Scalable & Efficient Training of Large Convolutional Neural Networks with Differential Privacy

Zhiqi Bu[*] zbu@sas.upenn.edu
Jialin Mao[*] jmao@sas.upenn.edu
Shiyun Xu shiyunxu@sas.upenn.edu
Department of Applied Mathematics and Computational Science
University of Pennsylvania

Abstract

Large convolutional neural networks (CNN) can be difficult to train in the differentially private (DP) regime, since the optimization algorithms require a computationally expensive operation, known as the per-sample gradient clipping. We propose an efficient and scalable implementation of this clipping on convolutional layers, termed as the mixed ghost clipping, that significantly eases the private training in terms of both time and space complexities, without affecting the accuracy. The improvement in efficiency is rigorously studied through the first complexity analysis for the mixed ghost clipping and existing DP training algorithms.

Extensive experiments on vision classification tasks, with large ResNet, VGG, and Vision Transformers (ViT), demonstrate that DP training with mixed ghost clipping adds $1 \sim 10\%$ memory overhead and $< 2\times$ slowdown to the standard non-private training. Specifically, when training VGG19 on CIFAR10, the mixed ghost clipping is $3\times$ faster than state-of-the-art Opacus library with $18\times$ larger maximum batch size. To emphasize the significance of efficient DP training on convolutional layers, we achieve 96.7% accuracy on CIFAR10 and 83.0% on CIFAR100 at $\epsilon = 1$ using BEiT, while the previous best results are 94.8% and 67.4%, respectively. We open-source a privacy engine (https://github.com/woodyx218/private_vision) that implements DP training of CNN (including convolutional ViT) with a few lines of code.

1 Introduction

Deep convolutional neural networks (CNN) [17, 31] are the backbone in vision-related tasks, including image classification [29], object detection [38], image generation [20], video recognition [40], and audio classification [23]. A closer look at the dominating success of deep CNNs reveals its basis on two factors.

The first factor is the strong capacity of the convolutional neural networks, usually characterized by the enormous model size. Recent state-of-the-art progresses usually result from very large models, with millions to billions of trainable parameters. For example, ImageNet [11] accuracy grows when larger VGGs [41] (increasing 11 to 19 layers⟹69% to 74% accuracy) or ResNets [21] (increasing 18 to 152 layers⟹70% to 78% accuracy) are used [43]. Consequently, the eager to use larger models for better accuracy naturally draws people's attention to the scalability and efficiency of training.

The second factor is the availability of big data, which oftentimes contain private and sensitive information. The usage of such data demands rigorous protection against potential privacy attacks. In fact, one standard approach to guarantee the protection is by differentially private (DP) [14, 15] training of the models. Since [1], CNNs have achieved promising results under strong DP guarantee: CIFAR10 achieves 92.4% accuracy in [46] and ImageNet achieves 81.1% accuracy in [10].

[*] Equal contribution.

Preprint. Under review.

Unifying the two driving factors of CNNs leads to the DP training of large CNNs. However, the following challenges are hindering our application of large and private CNNs in practice.

Challenge 1: Time and space efficiency in DP training. DP training can be extremely inefficient in memory and speed. For example, a straightforward implementation in Tensorflow Privacy library shows that DP training can be $1000\times$ slower than the non-DP training, even on a small RNN [5]; other standard DP libraries, Opacus [50] and JAX [30, 42], which trade off memory for speed, could not fit a single datapoint into GPU on GPT2-large [33]; addtionally, $3 \sim 9\times$ slowdown of DP training has been reported in [30, 10, 42] using JAX.

The computational bottleneck comes from the per-sample gradient clipping at each iteration (see (2.1)), a necessary step in DP deep learning. I.e., denoting the loss as $\sum_i \mathcal{L}_i$, we need to clip the per-sample gradient $\{\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}}\}_i$ individually. This computational issue is even more severe when we apply a large batch size, which is necessary to achieve high accuracy of DP neural networks. In [33, 30, 34], it is shown that the optimal batch size for DP training is significantly larger than for regular training. For instance, DP ResNet18 achieves best performance on ImageNet when batch size is 64*1024 [30]; and DP ResNet152 and ViT-Large use a batch size 2^{20} in [34]. As a result, an efficient implementation of per-sample gradient clipping is much-needed to fully leverage the benefit of large batch training.

Challenge 2: Do large DP vision models necessarily harm accuracy? An upsetting observation in DP vision models is that, over certain relatively small model size, larger DP CNNs seem to underperform smaller ones. This is observed in models that are either pre-trained or trained from scratch [26, 1]. As an example of the pre-trained cases, previously state-of-the-art CIFAR10 is obtained from a small DP linear model [46], and the fine-tuned DP ResNet50 underperforms DP ResNet18 on ImageNet [30]. On the contrary, the empirical evidence in DP language models shows that larger models can consistently achieve better accuracy [33]. Interestingly, we empirically demonstrate that this trend can possibly hold in vision models as well.

1.1 Contributions

In this work, we propose new algorithms to efficiently train large-scale CNNs with DP optimizers. To be specific, our contributions are as follows.

1. We propose a novel implementation, termed as the ***mixed ghost clipping***, of the per-sample gradient clipping for 1D$\sim$3D convolutional layers. The mixed ghost clipping is the first method that can ***implement per-sample gradient clipping without per-sample gradients*** of the convolutional layers. It works with any DP optimizer and any clipping function almost as memory efficiently as in standard training, thus significantly outperforming existing implementation like Opacus [50].
2. In some tasks, mixed ghost clipping also claims supremacy on speed using a fixed batch size. The speed can be further boosted (say $1.7\times$ faster than the fastest alternative DP algorithms and only $2\times$ slower than the non-private training) when the memory saved by our method is used to fit the largest possible batch size.
3. We provide the first complexity analysis of mixed ghost clipping in comparison to other training algorithms. This analysis clearly indicates the necessity of our layerwise decision principle, without which the existing methods suffer from high memory burden.
4. Leveraging our algorithms, we can efficiently train large DP models, such as VGG, ResNet, Wide-ResNet, DenseNet, and Vision Transformer (ViT). Using DP ViTs at ImageNet scale, we are the first to train **convolutional ViTs** under DP and achieve dominating SOTA on CIFAR10/100 datasets, thus bringing new insights that larger vision models can consistently achieve better accuracy under DP.

1.2 Previous arts

The straightforward yet highly inefficient way of per-sample gradient clipping is to use batch size 1 and compute gradients with respect to each individual loss. Recently, more advanced methods have significantly boosted the efficiency by avoiding such a naive approach. The most widely applied method is implemented in the Opacus library [50], which is fast but memory costly as per-sample gradients $\boldsymbol{g}_i = \frac{\partial \mathcal{L}_i}{\partial \mathbf{W}}$ are instantiated to compute the weighted gradient $\sum_i C_i \cdot \boldsymbol{g}_i$ in (2.1). A more efficient method, FastGradClip [32], is to use a second back-propagation with weighted loss $\sum_i C_i \cdot \mathcal{L}_i$ to indirectly derive the weighted gradient.

