A Further memory and speed comparison

Dataset	Model & # Param	Package	Time(sec) / Epoch	Memory (GB)
CIFAR10	ResNet18 11M	Opacus	59	8.30 | 14.05
		Ghost	65	2.21 | 3.31
		Mixed	44	2.21 | 3.31
		NonDP	14	2.20 | 3.31
	ResNet34 21M	Opacus	OOM	OOM
		Ghost	109	2.64 | 3.61
		Mixed	77	2.64 | 3.61
		NonDP	24	2.63 | 3.61
	ResNet50 23.5M	Opacus	OOM	OOM
		Ghost	174	8.85 | 11.6
		Mixed	137	8.85 | 11.6
		NonDP	53	8.7 | 11.6
	ResNet101 42.5M	Opacus	OOM	OOM
		Ghost	275	10.52 | 11.81
		Mixed	237	10.52 | 11.81
		NonDP	91	10.36 | 11.76
	ResNet152 58.2M	Opacus	OOM	OOM
		Ghost	350	12.54 | 13.90
		Mixed	389	12.54 | 13.90
		NonDP	133	12.39 | 13.89
	VGG11 9M	Opacus	40	6.19 | 14.11
		Ghost	18	1.85 | 2.89
		Mixed	16	1.85 | 2.89
		NonDP	5	1.83 | 2.86
	VGG13 9.4M	Opacus	43	6.72 | 14.18
		Ghost	29	1.94 | 3.53
		Mixed	22	1.94 | 3.53
		NonDP	7	1.93 | 3.53
	VGG16 14.7M	Opacus	OOM	OOM
		Ghost	35	2 | 3.57
		Mixed	28	2 | 3.57
		NonDP	9	1.98 | 3.57
	VGG19 20.0M	Opacus	OOM	OOM
		Ghost	40	2.05 | 3.63
		Mixed	33	2.05 | 3.63
		NonDP	11	2.03 | 3.59
	ResNeXt 9.1M	Opacus	162	10.77 | 12.51
		Ghost	189	6.93 | 7.05
		Mixed	140	6.93 | 7.05
		NonDP	54	6.56 | 6.99
	MobileNet 3.2M	Opacus	46	7.24 | 13.93
		Ghost	42	2.95 | 4.91
		Mixed	36	2.95 | 4.91
		NonDP	9	2.94 | 4.91

Table 6: Time and memory of models on CIFAR10, with physical batch size 128. There are two types of memory: active memory (left) and total memory (right). Out of memory (OOM) means the total memory exceeds 16GB. FastGradClip is excluded due to inflexibility to apply on general architectures.

