ZERO REDUNDANCY DISTRIBUTED LEARNING WITH DIFFERENTIAL PRIVACY

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep learning with large models have achieved amazing success in a wide range of domains, but the optimization on billions of parameters is challenging in terms of the training speed, memory cost, and communication efficiency, especially under the differential privacy (DP) regime. On the one hand, DP optimization has comparable efficiency to the standard non-private optimization on a single device, but existing DP distributed learning (such as data/pipeline parallel) has significant limitations in efficiency. On the other hand, the Zero Redundancy Optimizer (ZeRO) is a state-of-the-art solution to optimize memory and improve the training efficiency on large models under the standard regime, but it encounters technical challenges to work compatibly with DP. In this work, we develop a new systematic solution, DP-ZeRO, to scale up the model size and obtain almost the same computation and communication efficiency as the standard distributed learning, in both the full and mixed precision. Our DP-ZeRO, like the standard ZeRO, has the potential to train models with arbitrary size and is evaluated on DP models that has the world's largest number of trainable parameters.[1]

1 INTRODUCTION

Recent advances in differentially private (**DP**) deep learning has overcome the computational difficulty and allowed the optimization to be almost as efficient as the standard non-DP optimization on a single device (8; 24). It is high time to apply DP in distributed learning, where new challenges such as the communication efficiency and compatibility with mixed precision emerge. In this work, we equip state-of-the-art distributed learning solution, Zero Redundancy Optimizer (21) (**ZeRO**), with DP, and tackle these technical challenges. We term our solution DP-ZeRO and carefully analyze its performance on large model training, as well as the difference between DP optimization on single device and in distributed learning.

Solution	Parallelism	Model partition	Platform	Remark
Opacus DDP	Data	No	Pytorch	cannot fit large model; memory inefficient (due to per-sample gradients)
(9; 18)	Data	No	JAX	cannot fit large model; time inefficient (e.g. $2-9\times$ slowdown)
(14)	Pipeline	Yes	Pytorch	pipeline parallelism has bubble that wastes GPU time
Ours	Data(&Model)	Yes	Pytorch	time & memory & communication efficient

Table 1: DP distributed learning; more to be added

2 PRELIMINARY

2.1 DIFFERENTIAL PRIVACY

DP provides a strong privacy guarantee on a per-sample basis, making it difficult to extract any information about an arbitrary training sample. We study (ϵ, δ)-DP in Definition 2.1, with smaller (ϵ, δ) means lower privacy risk.

Definition 2.1 ((11))**.** A randomized algorithm M is (ε, δ)-DP if, for any two neighboring datasets S, S' that differ by one sample and for any event E, we have $\mathbb{P}[M(S) \in E] \leqslant \mathrm{e}^{\varepsilon} \mathbb{P}\left[M\left(S'\right) \in E\right]+\delta$.

[1]This paper is part of an upcoming full version.

2.1.1 Private gradient descent

In deep learning, DP can be realized by applying standard optimizers such as SGD, Adam, LAMB, and FedAvg on the private gradient (1), which is the noisy sum of the clipped per-sample gradients:

$$\text{private gradient: } G_{[m]} := \sum_i C_i(R_m)\boldsymbol{g}_{[m],i} + \sigma_{\text{DP}}\|[R_1, \cdots, R_M]\| \cdot \mathcal{N}(0, \mathbf{I}). \tag{1}$$

Note that the privacy guarantee is proportional to the noise multiplier σ_{DP}, according to the privacy accounting theory (1; 3; 10; 27; 13; 17), with $\sigma_{\text{DP}} = 0$ leading to $\epsilon = \infty$ (non-private).

Group-wise gradient clipping The trainable parameters and their gradients are mathematically partitioned into M groups, e.g. in all-layer clipping, all parameters form one group ($M = 1$); in layer-wise clipping (20), each layer's parameters form a group ($M = \#$ of layers). Other examples include per-device (14), parameter-wise (23), and uniform grouping. We denote $\boldsymbol{g}_{[m],i}$ as the i-th per-sample gradient of the m-th group's parameters.

