Efficient Federated and Privacy-preserving Machine Learning

Reza Nasirigerdeh

TUM School of Computation, Information and Technology
Technische Universität München

Efficient Federated and Privacy-preserving Machine Learning

Reza Nasirigerdeh

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz:
Prof. Dr. Pramod Bhatotia

Prüfende der Dissertation:
1. Prof. Dr. Daniel Rueckert
2. Assoc. Prof. Dr. Hamed Haddadi

Die Dissertation wurde am 07.03.2024 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 30.09.2024 angenommen.

Dedicated to my dear grandmother and my lovely wife.

Abstract

Machine learning models such as deep neural networks (*DNNs*) rely on large-scale datasets to be trained effectively, which is also the case for regression models in particular applications such as genome-wide association studies (*GWAS*). However, it is very difficult to procure such large datasets in a centralized manner due to the distributed nature of the data and the associated privacy regulations. *Federated learning* (*FL*) addresses the data availability problem, but it poses new challenges in terms of *utility* and *network communication*. Moreover, both FL and centralized learning (*CL*) face the *privacy* challenge, where they might leak privacy-sensitive information during training. *Differentially private* learning (*DP*) is the gold standard to tackle the privacy challenge, but it adversely impacts the model utility. Considering that, this dissertation aims to make the training procedure more efficient in terms of utility, communication, and/or privacy in the privacy-related domains (FL, DP, and DP-FL) given CL as baseline. In the first study, we introduce a tool called *sPLINK* for GWAS, implementing the federated versions of the linear and logistic regression models. We show that with *high communication efficiency*, sPLINK provides *optimal utility*, which is identical to the utility from CL, for the *regression models* independent of the data distribution across clients. In the second study, we demonstrate *federated DNN models* can also achieve *optimal utility* similar to the regression models provided that particular conditions hold for training components. In these studies, we do not improve all three aforementioned factors at the same time, which is closely related to the *communication-utility-privacy* (*CUP*) *trade-off*, stating that it is impossible to enhance all three aspects simultaneously for given training components. We argue that we can break the CUP trade-off by relaxing its underlying assumption, i.e. by replacing a training component with a more efficient one. In the third and fourth studies, we focus on the normalization layer of DNN models as a target training component to this end. We propose two novel layers called the *KernelNorm* and *kernel normalized convolutional* (*KNConv*) layers, and incorporate them into *kernel normalized convolutional networks* (*KNConvNets*). We experimentally illustrate KNConvNets are efficient not only in CL but also in FL, DP, and DP-FL. Finally, we show that using a kernel normalized ResNet, we can simultaneously enhance utility, communication, and privacy, and break the CUP trade-off.

Zusammenfassung

Modelle des maschinellen Lernens wie tiefe neuronale Netzwerke (DNNs) sind auf große Datensätze angewiesen, um effektiv trainiert werden zu können, was auch für Regressionsmodelle in bestimmten Anwendungen wie genomweiten Assoziationsstudien (GWAS) gilt. Aufgrund der verteilten Daten und der damit verbundenen Datenschutzbestimmungen ist es jedoch sehr schwierig, solche großen Datensätze zentral zu beschaffen. Föderiertes Lernen (FL) löst das Problem der Datenverfügbarkeit, stellt aber neue Herausforderungen an den Nutzen und die Netzwerkkommunikation. Darüber hinaus sind sowohl FL als auch zentralisiertes Lernen (CL) mit dem Problem des Datenschutzes konfrontiert, da während des Trainings datenschutzrelevante Informationen preisgegeben werden könnten. Differenziell privates Lernen (DP) ist der Goldstandard, um das Problem der Privatsphäre zu lösen, aber wirkt sich negativ auf den Modellnutzen aus. In Anbetracht dessen zielt diese Dissertation darauf ab, das Trainingsverfahren in Bezug auf Nutzen, Kommunikation und/oder Datenschutz in den datenschutzrelevanten Bereichen (FL, DP und DP-FL) effizienter zu gestalten, wobei CL als Basis dient. In der ersten Studie stellen wir ein Tool namens sPLINK für GWAS vor, das die föderierten Versionen der linearen und logistischen Regressionsmodelle implementiert. Wir zeigen, dass sPLINK bei hoher Kommunikationseffizienz einen optimalen Nutzen bietet, der mit dem Nutzen von CL identisch ist, und zwar für die Regressionsmodelle unabhängig von der Datenverteilung auf den Clients. In der zweiten Studie zeigen wir, dass föderierte DNN-Modelle ähnlich wie Regressionsmodelle einen optimalen Nutzen erzielen können, wenn bestimmte Bedingungen für die Trainingskomponenten erfüllt sind. In dieser Studie werden nicht alle drei oben genannten Faktoren gleichzeitig verbessert, was eng mit dem Kompromiss zwischen Kommunikation, Nutzen und Privatsphäre (CUP) zusammenhängt, welcher besagt, dass es unmöglich ist, alle drei Aspekte gleichzeitig für bestimmte Trainingskomponenten zu verbessern. Wir argumentieren, dass wir den CUP-Kompromiss beheben können, indem wir die zugrundeliegende Annahme lockern, d.h. indem wir eine Trainingskomponente durch eine effizientere Komponente ersetzen. In der dritten und vierten Studie konzentrieren wir uns auf die Normalisierungsschicht von DNN-Modellen. Wir schlagen zwei neue Schichten vor, die KernelNorm- und die kernelnormierte Faltungsschicht (KNConv), und integrieren sie in kernelnormierte