Dataset	Model & # Param	Package	Time(sec) / Epoch	Memory (GB)	Max Batch Size	Min Time/Epoch
ImageNet	Resnet18 11.7M	Opacus	392	3.20/4.35	145	138
		Ghost	OOM	OOM	7	1093
		Mixed	410	1.74/2.34	325	370
		NonDP	349	1.73/2.34	678	114
	Resnet34 21.8M	Opacus	444	5.29/5.94	93	197
		Ghost	OOM	OOM	7	1482
		Mixed	478	2.01/2.62	282	428
		NonDP	373	2.01/2.62	455	114
	Resnet50 25.6M	Opacus	518	9.13/10.73	55	316
		Ghost	OOM	OOM	7	1896
		Mixed	545	4.49/5.93	129	343
		NonDP	385	4.47/5.93	161	136
	Resnet101 44.6M	Opacus	762	11.53/12.80	28	735
		Ghost	OOM	OOM	7	2816
		Mixed	784	5.53/6.65	89	578
		NonDP	430	5.51/6.65	99	232
	Resnet152 60.2M	Opacus	OOM	OOM	22	1365
		Ghost	OOM	OOM	7	3789
		Mixed	1109	6.77/7.91	57	887
		NonDP	500	6.75/7.91	83	348
	VGG11 132.9M	Opacus	OOM	OOM	<5	NA
		Ghost	OOM	OOM	0	NA
		Mixed	441	5.23/7.47	71	347
		NonDP	361	4.96/6.34	145	148
	VGG13 133.1M	Opacus	OOM	OOM	<5	NA
		Ghost	OOM	OOM	0	NA
		Mixed	630	7.51/12.46	40	610
		NonDP	375	5.86/9.76	99	195
	VGG16 138.4M	Opacus	OOM	OOM	<5	NA
		Ghost	OOM	OOM	0	NA
		Mixed	755	7.81/12.48	35	796
		NonDP	385	6.12/9.24	87	277
	VGG19 143.7M	Opacus	OOM	OOM	<5	NA
		Ghost	OOM	OOM	0	NA
		Mixed	891	8.11/12.35	30	870
		NonDP	395	6.37/9.28	90	380
	wide_resnet50_2 68.9M	Opacus	OOM	OOM	17	979
		Ghost	OOM	OOM	8	2242
		Mixed	709	7.52/12.19	91	626
		NonDP	409	7.5/12.19	115	257
	wide_resnet101_2 126.9M	Opacus	OOM	OOM	8	2125
		Ghost	OOM	OOM	8	3208
		Mixed	1210	9.01/13.59	53	1088
		NonDP	536	8.99/13.59	65	470
	resnext50_32x4d 25.0M	Opacus	590	10.04/13.34	40	511
		Ghost	OOM	OOM	10	2141
		Mixed	685	7.36/8.98	87	536
		NonDP	398	7.34/8.97	120	196
	Densenet121 8.0M	Opacus	851	6.92/7.97	73	645
		Ghost	OOM	OOM	10	2912
		Mixed	802	4.34/5.33	79	490
		NonDP	453	4.11/5.32	100	161
	Densenet169 14.2M	Opacus	1158	9.04/9.61	54	698
		Ghost	OOM	OOM	10	3533
		Mixed	1062	5.58/6.04	66	625
		NonDP	496	5.18/5.58	78	208

	Densenet201 20.0M	Opacus	1224	12.99/13.56	33	1481
		Ghost	OOM	OOM	10	3974
		Mixed	1265	8.39/8.99	48	805
		NonDP	547	7.91/8.96	56	280
	Alexnet 61.1M	Opacus	OOM	OOM	10	175
		Ghost	408	2.60/3.41	154	455
		Mixed	393	1.01/1.31	1111	122
		NonDP	379	1.01/1.31	2380	117
	squeezenet1_0 1.25M	Opacus	404	2.79/4.06	269	335
		Ghost	OOM	OOM	11	859
		Mixed	415	2.01/3.22	312	118
		NonDP	366	2.00/3.22	393	101
	squeezenet1_1 1.24M	Opacus	404	2.07/3.79	470	158
		Ghost	OOM	OOM	11	679
		Mixed	426	1.67/2.84	501	113
		NonDP	341	1.65/2.84	650	108

Table 7: Time and memory of models [42] on ImageNet. There are two types of memory: active memory (left) and total memory (right). Out of memory (OOM) means the total memory exceeds 16GB. FastGradClip is excluded due to inflexibility to apply on general architectures. For the time per epoch and the memory columns, we use fixed physical batch size 25. For the max (physical) batch size and the min time/epoch (using the max batch size) columns, we use bisection method. We use 50000 images as training set.

B Explaining convolutional layers

In a 2D convolutional layer, the input $\mathbf{a}$ to the layer has dimension $(B, d, H_{\text{in}}, W_{\text{in}})$ and the folded output $F(\mathbf{s})$ has dimension $(B, p, H_{\text{out}}, W_{\text{out}})$, where B is the batch size, d is the number of input channels, and p is the number of output channels. H, W are the height and width of images (or hidden features in hidden layers). $H_{\text{out}}, W_{\text{out}}$ can be calculated by `https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html` as

$$H_{\text{out}} = \left\lfloor \frac{H_{\text{in}} + 2 \times \text{ padding } - \text{dilation} \times (k_H - 1) - 1}{\text{stride}} + 1 \right\rfloor,$$

and

$$W_{\text{out}} = \left\lfloor \frac{W_{\text{in}} + 2 \times \text{ padding } - \text{dilation} \times (k_W - 1) - 1}{\text{stride}} + 1 \right\rfloor.$$

Following the above formulae, we recall that in the layerwise decision of mixed ghost clipping (3), the kernel size increases the right hand side and decreases the left hand size. In words, large kernel size always favors the ghost norm over the per-sample gradient instantiation!

