8$	DP_{AUTO} $\epsilon=8$	DP_{Abadi} $\epsilon=3$	DP_{AUTO} $\epsilon=3$
Accuracy SST2	94.5	92.1	92.4	91.9	92.3	93.5	92.4	92.4	92.0	92.0
Accuracy QNLI	91.4	87.9	87.9	87.4	86.9	87.3	86.5	86.7	86.4	86.1
Accuracy QQP	87.3	86.1	86.6	85.6	85.8	86.1	83.4	84.0	83.0	83.8
Accuracy MNLI-m	85.9	83.2	83.8	82.5	83.2	83.4	82.6	82.6	81.5	82.0
RoBERTa-large										
	standard $\epsilon=\infty$	DP_{Abadi} $\epsilon=8$	DP_{AUTO} $\epsilon=8$	DP_{Abadi} $\epsilon=3$	DP_{AUTO} $\epsilon=3$	standard $\epsilon=\infty$	DP_{Abadi} $\epsilon=8$	DP_{AUTO} $\epsilon=8$	DP_{Abadi} $\epsilon=3$	DP_{AUTO} $\epsilon=3$
Accuracy SST2	96.2	93.8	94.6	93.0	93.9	95.5	94.5	94.7	94.5	94.6
Accuracy QNLI	93.6	91.1	91.5	90.8	91.0	92.2	91.0	91.1	90.3	90.8
Accuracy QQP	87.9	86.9	87.5	86.6	86.8	87.9	86.5	87.1	86.3	86.5
Accuracy MNLI-m	90.3	87.0	87.1	86.4	86.3	89.3	87.6	87.7	87.2	87.2

Table 10: Accuracy of full fine-tuning and BiTFiT with RoBERTa, under different per-sample clipping functions (indicated as subscript, Abadi Abadi et al. (2016) and AUTO-S Bu et al. (2022b)). Same setting as Appendix D.

Model	Fine-tuning	% of params	Privacy↓	Perplexity↓	BLEU↑	ROGUE-L↑	NIST↑	METEOR↑	CIDEr↑
GPT2-small (124M)	full	100%	standard	2.91	69.46	71.36	8.78	0.46	2.42
			DP ($\epsilon = 8$)	2.33	63.60	67.07	7.71	0.40	1.94
			DP ($\epsilon = 3$)	2.36	61.34	65.87	7.07	0.39	1.80
	LoRA	—	standard	—	69.68	71.71	8.82	0.46	2.49
			DP ($\epsilon = 8$)	—	63.39	67.53	7.45	0.41	1.95
			DP ($\epsilon = 3$)	—	58.15	65.77	5.46	0.37	1.58
	prefix	—	standard	—	68.85	70.81	8.72	0.45	2.35
			DP ($\epsilon = 8$)	—	49.26	60.73	5.53	0.36	1.57
			DP ($\epsilon = 3$)	—	47.77	58.96	5.25	0.36	1.51
	BiTFiT	0.082%	standard	3.19	64.46	63.67	4.25	0.36	1.36
			DP ($\epsilon = 8$)	2.89	60.13	64.96	6.14	0.37	1.62
			DP ($\epsilon = 3$)	3.00	54.78	63.55	4.78	0.34	1.31
GPT2-medium (355M)	full	100%	standard	2.08	68.50	71.46	8.63	0.45	2.14
			DP ($\epsilon = 8$)	2.25	64.22	67.53	8.17	0.42	2.08
			DP ($\epsilon = 3$)	2.62	63.85	67.07	7.11	0.39	1.75
	BiTFiT	0.076%	standard	2.85	64.48	67.81	8.50	0.43	2.11
			DP ($\epsilon = 8$)	2.67	61.02	66.13	7.18	0.39	1.80
			DP ($\epsilon = 3$)	2.67	57.11	66.16	5.07	0.37	1.47
GPT2-large (774M)	full	100%	standard	1.79	66.84	70.38	8.73	0.46	2.36
			DP ($\epsilon = 8$)	2.26	64.64	68.97	8.30	0.42	2.16
			DP ($\epsilon = 3$)	2.65	64.18	67.86	7.94	0.40	2.01
	BiTFiT	0.066%	standard	2.79	65.79	67.61	8.55	0.43	2.21
			DP ($\epsilon = 8$)	2.59	65.21	67.88	8.43	0.42	2.15
			DP ($\epsilon = 3$)	2.61	65.18	67.90	8.34	0.42	2.12

Table 11: Accuracy of fine-tuning with GPT2 on E2E dataset. LoRA and prefix results are taken from Li et al. (2021). Same setting as Appendix D.