Faltungsnetze (KNConvNets). Wir zeigen experimentell, dass KNConvNets nicht nur in CL, sondern auch in FL, DP und DP-FL effizient sind. Schließlich zeigen wir, dass wir mit einem kernelnormalisierten ResNet gleichzeitig den Nutzen erhöhen können, Kommunikation und Privatsphäre verbessern und den CUP-Kompromiss aufheben können.

Acknowledgements

Pursuing PhD studies is a long and challenging journey in your academic life. Fortunately, there are people who support and tolerate you during this journey and enthusiastically assist you to overcome the challenges. First and foremost, I am very grateful to my supervisors, Prof. Daniel Rueckert and Dr. Georgios Kaissis, for accepting me as their PhD student in the first place, and for providing me with freedom of thought, valuable advice, and continuous support. Your extensive knowledge, experience, enthusiasm, patience, and professional behavior greatly helped me to cope with the challenges of my PhD research. I feel honored and blessed to have you as my advisors.

I would also like to thank Reihaneh not only as my research collaborator but also as my wonderful, lovely, and supportive wife who was always there with me at every stage of my PhD. Additionally, I would like to thank my former advisor, Prof. Jan Baumbach, for giving me the opportunity to switch my research direction to federated learning, and for his eager support during the early stages of my PhD. Besides, I would like to thank Prof. Julia Schnabel for kindly supporting me in finalizing my doctoral dissertation concurrently with pursuing the PostDoc research at her chair.

I would also like to thank my peers on the FeatureCloud project: Julian Matschinske, Julian Späth, Balazs-Attila Orban, and Sándor-József Fejér. You were amazing teammates; I really enjoyed working with you, and learned lots of programming and communication skills from you. Moreover, I would like to express my gratitude to my senior collaborators Prof. Markus List, Prof. David B. Blumenthal, Dr. Nina Kerstin Wenke, Dr. Tobias Frisch, and Dr. Olga Zolotareva. Thank you for your advice and assistance in the initial phases of my PhD research. It was a great pleasure to work with you all. Additionally, I thank Prof. Tim Kacprowski, Prof. Richard Röttger, Dr. Anne Hartebrodt, Prof. Uwe Völker, Prof. Esa Pitkänen, Prof. Dominik Heider, and Dr. Stefan Weiss for their support and collaboration on my first PhD project (sPLINK). Besides, I thank Javad TorkzadehMahani for his fruitful discussions and collaboration in my final PhD project (KenrelNorm).

I would also like to thank Prof. Pramod Bhatotia and Prof. Hamed Haddadi for agreeing to serve as the chairman and examiner of my dissertation, respectively.