Gradient clipping Per-sample gradient clipping $\boldsymbol{g}_i \to C_i\boldsymbol{g}_i$ requires the clipping factor $C_i = C(\|\boldsymbol{g}_i\|)$. This factor is determined by (I) the clipping function, e.g. regular clipping $C_i = \min(R/\|\boldsymbol{g}_i\|, 1)$ (1), automatic clipping $C_i = 1/(\|\boldsymbol{g}_i\| + 0.01)$ (6; 22), and global clipping $C_i = \mathbb{I}(\|\boldsymbol{g}_i\| < R)$ (5); (II) the clipping threshold R, which is usually data-independent if defined prior to the training, or data-driven at the cost of extra hyperparameters and privacy spending (14; 2). For example, in the automatic clipping, one can set $R_m = 1/\sqrt{M}$ and $\|[R_1, \cdots, R_M]\| = 1$.

Per-sample gradient norm To clip the per-sample gradients, it is necessary to compute their norms, which is made efficient via the ghost clipping trick (12; 19; 4). In fact, the per-sample gradient norms can be computed without instantiating the computationally expensive gradients, thus having small overhead compared to the standard non-private optimization. Additionally, the noise generation in (1) has negligible overhead. In this work, we adopt the Book-Keeping (BK) algorithm (8), which is state-of-the-art and only adding $\approx 10\%$ time/space complexity to the standard optimization of large ViT and GPT2 models.

2.2 Zero Redundancy Optimizer

2.2.1 Parallel computing

Parallel computing is necessary to train large-scale models and is critical to the efficiency. For models that fit in a single device, data parallelism (DataP) can be used to scale up the training by partition the batch of samples into multiple micro-batches on multiple devices. That is, each device holds a full copy of parameters and executes its forward and backward propagation, and generates the parameter gradient on its micro-batch of samples, which is averaged across devices to update the trainable parameters. For models that do not fit in a single device, the model parameters are partitioned by model parallelism (ModelP) and pipeline parallelism (PipeP).

ModelP partitions a model vertically, e.g. using 3 GPUs to store the parameters of one layer. As a consequence, ModelP does not scale efficiently beyond a single node due to fine-grained computation and expensive communication between layers. Notice that ModelP is complementary to ZeRO for its use on very large models. In contrast, PipeP partitions a model horizontally across layers, e.g. storing 3 layers in each GPU. Each device deals with all micro-batches sequentially, and PipeP can be inefficient due to the "pipeline bubble", which is overcome by ZeRO.

2.2.2 Model state partition

We take an example of mixed precision Adam optimizer to introduce the three stages of ZeRO .

Optimizer state partition (P_{os}) The optimizer states in ZeRO are fp32: the (master) parameters, variance and momentum, each taking $4\Psi_{\text{model}}$ memory. ZeRO1 only applies the optimizer state partition, reducing the memory cost of model states from $16\Psi_{\text{model}}$ to $(4 + \frac{12}{N_d})\Psi_{\text{model}}$ at each device.

Gradient partition (P_g) During the back-propagation, the fp16 gradients are partitioned, taking $2\Psi_{\text{model}}$ memory. In addition to ZeRO1, ZeRO2 further applies the gradient partition, reducing the memory cost to $(2 + \frac{14}{N_d})\Psi_{\text{model}}$ at each device.

Parameter partition (P_p) During the forward propagation, the fp16 parameters are partitioned, taking $2\Psi_{\text{model}}$ memory. In addition to ZeRO2, ZeRO3 also applies the parameter partition, further reducing the memory cost to $\frac{16}{N_d}\Psi_{\text{model}}$ at each device. Notice that FullyShardedDataParallel (26) (FSDP) is based on ZeRO3.

