A DETAILED ANALYSIS OF BACK-PROPAGATION

We rigorously analyze the neural network represented in Section 2.3: 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 as discussed in Section 2.4.

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 8: 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 9: Hyperparameters of text classification in Table 4 and Table 13, 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 10: Hyperparameters of E2E generation task in Table 5 and Table 14, 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 11: Hyperparameters of image classification task in Section 4.3,Table 15,Table 16,Table 17.

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 12: 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 13: 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 14: 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

CIFAR10					
Model	Privacy	DP-BiTFiT	1+BiTFiT	2+BiTFiT	DP full
beit_large_patch16_224	$\epsilon = 1$	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
beit_base_patch16_224	$\epsilon = 1$	10.0	96.6	96.0	95.4
	$\epsilon = 2$	10.7	97.1	96.4	96.0
	$\epsilon = 4$	14.0	97.2	96.6	96.2
	$\epsilon = 8$	10.0	97.2	96.5	96.3
deit_base_patch16_224	$\epsilon = 1$	78.2	94.4	95.2	95.4
	$\epsilon = 2$	75.0	95.4	95.2	95.6
	$\epsilon = 4$	72.9	95.8	95.9	96.0
	$\epsilon = 8$	71.2	96.1	96.0	96.3
crossvit_base_240	$\epsilon = 1$	74.3	92.4	94.3	95.2
	$\epsilon = 2$	80.4	93.6	95.0	95.3
	$\epsilon = 4$	81.0	94.9	95.8	95.7
	$\epsilon = 8$	78.2	94.8	95.8	96.2
vit_large_patch16_224	$\epsilon = 1$	89.7	98.9	98.7	98.9
	$\epsilon = 2$	90.6	98.8	98.9	98.9
	$\epsilon = 4$	93.2	98.9	98.8	99.0
	$\epsilon = 8$	93.9	99.0	98.9	99.0
vit_base_patch16_224	$\epsilon = 1$	86.7	95.2	97.0	96.8
	$\epsilon = 2$	89.3	97.7	97.1	97.1
	$\epsilon = 4$	88.3	97.7	97.2	97.2
	$\epsilon = 8$	88.7	97.6	97.2	97.4

Table 15: Accuracy of two-phase fine-tuning on CIFAR10. Same setting as Appendix D.2 except ViT uses the following learning rate: DP full fine-tuning 5e-4, DP-BiTFiT 5e-3.

CIFAR100					
Model	Privacy	DP-BiTFiT	1+BiTFiT	2+BiTFiT	DP full
beit_large_patch16_224	$\epsilon = 1$	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
beit_base_patch16_224	$\epsilon = 1$	1.0	81.4	82.2	80.9
	$\epsilon = 2$	1.0	83.4	83.4	83.1
	$\epsilon = 4$	1.0	84.6	85.1	84.8
	$\epsilon = 8$	1.0	84.9	85.6	85.2
deit_base_patch16_224	$\epsilon = 1$	10.9	49.1	65.9	69.1
	$\epsilon = 2$	13.6	58.1	71.5	74.3
	$\epsilon = 4$	15.7	64.5	73.9	77.1
	$\epsilon = 8$	16.6	69.7	75.7	77.9
crossvit_base_240	$\epsilon = 1$	12.2	49.2	61.7	67.6
	$\epsilon = 2$	12.3	56.8	65.3	71.6
	$\epsilon = 4$	17.2	61.6	70.4	73.1
	$\epsilon = 8$	20.9	63.4	72.8	74.2
vit_large_patch16_224	$\epsilon = 1$	14.0	73.5	86.0	87.7
	$\epsilon = 2$	19.4	82.4	89.0	90.1
	$\epsilon = 4$	24.3	87.5	89.9	91.0
	$\epsilon = 8$	23.9	89.0	90.7	91.3
vit_base_patch16_224	$\epsilon = 1$	16.0	64.3	79.5	83.9
	$\epsilon = 2$	22.9	77.0	83.8	85.5
	$\epsilon = 4$	21.2	83.0	85.2	87.2
	$\epsilon = 8$	26.2	83.8	86.5	87.1

Table 16: Accuracy of two-phase fine-tuning on CIFAR100. Same setting as Appendix D.2 except ViT uses the following learning rate: DP full fine-tuning 5e-4, DP-BiTFiT 5e-3.

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 17: 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).