E.3 More results on two-phase training

Attributes	DP-BiTFiT	1+BiTFiT	2+BiTFiT	DP full	DP-BiTFiT	1+BiTFiT	2+BiTFiT	DP full
	$\epsilon = 3$				$\epsilon = 8$			
5 o Clock Shadow	90.01	90.01	90.14	91.32	90.01	90.01	90.51	91.64
Arched Eyebrows	71.56	73.12	76.01	77.33	71.56	73.74	75.49	78.82
Attractive	68.71	73.98	75.99	79.22	69.70	73.61	76.20	78.08
Bags Under Eyes	79.74	79.76	81.27	81.73	79.74	79.74	80.69	82.62
Bald	97.88	97.88	97.88	97.93	97.88	97.88	97.88	97.91
Bangs	84.43	84.43	84.80	94.06	84.43	84.44	86.51	94.22
Big Lips	67.30	67.30	67.30	67.78	67.30	67.30	67.29	68.34
Big Nose	78.80	78.95	80.08	81.19	78.80	78.92	79.23	81.86
Black Hair	72.84	74.86	82.37	85.84	73.02	78.71	83.33	86.47
Blond Hair	89.54	93.00	93.28	94.17	89.13	92.62	93.88	94.34
Blurry	94.94	94.94	94.94	95.05	94.94	94.94	94.96	95.10
Brown Hair	82.03	82.02	82.87	85.44	82.03	82.37	83.49	85.04
Bushy Eyebrows	87.05	87.05	87.21	88.26	87.05	87.05	87.15	89.02
Chubby	94.70	94.70	94.70	94.84	94.70	94.70	94.70	94.78
Double Chin	95.43	95.43	95.43	95.49	95.43	95.43	95.43	95.39
Eyeglasses	93.54	93.54	93.54	94.30	93.54	93.54	93.54	95.85
Goatee	95.42	95.42	95.42	95.96	95.42	95.42	95.42	95.89
Gray Hair	96.81	96.81	96.85	97.44	96.81	96.81	97.12	97.45
Heavy Makeup	76.51	82.76	85.71	88.48	77.22	83.03	85.86	89.05
High Cheekbones	62.13	68.20	81.63	83.77	61.43	67.27	81.33	84.20
Male	80.37	88.47	91.52	94.73	82.04	88.52	92.14	95.19
Mouth Slightly Open	54.03	59.32	77.61	86.75	55.26	60.70	79.42	90.24
Mustache	96.13	96.13	96.13	96.10	96.13	96.13	96.13	96.12
Narrow Eyes	85.13	85.13	85.13	85.14	85.13	85.13	85.13	85.16
No Beard	85.37	85.87	87.56	92.94	85.37	85.88	88.59	93.59
Oval Face	70.44	70.94	71.50	73.11	70.44	71.48	71.92	71.77
Pale Skin	95.79	95.79	95.79	95.79	95.79	95.79	95.79	95.79
Pointy Nose	71.43	71.51	71.63	71.89	71.43	71.47	71.77	72.87
Receding Hairline	91.51	91.51	91.51	91.59	91.51	91.51	91.51	91.61
Rosy Cheeks	92.83	92.83	92.86	93.07	92.87	92.83	92.86	93.33
Sideburns	95.36	95.36	95.36	96.44	95.36	95.36	95.36	96.63
Smiling	60.07	66.32	85.85	89.34	58.92	65.97	85.55	89.11
Straight Hair	79.01	79.01	79.02	79.65	79.01	79.01	79.13	78.60
Wavy Hair	71.24	73.09	76.22	77.35	70.86	73.62	77.11	72.73
Wearing Earrings	79.34	79.34	80.37	83.24	79.34	79.34	80.71	84.36
Wearing Hat	95.80	95.80	95.80	96.01	95.80	95.80	95.80	97.02
Wearing Lipstick	80.61	87.90	89.81	91.59	80.35	87.20	89.56	91.94
Wearing Necklace	86.21	86.21	86.21	86.21	86.21	86.21	86.21	86.21
Wearing Necktie	92.99	92.99	93.03	93.58	92.99	92.99	93.11	93.57
Young	75.71	79.33	81.23	83.69	75.71	78.52	80.66	83.11
Average	82.97	84.42	86.54	88.20	83.01	84.52	86.71	88.38
Total time	10:30	12:02	13:34	25:50	10:30	12:02	13:34	25:50

Table 12: Accuracy on CelebA dataset with settings in Appendix D.2 from one run. DP full fine-tuning is implemented with the most efficient MixGhostClip algorithm Bu et al. (2022a).