In all above-mentioned methods and [39][2], the per-sample gradients are instantiated, whereas this can be much inefficient and not necessary according to the 'ghost clipping' technique, as to be detailed in Section 3. In other words, ghost clipping proves that the claim 'DP optimizers require access to the per-sample gradients' is wrong. Note that ghost clipping is firstly proposed by [19] for linear layers, and then extended by [33] to sequential data and embedding layers for language models. However, the ghost clipping has not been extended to convolutional layers, due to the complication of the convolution operation and the high dimension of data (text data is mostly 2D, yet image data are 3D and videos are 4D). We give more details about the difference between this work and [33] in Appendix F. In fact, we will show that even the ghost clipping alone is not satisfactory for CNNs: e.g. it cannot fit even a single datapoint into the memory on VGGs and ImageNet dataset. Therefore, we propose the mixed ghost clipping, that narrows the efficiency gap between DP training and the regular training.

2 Preliminaries

2.1 Differential privacy

Differential privacy (DP) has become the standard approach to provide privacy guarantee for modern machine learning models. The privacy level is characterized through a pair of privacy quantities (ϵ, δ), where smaller (ϵ, δ) means stronger protection.

Definition 2.1 ([15])**.** A randomized algorithm M is (ε, δ)-DP if for any neighboring datasets S, S' that differ by one arbitrary sample, and for any event E, it holds that

$$\mathbb{P}[M(S) \in E] \leqslant \mathrm{e}^{\varepsilon} \mathbb{P}\left[M\left(S'\right) \in E\right] + \delta.$$

In deep learning where the number of parameters are large, the Gaussian mechanism [15, Theorem A.1] is generally applied to achieve DP at each training iteration, i.e. we use regular optimizers on the following privatized gradient:

$$\widetilde{\boldsymbol{g}} = \sum_i C(\|\boldsymbol{g}_i\|; R) \cdot \boldsymbol{g}_i + \sigma R \cdot \mathcal{N}(0, \mathbf{I}) = \sum_i C_i \boldsymbol{g}_i + \sigma R \cdot \mathcal{N}(0, \mathbf{I}) \tag{2.1}$$

where C is any function whose output is upper bounded by $R/\|\boldsymbol{g}_i\|$ and R is known as the clipping norm. To name a few examples of C, we have the Abadi's clipping $\min(R/\|\boldsymbol{g}_i\|, 1)$ in [1], the automatic clipping $R/(\|\boldsymbol{g}_i\| + 0.01)$ in [7], and the global clipping $\mathbb{I}(\|\boldsymbol{g}_i\| < R)$in [6]. Here σ is the noise multiplier that affects the privacy loss (ϵ, δ), but R only affects the convergence, not the privacy.

In words, DP training switches from updating with $\sum_i \boldsymbol{g}_i$ to updating with the private gradient $\widetilde{\boldsymbol{g}}$: SGD with private gradient is known as DP-SGD; Adam with private gradient is known as DP-Adam.

Algorithmically speaking, the Gaussian mechanism can be decomposed into two parts: the per-sample gradient clipping and the Gaussian noise addition. From the viewpoint of computational complexity, the per-sample gradient clipping is the bottleneck, while the noise addition costs negligible overhead.

In this work, our focus is the implementation of per-sample gradient clipping (2.1). We emphasize that our implementation is only on the algorithmic level, not affecting the mathematics and thus not the performance of DP optimizers. That is, our mixed ghost clipping provides exactly the same accuracy results as Opacus, FastGradClip, etc.

2.2 Per-sample gradient for free during standard back-propagation

In DP training, the per-sample gradient is a key quantity which can be derived for free from the standard back-propagation. We briefly introduce the back-propagation on linear layers, following the

[2] The method in [39] also extends the outer product trick (similar to Opacus, see (2.4)) in [19] to convolution layer, but does not use the ghost clipping trick.

analysis from [19, 33], so as to prepare our new clipping implementation for convolutional layers. Note that the convolutional layers can be viewed as equivalent to the linear layers in Section 2.3.

Let the input of a hidden layer be $\mathbf{a} \in \mathbb{R}^{B\times\cdots\times d}$ (a.k.a. post-activation). Here $\mathbf{a}$ can be in high dimension: for sequential data such as text, $\mathbf{a} \in \mathbb{R}^{B\times T\times d}$ where T is the sequence length; for image data, $\mathbf{a} \in \mathbb{R}^{B\times H\times W\times d}$ where (H, W) is the dimension of image and d is number of channels; for 3D objects or video data, $\mathbf{a} \in \mathbb{R}^{B\times H\times W\times D\times d}$ where D is the depth or time length, respectively.

Denote the weight of a linear layer as $\mathbf{W} \in \mathbb{R}^{d\times p}$, its bias as $\mathbf{b} \in \mathbb{R}^{p}$ and its output (a.k.a. pre-activation) as $\mathbf{s} \in \mathbb{R}^{B\times\cdots\times p}$, where B is the batch size and p is the output dimension.

In the l-th layer of a neural network with L layers in total, we denote its weight, bias, input and output as $\mathbf{W}_{(l)}, \mathbf{b}_{(l)}, \mathbf{a}_{(l)}, \mathbf{s}_{(l)}$ respectively, and the activation function as ϕ. Consider

$$\mathbf{a}_{(l+1),i} = \phi(\mathbf{s}_{(l),i}) = \phi(\mathbf{a}_{(l),i}\mathbf{W}_{(l)} + \mathbf{b}_{(l)}). \tag{2.2}$$

Clearly the i-th sample's *hidden feature* $\mathbf{a}_{(l),i}$ at layer l is freely extractable during the forward pass.

Let $\mathcal{L} = \sum_{i=1}^{n} \mathcal{L}_i$ be the total loss and $\mathcal{L}_i$ be the per-sample loss with respect to the i-th sample. During a standard back-propagation, the following *partial product* is maintained,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{s}_{(l),i}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_{(L),i}} \circ \frac{\partial \mathbf{a}_{(L),i}}{\partial \mathbf{s}_{(L-1),i}} \cdot \frac{\partial \mathbf{s}_{(L-1),i}}{\partial \mathbf{a}_{(L-1),i}} \circ \cdots \frac{\partial \mathbf{a}_{(l+1),i}}{\partial \mathbf{s}_{(l),i}} = \frac{\partial \mathcal{L}}{\partial \mathbf{s}_{(l+1),i}} \mathbf{W}_{(l+1)} \circ \phi'(\mathbf{s}_{(l),i}) \tag{2.3}$$

so as to compute the standard gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{(l)}} = \sum_i \frac{\partial \mathcal{L}_i}{\partial \mathbf{W}_{(l)}}$ in (2.4). Here $\circ$ is the Hadamard product and $\cdot$ is the matrix product. Therefore, $\frac{\partial \mathcal{L}}{\partial \mathbf{s}_{(l),i}}$ is also available for free from (2.3) and extractable by Pytorch hooks, which allows us compute the per-sample gradient by

$$\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}_{(l)}} = \frac{\partial \mathcal{L}_i}{\partial \mathbf{s}_{(l),i}}^{\top} \frac{\partial \mathbf{s}_{(l),i}}{\partial \mathbf{W}_{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{s}_{(l),i}}^{\top} \mathbf{a}_{(l),i}, \quad \frac{\partial \mathcal{L}_i}{\partial \mathbf{b}_{(l)}} = \frac{\partial \mathcal{L}_i}{\partial \mathbf{s}_{(l),i}}^{\top} \frac{\partial \mathbf{s}_{(l),i}}{\partial \mathbf{b}_{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{s}_{(l),i}}^{\top} \mathbf{1}. \tag{2.4}$$