2.3 Parameter efficient fine-tuning

In contrast to full fine-tuning, where all model parameters are trainable, the parameter efficient fine-tuning (PEFT) only trains a small amount (e.g. 0.1%) of model parameters. Some examples include LoRA (16), Adapter (15), BiTFiT (25), and linear probing. Consequently, PEFT significantly boosts the computation and communication efficiency in three ways: (I) The most amazing save is the communication cost of the micro-batched parameter gradient among devices, possibly by $1000\times$. (II) PEFT has a theoretically 50% speedup[2], which has been observed in LoRA (see Footnote 5 of (16)) and BiTFiT (see Section 3.2 of (7)) in practice. (III) PEFT can save the memory cost on non-trainable layers by not caching the parameter gradient nor the activation tensor, and therefore trade the saved memory for larger batch size and faster training. We highlight that this work is parallel to PEFT and will present results in this direction.

	model	# param(Ψ_{model})	# trainable param(Ψ_{train})
(14)	GPT3	175B	151M
(8)	GPT2-large	774M	774M
(7)	GPT2-large	774M	0.5M
this work	ViT-Gigantic	1.84B	1.84B
this work	GPT2-XL	1.56B	1.56B
this work	GPT-J	6B	6B
this work	GPT3	175B	175B
this work	GPT	1T	1T

Table 2: State-of-the-art large-scale optimization with differential privacy.

2.4 Mixed precision training

Mixed precision training is a key technique which saves $\approx 50\%$ of the memory and accelerates the large model training by $\approx 20\%$, in comparison to the full precision training. This is achieved by performing the forward and backward propagation on the half precision (fp16) parameters, activations, and gradients, which only occupy half the space of the full precision (fp32) and take less time to multiply or add. We highlight that because the per-sample gradient clipping already plays the role of scaling, one should not use loss scaling in DP mixed precision training.

Specifically, in standard mixed precision training, there are two steps: scaling up the loss before the back-propagation, and scaling down (by the same factor) the parameter gradient after the back-propagation. However, in DP mixed precision training, scaling up the loss may cause overflow, while scaling down the gradient is incorrect, as can be seen from the scales in the 'param gard' columns of Table 3.

loss scale=10^3	activation	output grad (scaled)	per-sample grad norm		clipping factor	param grad (not scaled down)	param grad (if scaled down)
			vec($AA^\top$)	vec($BB^\top$)			
standard w/o scaling	$10^{-3} \sim 10^2$	$10^{-7} \sim 10^1$	N/A	N/A	1	$10^{-10} \sim 10^1$	$10^{-10} \sim 10^1$
standard w/ scaling	$10^{-3} \sim 10^2$	$10^{-4} \sim 10^4$	N/A	N/A	1	$10^{-7} \sim 10^4$	$10^{-10} \sim 10^1$
DP w/o scaling	$10^{-3} \sim 10^2$	$10^{-7} \sim 10^1$	$10^2 \sim 10^3$	$10^{-6} \sim 10^0$	$10^{-3} \sim 10^2$	$10^{-10} \sim 10^1$	$10^{-10} \sim 10^1$
DP w/ scaling	$10^{-3} \sim 10^2$	$10^{-4} \sim 10^4$	$10^2 \sim 10^3$	$10^0 \sim 10^6$	$10^{-6} \sim 10^{-1}$	$10^{-10} \sim 10^1$	$10^{-13} \sim 10^{-2}$

Table 3: Illustrative example based on ViT-large CIFAR100.

3 Differentially private ZeRO

We describe the DP-ZeRO algorithm in Figure 1.

[2]Forward propagation, output gradient, parameter gradient each takes one unit of time complexity (see (8)). As PEFT almost eliminates the computation of parameter gradient, the training accelerates by 50% compared to full fine-tuning.