To further explain the convolution, we consider the kernel size (k_H, k_W) to (2.5), which establishes the equivalence between linear layer and convolutional layer. See example in `https://pytorch.org/docs/stable/generated/torch.nn.Unfold.html`.

$$\overbrace{\mathbf{a}_i \longrightarrow U(\mathbf{a}_i) \longrightarrow U(\mathbf{a}_i)\mathbf{W} + \mathbf{b} \longrightarrow F(U(\mathbf{a}_i)\mathbf{W} + \mathbf{b})}^{\text{Conv2d}(\mathbf{a}_i)}$$
$$(H_{in}, W_{in}, d) \longrightarrow (H_{out}, W_{out}, dk_H k_W) \longrightarrow (H_{out}W_{out}, p) \longrightarrow (H_{out}, W_{out}, p)$$

C Complexity analysis

In this section, we analyze the time and space complexity of different modules in the DP training pipeline. Our analysis follows a per-layer fashion, as all the dimension constants are layer-specific but ignored only in this section.

To simplify the representation and avoid the folding/unfolding U, F, we refer the forward pass of convolutional layer in (2.5) to the equivalent formula of linear layer in (2.2), with $\mathbf{a}_i \in \mathbb{R}^{T \times D}$ denoting $U(\mathbf{a}_i)$, $\mathbf{s}_i \in \mathbb{R}^{T \times p}$ denoting $F^{-1}(\mathbf{s}_i)$. Here $T = H_{\text{out}}W_{\text{out}}, D = dk_H k_W$, where d, p, k, H, W

are layer-dependent and have been introduced in the previous section. We ignore the bias without loss of generality.

C.1 Forward pass, Intialization, etc.

We note that the activation $\mathbf{a}_i$ is created during the forward pass, and that $\mathbf{W}$ is created during the random intialization. Since we only initialize and forward pass once, this complexity is the same for all training procedures (DP or non-private), therefore we do not study it. We also omit some trivial operations such as converting from per-sample gradient norm to the clipping factor $C_i = \min\left(R/\|\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}}\|_{\text{Fro}}, 1\right)$.

In what follows, we will use the complexity of matrix multiplication repeatedly.

Lemma C.1. *For the matrix multiplication between $\mathbb{R}^{m\times n}$ and $\mathbb{R}^{n\times r}$, the space complexity is mr, and the time complexity is $2mnr$.*

C.2 Back-propagation

Referring to the back-propagation in (2.3), we derive the whole time complexity is the matrix multiplication of $(B\times T\times p)\cdot(p\times D)\circ(B\times T\times D)$, which is $2BTDp+2BTD$ and the space complexity is $BTp+pD+2BTD$.

The last step (2.6) results in the per-sample gradients, where

$$\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}} = \sum_i \frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}\mathbf{a}_i$$

which gives $2BTpD$ time complexity and pD space complexity (since per-sample gradients are summed in-place).

In total we have $4BTpD+2BTD$ time complexity and $BTp+2BTD+pD$ space complexity for one back-propagation.

For the second round of back-propagation, we add another time complexity $4BTpD+2BTD$ but no space complexity as the space is freed by $torch.optim.Optimizer.zero\_grad()$.

C.3 Ghost norm

In this section we study the procedure of computing the ghost norm. That is, from inputs $\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}, \mathbf{a}_i$ to the output $\|\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}}\|_{\text{Fro}}^2$.

As mentioned in (2.7), the clipping norm can be calculated as following:

$$\|\frac{\partial \mathcal{L}_i}{\partial W}\|_{\text{Fro}}^2 = \text{vec}(\mathbf{a}_i\mathbf{a}_i^\top)\text{vec}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}^\top\right)$$

where $\|\cdot\|_{\text{Fro}}$ is the Frobenius norm and vec flattens the matrices to vectors. Since $\mathbf{a}_i\in\mathbb{R}^{T\times D}, \mathbf{s}_i\in\mathbb{R}^{T\times p}$. We then compute and store $\mathbf{a}_i\mathbf{a}_i^\top\in\mathbb{R}^{T\times T}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}^\top\in\mathbb{R}^{T\times T}$, where time complexity is $2T^2D+2T^2p$ and the space complexity is $2T^2$. For B data points, the time complexity is $B(2T^2D+2T^2p)$ and the space complexity is $2BT^2$.