2.3 Equivalence between convolutional and linear layer

In a convolutional layer[3], the forward pass is

$$\mathbf{a}_{(l+1),i} = \phi(\mathbf{s}_{(l),i}) = \phi(F(U(\mathbf{a}_{(l),i})\mathbf{W}_{(l)} + \mathbf{b}_{(l)})) \tag{2.5}$$

in which F is the folding operation and U is the unfolding operation. To be clear, we consider a 2D convolution that $\mathbf{a}_{(l,i)} \in \mathbb{R}^{H_{\text{in}}\times W_{\text{in}}\times d_{(l)}}$ is the input of hidden feature, $(H_{\text{in}}, W_{\text{in}})$ is the input dimension, $d_{(l)}$ is the number of input channels. Then U unfolds the hidden feature from dimension $(H_{\text{in}}, W_{\text{in}}, d_{(l)})$ to $(H_{\text{out}}W_{\text{out}}, d_{(l)}k_H k_W)$, where k_H, k_W are the kernel sizes and $(H_{\text{out}}, W_{\text{out}})$ is the output dimension. After the matrix multiplication with $\mathbf{W}_{(l)} \in \mathbb{R}^{d_{(l)}k_H k_W \times p_{(l)}}$, the intermediate output $\mathbf{s}_{(l),i}$ is folded by F from dimension $(H_{\text{out}}W_{\text{out}}, p_{(l)})$ to $(H_{\text{out}}, W_{\text{out}}, p_{(l)})$.

To present concisely, we ignore the layer index l and write the per-sample gradient of weight for the convolutional layer, in analogy to the linear layer in (2.4),

$$\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial F^{-1}(\mathbf{s}_i)}^{\top} U(\mathbf{a}_i) = F^{-1}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}\right)^{\top} U(\mathbf{a}_i). \tag{2.6}$$

Here F^{-1} is the inverse operation of F and simply flattens all dimensions except the last one: from $(H_{\text{out}}, W_{\text{out}}, p_{(l)})$ to $(H_{\text{out}}W_{\text{out}}, p_{(l)})$. From (2.6), we derive the per-sample gradient norm for the convolutional layers from the same formula as in [33, Appendix F],

$$\left\|\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}}\right\|_{\text{Fro}}^2 = \text{vec}(U(\mathbf{a}_i)U(\mathbf{a}_i)^{\top})\text{vec}\left(F^{-1}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}\right) F^{-1}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}\right)^{\top}\right). \tag{2.7}$$

3 Ghost clipping for Convolutional Layers

Leveraging our derivation in (2.7), we propose the ghost clipping to compute the clipped gradient without ever generating the per-sample gradient $\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}}$. The entire procedure is comprised of the ghost norm computation and the second back-propagation, as demonstrated in Figure 1.

[3] See a detailed explanation in Appendix B for the U, F operation and the dimension formulae in convolution.

3.1 Ghost norm: computing gradient norm without the gradient

The per-sample gradient norm is required to compute the per-sample C_i in (2.1). While it is natural to instantiate the per-sample gradients and then compute their norms [39, 32, 50, 10, 34], this is not always optimal nor necessary. Instead, we can leverage (2.7), the ghost norm, to compute the per-sample gradient norm and avoid the possibly expensive per-sample gradient. Put differently, when $T = H_{\text{out}} W_{\text{out}}$ is small, the multiplication $U(\mathbf{a}_i)U(\mathbf{a}_i)^\top$ plus $F^{-1}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}\right) F^{-1}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}\right)^\top$ is cheap, but the multiplication $F^{-1}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}\right)^\top U(\mathbf{a}_i)$ is expensive. We demonstrate the ghost clipping's supremacy over complexity empirically in Table 4 and theoretically in Table 2.

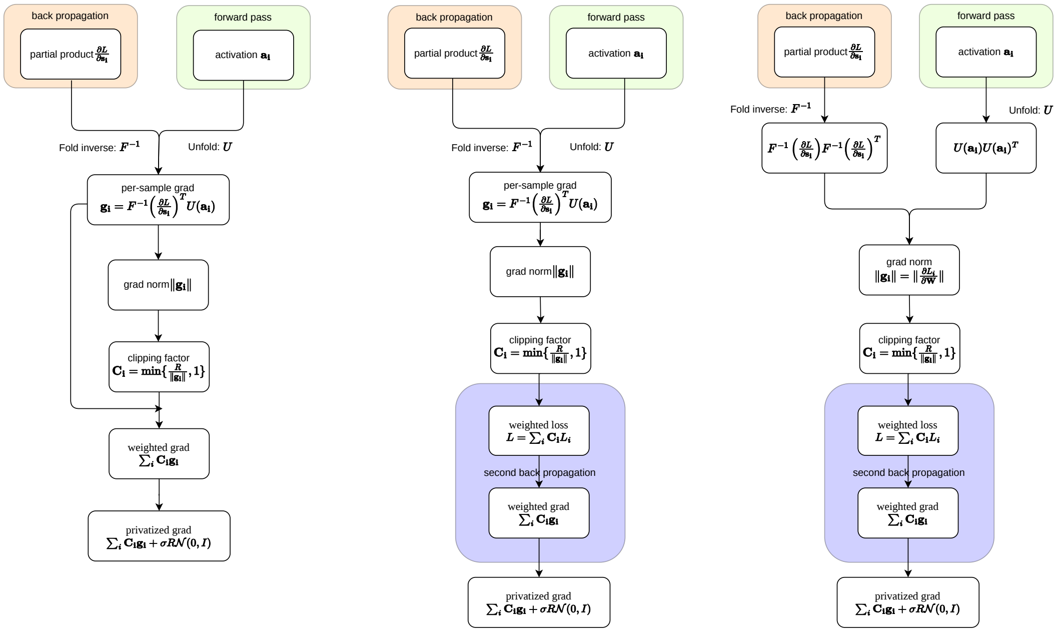
back propagation
forward pass
partial product $\frac{\partial L}{\partial \mathbf{s_i}}$
activation $\mathbf{a_i}$
Fold inverse: F^{-1}
Unfold: U
per-sample grad
$\mathbf{g_i} = F^{-1}\left(\frac{\partial L}{\partial \mathbf{s_i}}\right)^T U(\mathbf{a_i})$
grad norm $\|\mathbf{g_i}\|$
clipping factor
$\mathbf{C_i} = \min\{\frac{R}{\|\mathbf{g_i}\|}, 1\}$
weighted grad
$\sum_i \mathbf{C_i}\mathbf{g_i}$
privatized grad
$\sum_i \mathbf{C_i}\mathbf{g_i} + \sigma R \mathcal{N}(0, I)$
back propagation
forward pass
partial product $\frac{\partial L}{\partial \mathbf{s_i}}$
activation $\mathbf{a_i}$
Fold inverse: F^{-1}
Unfold: U
per-sample grad
$\mathbf{g_i} = F^{-1}\left(\frac{\partial L}{\partial \mathbf{s_i}}\right)^T U(\mathbf{a_i})$
grad norm $\|\mathbf{g_i}\|$
clipping factor
$\mathbf{C_i} = \min\{\frac{R}{\|\mathbf{g_i}\|}, 1\}$
weighted loss
$L = \sum_i \mathbf{C_i} L_i$
second back propagation
weighted grad
$\sum_i \mathbf{C_i}\mathbf{g_i}$
privatized grad
$\sum_i \mathbf{C_i}\mathbf{g_i} + \sigma R \mathcal{N}(0, I)$
back propagation
forward pass
partial product $\frac{\partial L}{\partial \mathbf{s_i}}$
activation $\mathbf{a_i}$
Fold inverse: F^{-1}
Unfold: U
$F^{-1}\left(\frac{\partial L}{\partial \mathbf{s_i}}\right) F^{-1}\left(\frac{\partial L}{\partial \mathbf{s_i}}\right)^T$
$U(\mathbf{a_i})U(\mathbf{a_i})^T$
grad norm
$\|\mathbf{g_i}\| = \|\frac{\partial L_i}{\partial \mathbf{W}}\|$
clipping factor
$\mathbf{C_i} = \min\{\frac{R}{\|\mathbf{g_i}\|}, 1\}$
weighted loss
$L = \sum_i \mathbf{C_i} L_i$
second back propagation
weighted grad
$\sum_i \mathbf{C_i}\mathbf{g_i}$
privatized grad
$\sum_i \mathbf{C_i}\mathbf{g_i} + \sigma R \mathcal{N}(0, I)$