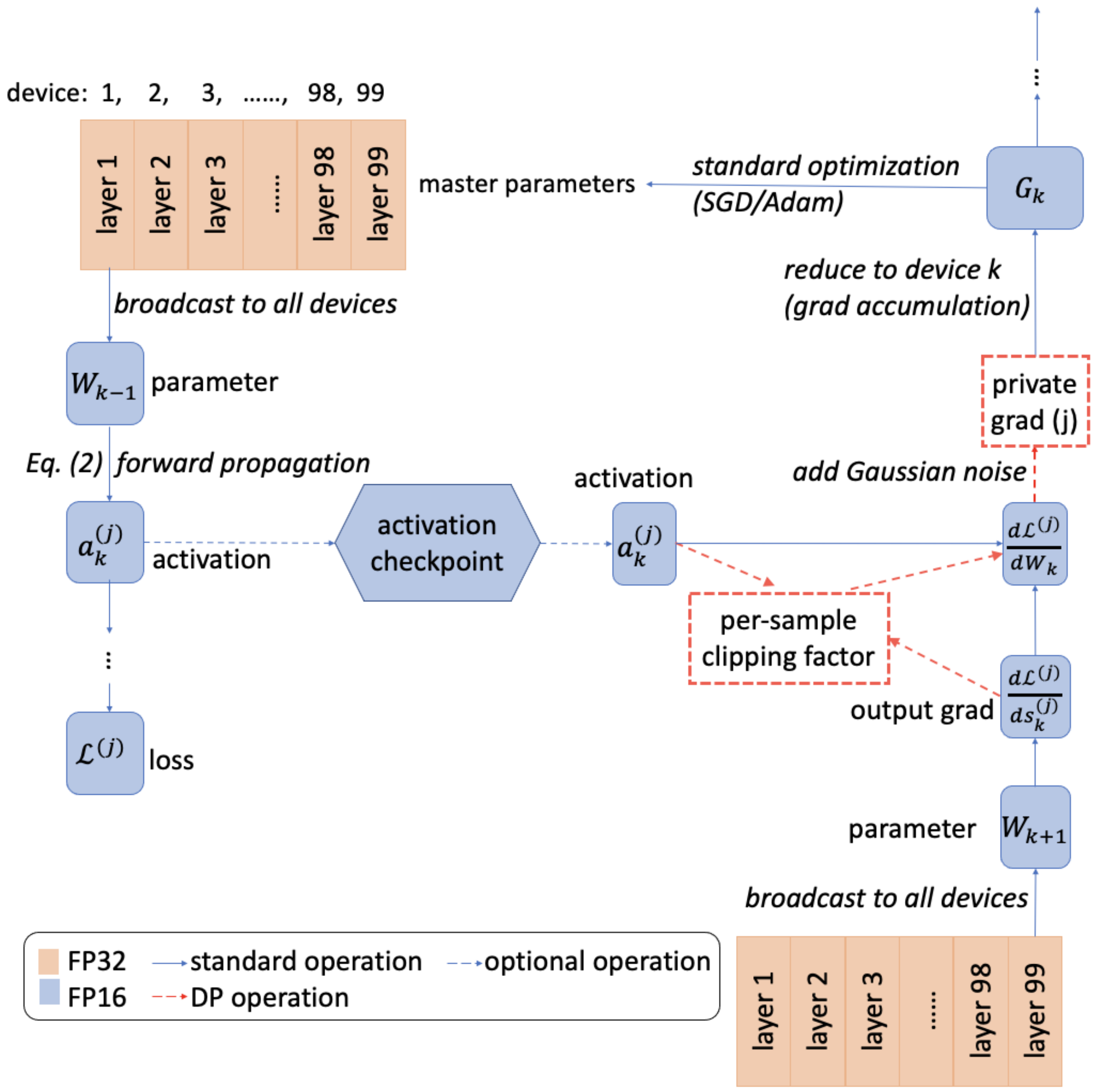


Figure 1: DP-ZeRO (stage 3).