The final vector-vector product for a batch takes the time complexity is $B(2T^2-1)$ and space complexity B.

C.4 Gradient instantiation and the norm

In this section we study the procedure of computing norm via instantiating the per-sample gradients. That is, from inputs $\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}, \mathbf{a}_i$, to the intermediate $\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}}$, to the output $\|\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}}\|_{\text{Fro}}^2$.

To compute the per-sample gradients, which is not available in the first back-propagation due to the in-place summation, we need to re-compute

$$\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}\mathbf{a}_i.$$

For a batch, the time complexity is $2BTpD$ and the space complexity is BpD.

To calculate the norm of $\{\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}}\}_i$, each with size $D \times p$, the time complexity is $2BDp$ and the space complexity is B.

C.5 Weighted gradient

To calculate the weighted gradient, $\{\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}}\}_i \to \sum_i C_i \frac{\partial \mathcal{L}_i}{\partial \mathbf{W}}$, the time complexity is $2BpD$ with no space complexity.

C.6 Combining the modules to algorithms

- Ghost clipping = Back-propagation + Ghost norm + Second back-propagation
- Opacus = Back-propagation + Gradient instantiation + Weighted gradient
- FastGradClip = Back-propagation + Gradient instantiation + Second back-propagation
- Mixed ghost clipping = Back-propagation + min{Ghost norm, Gradient instantiation} + Second back-propagation

D ViT details

Our ViTs are imported from PyTorch Image Models [46]. For all ViTs, if they contain the batch normalization, we replace with the group normalization (16 groups). We freeze modules that are not supported by our privacy engine. We do not apply learning rate schedule, random data augmentation, weight standardization, or parameter averaging as in [9]. We describe the models as their configuration argument in [46].

Dataset	Model & # Param	Package	Memory (GB)	Accuracy (%)	Max Batch Size	Min Time/Epoch
CIFAR10	crossvit_18_240 42.6M	Mixed	4.49 | 5.00	95.08	72	700
		NonDP	4.07 | 4.87	97.11	78	420
	crossvit_15_240 27.0M	Mixed	3.42 | 3.48	93.97	95	507
		NonDP	3.08 | 3.31	96.66	103	303
	crossvit_9_240 8.2M	Mixed	1.63 | 1.66	88.67	187	248
		NonDP	1.45 | 1.63	93.33	212	180
	crossvit_base_240 103.9M	Mixed	7.33 | 7.49	95.22	48	1228
		NonDP	6.49 | 6.85	97.37	53	731
	crossvit_small_240 26.3M	Mixed	3.21 | 3.25	94.05	102	463
		NonDP	2.88 | 3.13	96.17	110	287
	crossvit_tiny_240 6.7M	Mixed	1.47 | 1.64	88.31	204	233
		NonDP	1.34 | 1.62	93.12	223	167
	deit_base_patch16_224 85.8M	Mixed	4.15 | 4.47	94.56	82	882
		NonDP	3.76 | 4.06	97.14	86	513
	deit_small_patch16_224 21.7M	Mixed	1.88 | 1.97	91.16	170	330
		NonDP	1.75 | 1.85	96.35	176	204
	deit_tiny_patch16_224 5.5M	Mixed	0.89 | 1.02	84.22	346	175
		NonDP	0.84 | 1.00	93.46	360	154
	beit_large_patch16_224 303.4M	Mixed	10.72 | 11.27	93.94	27	2703
		NonDP	9.83 | 10.23	97.80	30	1597
	beit_base_patch16_224 85.8M	Mixed	3.84 | 4.06	91.68	86	805
		NonDP	3.56 | 3.79	97.14	91	506
	convit_base 85.8M	Mixed	6.46 | 6.77	94.76	47	1115
		NonDP	5.93 | 6.31	96.82	50	649
	convit_small 27.3M	Mixed	3.45 | 3.65	93.16	86	513
		NonDP	3.22 | 3.51	97.07	90	311
	convit_tiny 5.5M	Mixed	1.54 | 1.63	86.56	199	236
		NonDP	1.47 | 1.57	94.51	200	157
	vit_base_patch16_224 85.8M	Mixed	4.80 | 5.13	94.40*	82	926
		NonDP	4.40 | 4.91	97.43*	86	550
	vit_small_patch16_224 21.7M	Mixed	2.04 | 2.13	92.77	170	334
		NonDP	1.92 | 2.04	97.69	176	204
	vit_tiny_patch16_224 5.5M	Mixed	0.93 | 1.04	87.56	346	179
		NonDP	0.89 | 1.02	95.20	360	163