Figure 1: Per-sample gradient clipping for convolutional layers. **Left: Opacus** = Back-propagation + Gradient instantiation + Weighted gradient. **Middle: FastGradClip** = Back-propagation + Gradient instantiation + Second back-propagation. **Right: Ghost clipping** = Back-propagation + Ghost norm + Second back-propagation. See Section 4.1 for their complexity analysis.

3.2 Second back-propagation: weighted loss leads to weight gradient

We conduct a second back-propagation with the weighted loss $\sum_i C_i \mathcal{L}_i$ to derive the weighted gradient $\sum_i C_i \boldsymbol{g}_i$ in (2.1), which costs extra time. In contrast, Opacus [50] and JAX [10, 42] generate and store the per-sample gradient $\boldsymbol{g}_i$ for all $i \in [B]$. Thus the weighted gradient is directly computable from $\boldsymbol{g}_i$ as the memory is traded off for faster computation. However, in some cases like Table 7 on ImageNet and Table 9 on CIFAR100, we can use larger batch size to compensate the slowdown of the second back-propagation.

4 Mixed Ghost Clipping: To be a ghost or not, that is the question

While the ghost norm offers the direct computation of gradient norm at the cost of an indirect computation of the weighted gradient, we will show that ghost clipping alone may not be sufficient for efficient DP training, as we demonstrate in Table 4 Figure 3, and Table 7. In Table 2, we give the first fine-grained analysis of the space and time complexity for DP training algorithms. Our analysis gives the precise condition when the per-sample gradient instantiation (adopted in Opacus [50]) is more or less efficient than our ghost norm method. To take the advantage of both methods, we propose the mixed ghost clipping method in Algorithm 1, which applies the ghost clipping or non-ghost clipping by a layerwise decision.

Algorithm 1 Mixed Ghost Clipping (single iteration)

```
Parameters: number of layers $L$, gradient clipping norm $R$.
for $l = 1, 2, \dots, L$ do
    if $2T_{(l)}^2 < p_{(l)}D_{(l)}$ then
        $\mathbf{W}_{(l)}$.ghost_norm = True                                    ▷ Forward pass
    Compute $\mathbf{a}_{(l+1),i} = \phi(F(U(\mathbf{a}_{(l),i})\mathbf{W}_{(l)} + \mathbf{b}_{(l)}))$
Compute per-sample losses $\mathcal{L}_i$.
for $l = L, L-1, \dots, 1$ do
    Compute $\frac{\partial \mathcal{L}}{\partial \mathbf{s}_{(l),i}} = \frac{\partial \mathcal{L}}{\partial \mathbf{s}_{(l+1),i}} \mathbf{W}_{(l+1)} \circ \phi'(\mathbf{s}_{(l),i})$ ▷ Mixed ghost norm in first back-propagation
    if $\mathbf{W}_{(l)}$.ghost_norm = True then
        $\| \frac{\partial \mathcal{L}_i}{\partial \mathbf{W}_{(l)}} \|_{\text{Fro}}^2 = \text{vec}(U(\mathbf{a}_{(l),i})U(\mathbf{a}_{(l),i})^\top)\text{vec}\left( \frac{\partial \mathcal{L}}{\partial F^{-1}(\mathbf{s}_{(l),i})} \frac{\partial \mathcal{L}}{\partial F^{-1}(\mathbf{s}_{(l),i})}^\top \right)$
    else
        $\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}_{(l)}} = F^{-1}(\frac{\partial \mathcal{L}}{\partial \mathbf{s}_{(l),i}})U(\mathbf{a}_{(l),i}) \longrightarrow \| \frac{\partial \mathcal{L}_i}{\partial \mathbf{W}_{(l)}} \|_{\text{Fro}}^2$
Compute per-sample gradient norm $\| \frac{\partial \mathcal{L}_i}{\partial \mathbf{W}} \|_{\text{Fro}}^2 = \sum_l \| \frac{\partial \mathcal{L}_i}{\partial \mathbf{W}_{(l)}} \|_{\text{Fro}}^2$
Compute weighted loss $\mathcal{L}_{\text{weighted}} = \sum_i C(\| \frac{\partial \mathcal{L}_i}{\partial \mathbf{W}} \|_{\text{Fro}}; R) \cdot \mathcal{L}_i$
Second back-propagation with $\mathcal{L}_{\text{weighted}}$ to generate $\sum_i C_i \frac{\partial \mathcal{L}_i}{\partial \mathbf{W}}$
```

We highlight that the key reason supporting the success of mixed ghost clipping method is its layerwise adaptivity to the dimension parameters, $(p_{(l)}, d_{(l)}, T_{(l)}, k_H, k_W)$, which vary largely across different layers (see Figure 2). The variance results from the fact that images are non-sequential data, and that the convolution and pooling can change the size ($T = H_{\text{out}} W_{\text{out}}$) of hidden features drastically.

In the next two sections, we will analyze rigorously the time and memory complexities of the regular training and the DP training, using ghost or non-ghost clippings.

Remark 4.1. In Algorithm 1, we present the mixed ghost clipping that prioritizes the space complexity by (4.1). We also derive and implement a speed-priority version by comparing the time complexity of ghost norm and gradient instantiation in Table 1. However, the efficiency difference is empirically insignificant and implied by Table 1.