Suppose we use N_d devices and train a neural network as

$$\boldsymbol{s}_l = \boldsymbol{a}_l \mathbf{W}_l + \mathbf{b}_l, \boldsymbol{a}_{l+1} = \phi_l(\boldsymbol{s}_l) \tag{2}$$

where $\boldsymbol{a}_l \in \mathbb{R}^{BT_l d_l}$ is the layer's input, also known as the activation, $\boldsymbol{s}_l \in \mathbb{R}^{BT_l p_l}$ is the layer's output, $\mathbf{W}_l \in \mathbb{R}^{d_l p_l}$ is the weight, $\mathbf{b}_l \in \mathbb{R}^{p_l}$ is the bias, and ϕ_l is any inter-layer operation such as ReLU, tanh or pooling. We denote B as the physical batch size[3] and T_l as the hidden feature dimension (e.g. text sequence length or number of pixels).

During the forward propagation, $\boldsymbol{a}_l$ is computed for each layer and stored in the computation graph. This activation is then used during the back-propagation to compute the parameter gradient $\boldsymbol{g}_l = \frac{\partial L}{\partial \mathbf{W}_l}$, where $L = \sum_i L_i$ is the sum of per-sample losses L_i.

[3] B (also known as the micro-batch size) is the number of samples processed by each of N_d devices and determines the time and memory efficiency, but not the accuracy. The logical batch size that determines the convergence/accuracy is $B \times N_d \times N_{acc}$, where N_{acc} is the steps of gradient accumulation.

During the back-propagation, the output gradient $\frac{\partial L}{\partial \boldsymbol{s}_l}$ is computed for each layer as

$$\frac{\partial L}{\partial \boldsymbol{s}_l} = \frac{\partial L}{\partial \boldsymbol{s}_{l+1}} \frac{\partial \boldsymbol{s}_{l+1}}{\partial \boldsymbol{a}_{l+1}} \circ \frac{\partial \boldsymbol{a}_{l+1}}{\partial \boldsymbol{s}_l} = \frac{\partial L}{\partial \boldsymbol{s}_{l+1}} \mathbf{W}_{l+1} \circ \phi_l'(\boldsymbol{s}_l).$$

In constrast, the parameter gradient is only computed if a layer is trainable, as

$$\text{Standard (non-DP): } \frac{\partial L}{\partial \mathbf{W}_l} = \frac{\partial \sum_i L_i}{\partial \mathbf{W}_l} = \boldsymbol{a}^\top \frac{\partial L}{\partial \boldsymbol{s}_l},$$

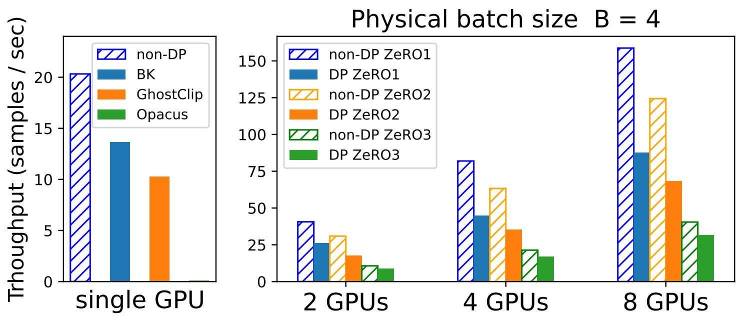
$$\text{DP (without noise): } \frac{\partial \sum_i C_i L_i}{\partial \mathbf{W}_l} = \boldsymbol{a}^\top \text{diag}(C_1, \cdots, C_B) \frac{\partial L}{\partial \boldsymbol{s}_l}.$$

Here the per-sample gradient norm (or the clipping factor C_i) can be computed at small cost, by the mixed ghost norm (4) (see also Appendix A).

In what follows, we describe certain insights in DP-ZeRO. First of all, it is clear that DP only affects the computation of gradients and thus do not affect the communication efficiency at all.