Table 8: Performance of selected ViTs on CIFAR10 under $\epsilon = 2$. Here batch size 1000, physical batch size 20, except for max (physical) batch size and min time/epoch (using max batch size). There are two types of memory: active memory (left) and total memory (right). All ViTs use DP learning rate $2e-3$ and non-DP learning rate $2e-4$ by default, except the ViT base that uses half the learning rate, since the default learning rate gives $< 80\%$ accuracy.

Dataset	Model & # Param	Package	Memory (GB)	Accuracy (%)	Max Batch Size	Min Time/Epoch
CIFAR100	crossvit_18_240 42.7M	Mixed	4.49 | 5.00	71.78	72	696
		NonDP	4.07 | 4.87	79.46	78	421
	crossvit_15_240 27.0M	Mixed	3.42 | 3.50	67.21	95	495
		NonDP	3.08 | 3.31	75.31	103	302
	crossvit_9_240 8.2M	Mixed	1.63 | 1.66	56.60	187	247
		NonDP	1.45 | 1.63	59.92	212	168
	crossvit_base_240 104.0M	Mixed	7.34 | 7.60	69.09	48	1221
		NonDP	6.50 | 6.85	80.85	53	730
	crossvit_small_240 26.3M	Mixed	3.21 | 3.25	67.73	102	468
		NonDP	2.88 | 3.13	75.37	110	285
	crossvit_tiny_240 6.8M	Mixed	1.47 | 1.65	54.31	204	239
		NonDP	1.34 | 1.62	59.03	223	164
	deit_base_patch16_224 85.8M	Mixed	4.15 | 4.47	70.04	82	920
		NonDP	3.76 | 4.06	85.70	86	549
	deit_small_patch16_224 21.7M	Mixed	1.88 | 1.97	62.73	170	332
		NonDP	1.75 | 1.85	80.35	176	206
	deit_tiny_patch16_224 5.5M	Mixed	0.89 | 1.02	49.92	346	176
		NonDP	0.84 | 1.00	65.18	360	148
	beit_large_patch16_224 303.4M	Mixed	10.72 | 11.27	77.46	27	2707
		NonDP	9.83 | 10.23	89.85	30	1592
	beit_base_patch16_224 85.8M	Mixed	3.84 | 4.06	61.36	86	809
		NonDP	3.56 | 3.79	83.00	91	505
	convit_base 85.8M	Mixed	6.46 | 6.77	71.61	47	1136
		NonDP	5.93 | 6.31	85.49	50	682
	convit_small 27.3M	Mixed	3.45 | 3.65	65.98	86	525
		NonDP	3.22 | 3.51	82.85	90	310
	convit_tiny 21.7M	Mixed	1.54 | 1.63	51.72	194	235
		NonDP	1.47 | 1.57	70.48	200	160
	vit_base_patch16_224 85.9M	Mixed	4.80 | 5.13	65.78*	82	917
		NonDP	4.40 | 4.91	90.21*	86	550
	vit_small_patch16_224 21.7M	Mixed	2.05 | 2.13	73.34	170	330
		NonDP	1.92 | 2.04	86.93	176	204
	vit_tiny_patch16_224 5.5M	Mixed	0.93 | 1.04	49.54	346	176
		NonDP	0.89 | 1.02	72.30	360	156

Table 9: Performance of selected ViTs on CIFAR10 under $\epsilon = 2$. Here batch size 1000, physical batch size 20, except for max (physical) batch size and min time/epoch (using max batch size). There are two types of memory: active memory (left) and total memory (right). All ViTs use DP learning rate $2e-3$ and non-DP learning rate $2e-4$ by default, except the ViT base that uses half the learning rate, since the default learning rate gives $< 50\%$ accuracy.