4.1 Complexity analysis

We now break each clipping method into operation modules and analyze their complexities. A similar but coarse analysis from [33] only claims, on sequential layers, $O(BT^2)$ space complexity with ghost clipping and $O(Bpd)$ without ghost clipping. The time complexity and/or convolutional layers are not analyzed until this work.

Complexity	Forward pass	Back-propagation	Ghost norm	Grad instantiation	Weighted grad
Time	$2BTpD$	$2BTD(2p+1)$	$2BT^2(D+p+1) - B$	$2B(T+1)pD$	$2BpD$
Space	—	$BTp + 2BTD + pD$	$B(2T^2+1)$	$B(pD+1)$	0

Table 1: Complexities of operation modules in per-sample gradient clipping methods, contributed by a single 2D convolutional layer.

Here B is the batch size, $D = dk_H k_W$ where d is the number of input channels, k is the kernel sizes, p is the number of output channels, and $T = H_{\text{out}} W_{\text{out}}$. We leave the detailed complexity computation in Appendix C. Leveraging Table 1, we give the complexities of different clipping algorithms in Table 2.

4.2 Layerwise decision in mixed clipping

From the space complexity in Table 2, we derive the layerwise decision that selects the more memory efficient of FastGradClip (gradient instantiation) and ghost clipping (ghost norm):

$$\text{Choose ghost norm over per-sample gradient instantiation if } 2T^2 < pD = pdk_H k_W. \tag{4.1}$$

Complexity	Time	Space
Standard (non-DP)	$6BTpD$	$pD + B(Tp + 2TD)$
Opacus [50]	$8BTpD$	$B(pD + Tp + 2TD)$*
FastGradClip [32]	$10BTpD$	$B(pD + Tp + 2TD)$
Ghost clipping (ours)	$10BTpD + 2BT^2(p + D)$	$B(2T^2 + Tp + 2TD)$
Mixed ghost clipping (ours)	$\approx 10BTpD$	$B(\min(2T^2, pD) + Tp + 2TD)$

Table 2: Complexity of DP algorithms with per-sample gradient clipping, on a single 2D convolution layer. Only highest order terms are listed. * indicates that Opacus stores the per-sample gradients of all layers, thus a per-layer space complexity does not accurately characterize its memory burden, since other methods only store the intermediate variables one layer at a time. The mixed ghost clipping's time complexity is between FastGradClip and ghost clipping, depending on which of $(2T^2, pD)$ is smaller.

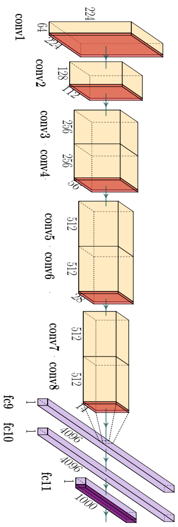

Figure 2: VGG-11 architecture on ImageNet (224 × 224).

	Ghost norm	Non-ghost norm
Space complexity	$2T_{(l)}^2 = 2H_{\text{out}}^2 W_{\text{out}}^2$	$p_{(l)} d_{(l)} k_H k_W$
conv1	5.0×10^9	$\mathbf{1.7 \times 10^3}$
conv2	3.0×10^8	$\mathbf{7.3 \times 10^4}$
conv3	2.0×10^7	$\mathbf{2.9 \times 10^5}$
conv4	2.0×10^7	$\mathbf{5.8 \times 10^5}$
conv5	1.2×10^6	$\mathbf{1.1 \times 10^6}$
conv6	$\mathbf{1.2 \times 10^6}$	2.3×10^6
conv7	$\mathbf{7.6 \times 10^4}$	2.3×10^6
conv8	$\mathbf{7.6 \times 10^4}$	2.3×10^6
fc9	$\mathbf{2}$	1.0×10^8
fc10	$\mathbf{2}$	1.6×10^7
fc11	$\mathbf{2}$	4.1×10^6
Total complexity	5.34×10^9	1.33×10^8
Mixed ghost norm	3.40×10^6	

Table 3: Layerwise decision of mixed ghost clipping on VGG-11. Green background indicates being selected.

Therefore, our mixed ghost clipping is a mixup of FastGradClip and the ghost clipping (c.f. Figure 1). We note that the decision by the mixed ghost clipping (4.1) depends on different dimensions: the ghost clipping depends on the size of hidden features (height H and width W) which in turn depends on kernel size, stride, dilation and padding (see Appendix B for introduction of convolution), while only the non-ghost clipping depends on the number of channels. In ResNet and VGG, the hidden feature size decreases as layer depth increases, due to the shrinkage from the convolution and pooling operation; on the opposite, the number of channels increases in deeper layers.

Remark 4.2 (Ghost clipping favors bottom layers). As a consequence of decreasing hidden feature size and increasing number of channels, there exists a depth threshold beyond which the ghost clipping is preferred in bottom layers, where the save in complexity is substantial. In Figure 2 and Table 3, as the layer of VGG 11 goes deeper, the hidden feature size shrinks from $224 \to 112 \to \cdots \to 14$ and the number of channels increases from $3 \to 64 \to \cdots \to 512$.

5 Performance

We compare our ghost clipping and mixed ghost clipping methods to state-of-the-art clipping algorithms, namely Opacus [50] and FastGradClip [32], which are implemented in Pytorch. We are aware of but will not compare to implementations of these two algorithms in JAX [4], e.g. [30, 42, 10], so as to only focus on the algorithms rather than the operation framework. All experiments run on one Tesla V100 GPU (16GB RAM).

We highlight that switching from the regular training to DP training only needs a few lines of code using our privacy engine (see Appendix E). For CNNs, we use models from `https://github.com/kuangliu/pytorch-cifar` on CIFAR10 (32 × 32) [28] and models from Torchvision [43] on ImageNet (224 × 224) [11]. For ViTs, regardless of datasets, we resize images to 224 × 224 and use models from PyTorch Image Models (TIMM) [47].

5.1 Time and memory efficiency (fixed batch size)

We first measure the time and space complexities when the physical batch size is fixed. Here we define the physical batch size (or the virtual batch size) as the number of samples actually fed into the memory, which is different from the logical batch size. For example, if we train with batch size 1000 but can only feed 40 samples to GPU at one time, we back-propagate 25 times before updating the weights for 1 time. This technique is known as the gradient accumulation and is widely applied in large batch training, which particularly benefits the accuracy of DP training [33, 30, 10, 34].