Secondly, an important difference between DP and standard mixed precision training is the loss scaling. Without DP, the mixed precision training scales up the loss L_i by $10^3 \sim 10^9$, computes the back-propagation, and then scales down by the same factor to fp32 gradient. The scale up prevents the underflow (i.e. fp16 gradient is too small to be representable), and the scale down recovers the correct magnitude of gradient, thus achieving almost the same accuracy as full precision training. However, because the per-sample gradient clipping already plays the role of scaling, one should not use loss scaling in DP mixed precision training, in order to prevent overflow and failure to train. We explain the details in appendix.

Thirdly, the noise addition in (1) is implemented with different magnitude, depending on whether a random seed is fixed on each device. If the seed is fixed for reproducibility, each device (say the j-th) will generate the same noise for the j-th micro-batch private gradient. Therefore, the noise magnitude is σ_{DP}/N in order to sum to the total noise $\sigma_{\text{DP}}\mathcal{N}(0, \mathbf{I})$; otherwise, the j-th private gradient requires $\frac{\sigma_{\text{DP}}}{\sqrt{N}}$ noise because each device has independent noise. In the single-device learning ($N = 1$), the noise addition is equivalent either way.



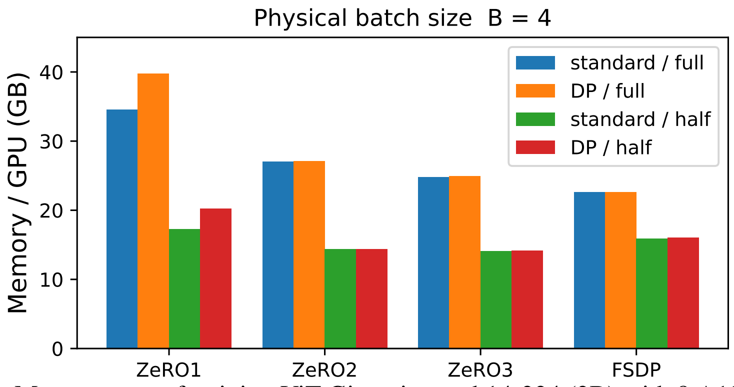


Figure 2: Memory cost of training ViT-Gigantic-patch14-224 (2B) with 8 A100 GPUs.

We provide the code to experiment with GPT2-XL and GPT-J in the supplementary material. Larger models (e.g. GPT 175B/1T) will be released in the full version.

References

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

[2] Galen Andrew, Om Thakkar, Brendan McMahan, and Swaroop Ramaswamy. Differentially private learning with adaptive clipping. *Advances in Neural Information Processing Systems*, 34, 2021.

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

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

[5] Zhiqi Bu, Hua Wang, and Qi Long. On the convergence and calibration of deep learning with differential privacy. *arXiv preprint arXiv:2106.07830*, 2021.

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

[7] Zhiqi Bu, Yu-Xiang Wang, Sheng Zha, and George Karypis. Differentially private bias-term only fine-tuning of foundation models. *arXiv preprint arXiv:2210.00036*, 2022.

[8] Zhiqi Bu, Yu-Xiang Wang, Sheng Zha, and George Karypis. Differentially private optimization on large model at small cost. *arXiv preprint arXiv:2210.00038*, 2022.

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

[10] Jinshuo Dong, Aaron Roth, and Weijie J Su. Gaussian differential privacy. *arXiv preprint arXiv:1905.02383*, 2019.

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

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

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

[14] Jiyan He, Xuechen Li, Da Yu, Huishuai Zhang, Janardhan Kulkarni, Yin Tat Lee, Arturs Backurs, Nenghai Yu, and Jiang Bian. Exploring the limits of differentially private deep learning with group-wise clipping. *arXiv preprint arXiv:2212.01539*, 2022.

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

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

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

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

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

[20] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *International Conference on Learning Representations*, 2018.

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

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

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

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

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

[26] Yanli Zhao, Rohan Varma, Chien-Chin Huang, Shen Li, Min Xu, and Alban Desmaison. Introducing pytorch fully sharded data parallel (fsdp) api. `https://pytorch.org/blog/introducing-pytorch-fully-sharded-data-parallel-api/`.