E Demo of privacy engine

We demonstrate how to use our privacy engine to train any vision models differentially privately. We term our library as *private_vision*, which is significantly based on the *private_transformers* library [32] at `https://github.com/lxuechen/private-transformers`. We provide two modes through the 'mode' argument in the privacy engine: 'ghost-mixed' for the mixed ghost clipping, and 'ghost' for the ghost clipping.

```
import torchvision, torch, timm, opacus
from private_vision import PrivacyEngine

model = torchvision.models.resnet18()
# model = timm.create_model('crossvit_small_240', pretrained= True)

model=opacus.validators.ModuleValidator.fix(model)
# replace BatchNorm by GroupNorm or LayerNorm

optimizer = torch.optim.Adam(params=model.parameters(), lr=1e-4)
privacy_engine = PrivacyEngine(
    model,
    batch_size=256,
    sample_size=50000,
    epochs=3,
    max_grad_norm=0.1,
    target_epsilon=3,
    mode='ghost-mixed',
)
privacy_engine.attach(optimizer)

# Same training procedure, e.g. data loading, forward pass...
loss = F.cross_entropy(model(batch), labels, reduction="none")
# do not use loss.backward()
optimizer.step(loss=loss)
```

A special use of our privacy engine is to use the gradient accumulation. This is achieved with virtual step function.

```
import torchvision, torch, timm, opacus
from private_vision import PrivacyEngine

gradient_accumulation_steps = 10
# Batch size/physical batch size.

model = torchvision.models.resnet18()
model=opacus.validators.ModuleValidator.fix(model)
optimizer = torch.optim.Adam(model.parameters())
privacy_engine = PrivacyEngine(...)
privacy_engine.attach(optimizer)

for i, batch in enumerate(dataloader):
    loss = F.cross_entropy(model(batch), labels, reduction="none")
    if i % gradient_accumulation_steps == 0:
        optimizer.step(loss=loss)
        optimizer.zero_grad()
    else:
        optimizer.virtual_step(loss=loss)
```

F Comparison with GhostClip in [32]

We give a thorough comparison between our work and [32] (specifically codebase v0.1.0 which was the public version during the preparation of this paper), which distinguishes our contribution from a simple application of ghost clipping on convolutional layers.

1. Our contribution is on Conv1d/2d/3d layers, while [32] applies the ghost clipping on linear and embedding layers. To be specific, we show that $T_{(l)}$ is layer-dependent (which motivates the layerwise decision in (4.1)), while [32] studies sequential data and T is layer-independent. We also precisely quantifies the effect of kernel size/padding/stride on the complexity in DP training in Appendix B.
2. We provide a fine-grained complexity analysis of the clipping (see Section 4.1), while [32] shows only asymptotic complexity. For example, we show that the space complexity of ghost norm technique is $2T_{(l)}^2$ and that of per-sample gradient instantiation is $p_{(l)}d_{(l)}$. In contrast, [32] gives $O(T^2)$ and $O(pd)$, respectively. We highlight that our mixed ghost clipping, or the layerwise decision (4.1), is only made possible through our complexity analysis.
3. We additionally analyze the complexity of entire DP algorithms – e.g. Opacus, FastGradClip, and GhostClip, while [32] only focuses on the clipping part of algorithms. Thus their analysis cannot directly help us to compare different DP algorithms, which not only include the clipping but also the back-propagation. Notice that ghost clipping needs two back-propagation but Opacus only needs one back-propagation, so it is insufficient to study the complexity difference between DP algorithms by only looking at the complexity of the clipping part.
4. Our key contribution is the mixed ghost clipping, which is novel, simple, but extremely important on large image tasks. Our mixed ghost clipping is much more efficient than the vanilla ghost clipping, as visualized in Table 3, Figure 3 and especially Table 7 (on 224×224 ImageNet). As a concrete example on ImageNet, ghost clipping incurs huge memory cost on most models (e.g. ResNet18, more than 16GB and thus OOM), while mixed ghost clipping costs only 2.34GB memory for ResNet18 and 7.91GB for ResNet152, almost the same as non-DP training.