Dataset	Model & # Params	Package	Time (sec) / Epoch	Active Memory (GB)
CIFAR10	CNN [46, 36] 0.551M	Opacus	12	1.37
		FastGradClip	11	0.94
		Ghost (ours)	11	2.47
		Mixed (ours)	7	0.79
		Non-DP	5	0.66
	ResNet 18 / 34 / 50 11M / 21M / 23.5M	Opacus	OOM / OOM / OOM	OOM / OOM / OOM
		FastGradClip	45 / 80 / OOM	6.32 / 7.65 / OOM
		Ghost (ours)	59 / 98 / 158	4.00 / 4.90 / 9.62
		Mixed (ours)	37 / 66 / 119	3.31 / 4.13 / 9.62
		Non-DP	14 / 24 / 49	3.30 / 4.12 / 9.56
	VGG 11 / 13 / 16 9M / 9.4M / 14.7M	Opacus	OOM / OOM / OOM	OOM / OOM / OOM
		FastGradClip	18 / 25 / 33	5.17 / 5.45 / 5.61
		Ghost (ours)	14 / 25 / 29	2.58 / 3.30 / 3.41
		Mixed (ours)	13 / 18 / 23	2.58 / 2.76 / 2.84
		Non-DP	5 / 6 / 8	2.54 / 2.73 / 2.81

Table 4: Time and memory of selected models on CIFAR10 (32 × 32), with physical batch size 256. Additional models are in Table 6. Out of memory (OOM) means the total memory exceeds 16GB.

From Table 4 and the extended Table 6, we see a clear advantage of mixed ghost clipping: our clipping only incurs $\leq 1\%$ memory overhead than the regular training, and is the fastest DP training algorithm. In contrast on ResNet18, Opacus uses 5× memory, and FastGradClip uses 2× memory. Even the ghost clipping uses 1.2× memory of regular training, while being slower than both Opacus and FastGradClip.

Similar phenomenon is observed on ImageNet in Table 7: at physical batch size 25, while the mixed ghost clipping works efficiently, we observe that the ghost clipping and Opacus incur heavy memory burden that leads to OOM error on all VGGs and wide ResNets. In fact, the ghost clipping fails in memory on all models except the small AlexNet [27].

5.2 Maximum batch size and throughput

Importantly, the speed efficiency in Table 4 can be further boosted, if we use up the saved memory to increase the batch size. To stress test the maximum physical batch size and the throughput of each clipping method, we train ResNet [21], VGG [41], MobileNet[24], ResNeXt [48],AlexNet[27],Wide-ResNet[52], DenseNet[25] and ViTs on CIFAR10 and ImageNet, as summarized partially in Figure 3 and in Table 7, respectively. For example, on VGG19 and CIFAR10, the mixed ghost clipping has a maximum batch size 18× bigger (thus 3× faster) than Opacus, 3× bigger (thus 1.7× faster) than FastGradClip, and 2× bigger (thus 1.3× faster) than the ghost clipping. Similarly, on Wide-ResNet50 and ImageNet, the mixed ghost clipping has a maximum batch size 5× bigger than Opacus, 11× bigger than the ghost clipping, and $< 0.3\%$ more memory costly than the non-private training.

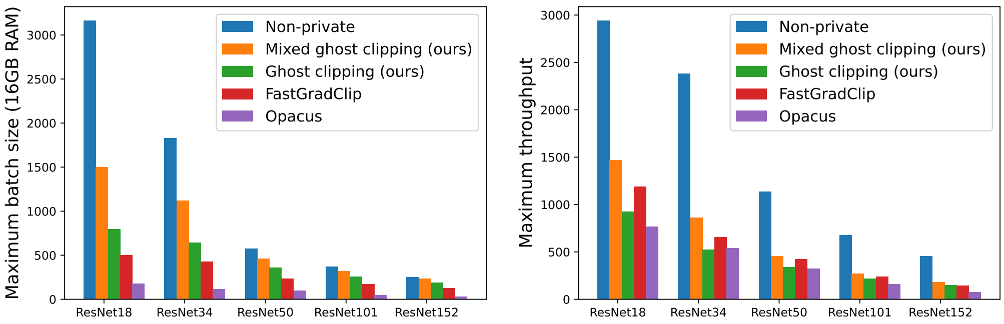

Figure 3: Memory (left) and speed (right) comparison of DP clipping algorithms on CIFAR10.

5.3 Vision transformers with convolution on ImageNet scale

In addition to training large-scale CNNs such as ResNet152, we apply our mixed ghost clipping to train ViTs, which substantially outperform existing SOTA on CIFAR10 and CIFAR100. Notice that the ViTs are pretrained on ImageNet scale, by which we resize CIFAR images (from 32×32 pixels to 224×224 pixels).

It is worth mentioning that ViT [13] is originally proposed as a substitute of CNN. Hence it and many variants do not contain convolutional layers. Here we specifically consider the convolutional ViTs, including ScalableViT[49], XCiT[2], Visformer[9], CrossVit[8], NesT[54], CaiT[45], DeiT[44], BEiT[3], PiT[22], and ConViT[16]. Performance of these ViTs on CIFAR10 and CIFAR100 are listed in Appendix D for a single-epoch DP training and several ViTs already beat previous SOTA, even though we do not apply additional techniques as in [10, 34] (e.g. learning rate schedule or random data augmentation).

	ε	CIFAR-10	CIFAR-100
Yu et al. [51] (ImageNet1k)	1	94.3%	–
	2	94.8%	–
Tramer et al. [46] (ImageNet1k)	2	92.7%	–
De et al. [10]	1	94.8%	67.4%
	2	95.4%	74.7%
(ImageNet1k)	4	96.1%	79.2%
	8	96.6%	81.8%
Our CrossViT base (104M params)	1	95.5%	71.9%
	2	96.1%	74.3%
(ImageNet1k)	4	96.2%	76.7%
	8	96.5%	77.8%
Our BEiT large (303M params)	1	96.7%	83.0%
	2	97.1%	86.2%
(ImageNet21k)	4	97.2%	87.7%
	8	97.4%	88.4%

Table 5: CIFAR-10 and CIFAR-100 (resized to 224×224) test accuracy when fine-tuning with DP-Adam. We train CrossViT base for 5 epochs, learning rate 0.002. We train BEiT large for 3 epochs and learning rate 0.001. Here batch size is 1000 and clipping norm is 0.1. '( )' indicates the pretrained datasets.

By training multiple epochs with best performing ViTs in Table 8 and Table 9, we achieve new SOTA under DP in Table 5, with substantial improvement especially for strong privacy guarantee (e.g. $\epsilon < 2$). Our DP training is at most $2\times$ slower and 10% more memory expensive than the non-private training, even on BEiT large, thus significantly improving the $9\times$ slowdown reported in [10].

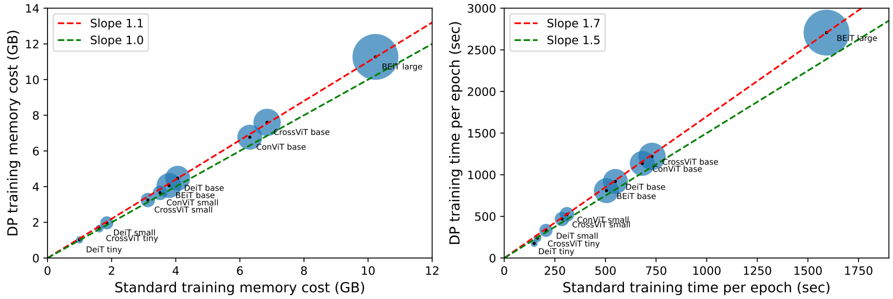

Figure 4: Memory (left) and speed (right) comparison of DP and non-DP training on CIFAR100 (resized to 224×224, ImageNet scale) with convolutional ViTs. Note that CIFAR10 has an almost identical pattern.