[27] Yuqing Zhu, Jinshuo Dong, and Yu-Xiang Wang. Optimal accounting of differential privacy via characteristic function. *arXiv preprint arXiv:2106.08567*, 2021.

A EFFICIENT COMPUTATION OF PER-SAMPLE GRADIENT NORMS

The mixed ghost norm (4) is the state-of-the-art technique to compute the per-sample gradient norm. It hybridizes two basic techniques to compute the Frobenius norm of a product of tensors, say $X = A^\top B$, from $A \in \mathbb{R}^{Td}, B \in \mathbb{R}^{Tp}$.

$$\left\|A^\top B\right\|^2 = \|X\|_{\text{Fro}}^2 = \text{vec}\left(AA^\top\right) \cdot \text{vec}(BB^\top), \tag{3}$$

where vec flattens the tensor to an one-dimensional vector.

To be more specific, although both equalities are equivalent mathematically, they are computed differently and thus have significantly different efficiency:

- $\|X\|_{\text{Fro}}^2 = \left\|A^\top B\right\|^2$ firstly computes $A^\top B$ and then its norm.
- $\|X\|_{\text{Fro}}^2 = \text{vec}\left(AA^\top\right) \cdot \text{vec}(BB^\top)$ firstly computes $AA^\top, BB^\top \in \mathbb{R}^{TT}$ and then their dot product.

That is, the second technique computes X's norm without ever instantiating the variable X, similar to the idea of Gaussian elimination that solves $A^{-1}b$ without inverting A.

The complexity analysis (4; 8) has shown that $\|X\|_{\text{Fro}}^2 = \left\|A^\top B\right\|^2$ requires pd space complexity and $2Tpd$ time complexity, while $\|X\|_{\text{Fro}}^2 = \text{vec}\left(AA^\top\right) \cdot \text{vec}(BB^\top)$ requires $2T^2$ space complexity and $2T^2(p+d)$ time complexity.

To put this into perspective, to compute the per-sample gradient norm, we denote $X = \frac{\partial L_i}{\partial W_l}, A = \boldsymbol{a}_{l,i}, B = \frac{\partial L}{\partial \boldsymbol{s}_{l,i}} = \frac{\partial L_i}{\partial \boldsymbol{s}_{l,i}}$. Consequently, the two techniques are termed the per-sample gradient instantiation and the ghost norm.

$$\left\|\boldsymbol{a}_{(l),i}^\top \frac{\partial L}{\partial \boldsymbol{s}_{(l),i}}\right\|^2 \overset{\text{per-sample grad}}{=\!=\!=} \left\|\frac{\partial L_i}{\partial \mathbf{W}_{(l)}}\right\|_{\text{Fro}}^2 \overset{\text{ghost norm}}{=\!=\!=} \text{vec}\left(\frac{\partial L}{\partial \boldsymbol{s}_{(l),i}} \frac{\partial L}{\partial \boldsymbol{s}_{(l),i}}^\top\right) \cdot \text{vec}(\boldsymbol{a}_{(l),i}\boldsymbol{a}_{(l),i}^\top) \tag{4}$$

To maximize out the memory efficiency, the mixed ghost norm uses a layer-wise decision rule that applies the ghost norm if and only if $2T^2 < pd$ on a layer. In other words, the mixed ghost norm always applies whichever techniques that is the cheapest, thus reducing the space complexity of computing the weight's per-sample gradient norm to $\min(2BT^2, Bpd)$.

Finally, we note that the per-sample gradient of the bias is computed differently. This is because

$$\frac{\partial L_i}{\partial b} = \mathbf{1}^\top \frac{\partial L}{\partial \boldsymbol{s}_i}$$

is activation-free and not actually a product of tensors like $X = A^\top B$. In fact, the multiplication with $\mathbf{1}$ turns out to be a summation along the first dimension $T \times p \to p$. This is equivalent to set $d = 1$ and hence we always use per-sample gradient instantiation to compute the bias gradient norms.