6 Discussion

We have shown that DP training can be efficient for large CNNs and ViTs with convolutional layers. For example, in comparison to non-private training, we reduce the training time to $< 2\times$ and the memory overhead to $< 10\%$ for all vision models examined (up to 303.4 million parameters), including BEiT that achieves SOTA accuracy on CIFAR100 ($+15.6\%$ absolutely at $\epsilon = 1$). We have observed that for many tasks and large CNNs and ViTs, the memory overhead of DP training can be as low as less than 1%.

We emphasize that our DP training only improves the efficiency, not affecting the accuracy, and therefore is generally applicable, e.g. with SOTA data augmentations in [10]. With efficient training algorithms, we look forward to applying DP CNNs to generation tasks [20] , seq-to-seq learning [18], text classification [53], reinforcement learning [35], and multi-modal learning. Further reducing time complexity and prioritizing speed in DP training is another future direction.

In particular, our layerwise decision principle in (4.1) highlights the advantages of ghost clipping when $T = HW$ is small. This advocates the use of large kernel sizes in DP learning, as they shrink the hidden feature aggressively, and have been shown to be highly accurate on non-private tasks [21, 12, 37].

References

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

[2] A. Ali, H. Touvron, M. Caron, P. Bojanowski, M. Douze, A. Joulin, I. Laptev, N. Neverova, G. Synnaeve, J. Verbeek, et al. Xcit: Cross-covariance image transformers. *Advances in neural information processing systems*, 34, 2021.

[3] H. Bao, L. Dong, S. Piao, and F. Wei. Beit: Bert pre-training of image transformers. In *International Conference on Learning Representations*, 2021.

[4] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[5] Z. Bu, S. Gopi, J. Kulkarni, Y. T. Lee, H. Shen, and U. Tantipongpipat. Fast and memory efficient differentially private-sgd via jl projections. *Advances in Neural Information Processing Systems*, 34, 2021.

[6] Z. Bu, H. Wang, Q. Long, and W. J. Su. On the convergence and calibration of deep learning with differential privacy. *arXiv preprint arXiv:2106.07830*, 2021.

[7] Z. Bu, Y.-X. Wang, S. Zha, and G. Karypis. Automatic clipping: Differentially private deep learning made easier and stronger. *arXiv preprint arXiv:2206.07136*, 2022.

[8] C.-F. R. Chen, Q. Fan, and R. Panda. Crossvit: Cross-attention multi-scale vision transformer for image classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 357–366, 2021.

[9] Z. Chen, L. Xie, J. Niu, X. Liu, L. Wei, and Q. Tian. Visformer: The vision-friendly transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 589–598, 2021.

[10] S. De, L. Berrada, J. Hayes, S. L. Smith, and B. Balle. Unlocking high-accuracy differentially private image classification through scale. *arXiv preprint arXiv:2204.13650*, 2022.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[12] X. Ding, X. Zhang, J. Han, and G. Ding. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11963–11975, 2022.

[13] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

[14] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

[15] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

[16] S. d'Ascoli, H. Touvron, M. L. Leavitt, A. S. Morcos, G. Biroli, and L. Sagun. Convit: Improving vision transformers with soft convolutional inductive biases. In *International Conference on Machine Learning*, pages 2286–2296. PMLR, 2021.

[17] K. Fukushima and S. Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[18] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, pages 1243–1252. PMLR, 2017.

[19] I. Goodfellow. Efficient per-example gradient computations. *arXiv preprint arXiv:1510.01799*, 2015.

[20] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[21] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[22] B. Heo, S. Yun, D. Han, S. Chun, J. Choe, and S. J. Oh. Rethinking spatial dimensions of vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11936–11945, 2021.

[23] S. Hershey, S. Chaudhuri, D. P. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, et al. Cnn architectures for large-scale audio classification. In *2017 ieee international conference on acoustics, speech and signal processing (icassp)*, pages 131–135. IEEE, 2017.

[24] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[25] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[26] H. Klause, A. Ziller, D. Rueckert, K. Hammernik, and G. Kaissis. Differentially private training of residual networks with scale normalisation. *arXiv preprint arXiv:2203.00324*, 2022.

[27] A. Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

[28] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[30] A. Kurakin, S. Chien, S. Song, R. Geambasu, A. Terzis, and A. Thakurta. Toward training at imagenet scale with differential privacy. *arXiv preprint arXiv:2201.12328*, 2022.

[31] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[32] J. Lee and D. Kifer. Scaling up differentially private deep learning with fast per-example gradient clipping. *arXiv preprint arXiv:2009.03106*, 2020.

[33] X. Li, F. Tramer, P. Liang, and T. Hashimoto. Large language models can be strong differentially private learners. In *International Conference on Learning Representations*, 2021.

[34] H. Mehta, A. Thakurta, A. Kurakin, and A. Cutkosky. Large scale transfer learning for differentially private image classification. *arXiv preprint arXiv:2205.02973*, 2022.

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[36] N. Papernot, A. Thakurta, S. Song, S. Chien, and Ú. Erlingsson. Tempered sigmoid activations for deep learning with differential privacy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9312–9321, 2021.

[37] C. Peng, X. Zhang, G. Yu, G. Luo, and J. Sun. Large kernel matters–improve semantic segmentation by global convolutional network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4353–4361, 2017.

[38] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[39] G. Rochette, A. Manoel, and E. W. Tramel. Efficient per-example gradient computations in convolutional neural networks. *arXiv preprint arXiv:1912.06015*, 2019.

[40] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. *Advances in neural information processing systems*, 27, 2014.

[41] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[42] P. Subramani, N. Vadivelu, and G. Kamath. Enabling fast differentially private sgd via just-in-time compilation and vectorization. *Advances in Neural Information Processing Systems*, 34, 2021.

[43] Torchvision. Pytorch vision description. `https://pytorch.org/hub/pytorch_vision_vgg/` and `https://pytorch.org/hub/pytorch_vision_resnet/`.

[44] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

[45] H. Touvron, M. Cord, A. Sablayrolles, G. Synnaeve, and H. Jégou. Going deeper with image transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 32–42, 2021.

[46] F. Tramer and D. Boneh. Differentially private learning needs better features (or much more data). In *International Conference on Learning Representations*, 2020.

[47] R. Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

[48] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[49] R. Yang, H. Ma, J. Wu, Y. Tang, X. Xiao, M. Zheng, and X. Li. Scalablevit: Rethinking the context-oriented generalization of vision transformer. *arXiv preprint arXiv:2203.10790*, 2022.

[50] A. Yousefpour, I. Shilov, A. Sablayrolles, D. Testuggine, K. Prasad, M. Malek, J. Nguyen, S. Ghosh, A. Bharadwaj, J. Zhao, et al. Opacus: User-friendly differential privacy library in pytorch. In *NeurIPS 2021 Workshop Privacy in Machine Learning*, 2021.

[51] D. Yu, H. Zhang, W. Chen, and T.-Y. Liu. Do not let privacy overbill utility: Gradient embedding perturbation for private learning. In *International Conference on Learning Representations*, 2020.

[52] S. Zagoruyko and N. Komodakis. Wide residual networks. In *British Machine Vision Conference 2016*. British Machine Vision Association, 2016.

[53] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

[54] Z. Zhang, H. Zhang, L. Zhao, T. Chen, S. Arik, and T. Pfister. Nested hierarchical transformer: Towards accurate, data-efficient and interpretable visual understanding. 2022.

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 [43] 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`.

$$\begin{array}{ccccccc}
\multicolumn{7}{c}{\overbrace{\hspace{26em}}^{\text{Conv2d}(\mathbf{a}_i)}} \\
\mathbf{a}_i & \longrightarrow & U(\mathbf{a}_i) & \longrightarrow & U(\mathbf{a}_i)\mathbf{W} + \mathbf{b} & \longrightarrow & F(U(\mathbf{a}_i)\mathbf{W} + \mathbf{b}) \\
(H_{in}, W_{in}, d) & \longrightarrow & (H_{out}, W_{out}, dk_H k_W) & \longrightarrow & (H_{out} W_{out}, p) & \longrightarrow & (H_{out}, W_{out}, p)
\end{array}$$

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 $\|\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}}\|^2_{\text{Fro}}$.

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

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

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 $\|\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}}\|^2_{\text{Fro}}$.

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 [47]. 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 [10]. We describe the models as their configuration argument in [47].

Dataset	Model & # Param	Package	Memory (GB)	Accuracy (%)	Max	Min
					Batch Size	Time/Epoch
CIFAR10	crossvit_18_240	Mixed	4.49 | 5.00	95.08	72	700
	42.6M	NonDP	4.07 | 4.87	97.11	78	420
	crossvit_15_240	Mixed	3.42 | 3.48	93.97	95	507
	27.0M	NonDP	3.08 | 3.31	96.66	103	303
	crossvit_9_240	Mixed	1.63 | 1.66	88.67	187	248
	8.2M	NonDP	1.45 | 1.63	93.33	212	180
	crossvit_base_240	Mixed	7.33 | 7.49	95.22	48	1228
	103.9M	NonDP	6.49 | 6.85	97.37	53	731
	crossvit_small_240	Mixed	3.21 | 3.25	94.05	102	463
	26.3M	NonDP	2.88 | 3.13	96.17	110	287
	crossvit_tiny_240	Mixed	1.47 | 1.64	88.31	204	233
	6.7M	NonDP	1.34 | 1.62	93.12	223	167
	deit_base_patch16_224	Mixed	4.15 | 4.47	94.56	82	882
	85.8M	NonDP	3.76 | 4.06	97.14	86	513
	deit_small_patch16_224	Mixed	1.88 | 1.97	91.16	170	330
	21.7M	NonDP	1.75 | 1.85	96.35	176	204
	deit_tiny_patch16_224	Mixed	0.89 | 1.02	84.22	346	175
	5.5M	NonDP	0.84 | 1.00	93.46	360	154
	beit_large_patch16_224	Mixed	10.72 | 11.27	93.94	27	2703
	303.4M	NonDP	9.83 | 10.23	97.80	30	1597
	beit_base_patch16_224	Mixed	3.84 | 4.06	91.68	86	805
	85.8M	NonDP	3.56 | 3.79	97.14	91	506
	convit_base	Mixed	6.46 | 6.77	94.76	47	1115
	85.8M	NonDP	5.93 | 6.31	96.82	50	649
	convit_small	Mixed	3.45 | 3.65	93.16	86	513
	27.3M	NonDP	3.22 | 3.51	97.07	90	311
	convit_tiny	Mixed	1.54 | 1.63	86.56	199	236
	5.5M	NonDP	1.47 | 1.57	94.51	200	157
	vit_base_patch16_224	Mixed	4.80 | 5.13	94.40*	82	926
	85.8M	NonDP	4.40 | 4.91	97.43*	86	550
	vit_small_patch16_224	Mixed	2.04 | 2.13	92.77	170	334
	21.7M	NonDP	1.92 | 2.04	97.69	176	204
	vit_tiny_patch16_224	Mixed	0.93 | 1.04	87.56	346	179
	5.5M	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	Min
					Batch Size	Time/Epoch
CIFAR100	crossvit_18_240	Mixed	4.49 | 5.00	71.78	72	696
	42.7M	NonDP	4.07 | 4.87	79.46	78	421
	crossvit_15_240	Mixed	3.42 | 3.50	67.21	95	495
	27.0M	NonDP	3.08 | 3.31	75.31	103	302
	crossvit_9_240	Mixed	1.63 | 1.66	56.60	187	247
	8.2M	NonDP	1.45 | 1.63	59.92	212	168
	crossvit_base_240	Mixed	7.34 | 7.60	69.09	48	1221
	104.0M	NonDP	6.50 | 6.85	80.85	53	730
	crossvit_small_240	Mixed	3.21 | 3.25	67.73	102	468
	26.3M	NonDP	2.88 | 3.13	75.37	110	285
	crossvit_tiny_240	Mixed	1.47 | 1.65	54.31	204	239
	6.8M	NonDP	1.34 | 1.62	59.03	223	164
	deit_base_patch16_224	Mixed	4.15 | 4.47	70.04	82	920
	85.8M	NonDP	3.76 | 4.06	85.70	86	549
	deit_small_patch16_224	Mixed	1.88 | 1.97	62.73	170	332
	21.7M	NonDP	1.75 | 1.85	80.35	176	206
	deit_tiny_patch16_224	Mixed	0.89 | 1.02	49.92	346	176
	5.5M	NonDP	0.84 | 1.00	65.18	360	148
	beit_large_patch16_224	Mixed	10.72 | 11.27	77.46	27	2707
	303.4M	NonDP	9.83 | 10.23	89.85	30	1592
	beit_base_patch16_224	Mixed	3.84 | 4.06	61.36	86	809
	85.8M	NonDP	3.56 | 3.79	83.00	91	505
	convit_base	Mixed	6.46 | 6.77	71.61	47	1136
	85.8M	NonDP	5.93 | 6.31	85.49	50	682
	convit_small	Mixed	3.45 | 3.65	65.98	86	525
	27.3M	NonDP	3.22 | 3.51	82.85	90	310
	convit_tiny	Mixed	1.54 | 1.63	51.72	194	235
	21.7M	NonDP	1.47 | 1.57	70.48	200	160
	vit_base_patch16_224	Mixed	4.80 | 5.13	65.78*	82	917
	85.9M	NonDP	4.40 | 4.91	90.21*	86	550
	vit_small_patch16_224	Mixed	2.05 | 2.13	73.34	170	330
	21.7M	NonDP	1.92 | 2.04	86.93	176	204
	vit_tiny_patch16_224	Mixed	0.93 | 1.04	49.54	346	176
	5.5M	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 [33] 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 [33]

We give a thorough comparison between our work and [33] (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 [33] 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 [33] 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 [33] 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, [33] 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 [33] 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.