An incredible source of support comes from my colleagues in the AIM lab: Veronika, Simone, Sabine, Kerstin, Alex, Maik, Jiazhen, Özgün, Sophie, Tamara, Vasiliki, Wenqi, Yundi, Nil, Alina, Martin, Moritz, Paul, Felix, Can, Daniel, Dima, Florent, Florian, Haifa, Ivan, Johannes, Jonas, Julian, Kristian, Leo, Linus, Manuel, Philip, Robbie, Supro, and Deborah. Thank you all for the happy times during lunch, get-togethers, Christmas parties, and scientific retreat at Berndlalm. It was a great pleasure both scientifically and emotionally to be surrounded by you.

A special thanks to Martina for her constant support throughout my PhD studies. Thank you for finding a flat for us (my wife and me) before we moved to Germany, assisting us with the visa and contract stuff, and for all the happy times you gave us.

A considerable source of emotional support comes from my dearest friends outside academia. Thank you, Isa, AmirHossein, Arash, Parisa, Sophia, Saeed, Keyvan, Soroush, Mahyar, Mohammad, Mahdi, Amir, Sajjad, Akram, Ali, Yasamin, Mohsen, Keyhan, Amin, Leila, Fatemeh, Yaser, and Zakaria. That was indispensable during my doctoral studies to keep me motivated and healthy throughout.

Last but not least, I would like to express my profound gratitude to my family as well as my wife's family for their emotional support and unconditional love throughout my PhD studies. I always miss you. Love and respect.

Contents

List of Figures

Publication List

The following *four sole first-author*, peer-reviewed publications constitute the core of this *cumulative doctoral dissertation*:

[1] **R. Nasirigerdeh**, R. Torkzadehmahani, J. Matschinske, T. Frisch, M. List, J. Späth, S. Weiss, U. Völker, E. Pitkänen, D. Heider, et al. “sPLINK: a hybrid federated tool as a robust alternative to meta-analysis in genome-wide association studies.” In: *Genome Biology* 23 (2022), pp. 1–24.

[2] **R. Nasirigerdeh**, D. Rueckert, and G. Kaissis. “Utility-preserving Federated Learning.” In: *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*. AISec ’23. Copenhagen, Denmark: Association for Computing Machinery, 2023, pp. 55–65.

[3] **R. Nasirigerdeh**, R. Torkzadehmahani, D. Rueckert, and G. Kaissis. “Kernel Normalized Convolutional Networks.” In: *Transactions on Machine Learning Research* (2024), pp. 1–21.

[4] **R. Nasirigerdeh**, J. Torkzadehmahani, D. Rueckert, and G. Kaissis. “Kernel Normalized Convolutional Networks for Privacy-Preserving Machine Learning.” In: *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*. IEEE. 2023, pp. 107–118.

The following additional *eight* publications were further co-authored *during the time of the doctoral thesis*. A * indicates shared first authorship.

2023

[1] R. Torkzadehmahani, **R. Nasirigerdeh**, D. Rueckert, and G. Kaissis. “Label Noise-Robust Learning using a Confidence-Based Sieving Strategy.” In: *Transactions on Machine Learning Research* (2023).

[2] J. Matschinske, J. Späth, M. Bakhtiari, N. Probul, M. M. Kazemi Majdabadi, **R. Nasirigerdeh**, R. Torkzadehmahani, A. Hartebrodt, B.-A. Orban, S.-J. Fejér, et al. “The FeatureCloud Platform for Federated Learning in Biomedicine: Unified Approach.” In: *Journal of Medical Internet Research* 25 (2023), e42621.

2022

[1] R. Torkzadehmahani, **R. Nasirigerdeh**, D. B. Blumenthal, T. Kacprowski, M. List, J. Matschinske, J. Spaeth, N. K. Wenke, and J. Baumbach. “Privacy-preserving artificial intelligence techniques in biomedicine.” In: *Methods of Information in Medicine* 61 (2022), e12–e27.

2021

[1] O. Zolotareva*, **R. Nasirigerdeh***, J. Matschinske, R. Torkzadehmahani, M. Bakhtiari, T. Frisch, J. Späth, D. B. Blumenthal, A. Abbasinejad, P. Tieri, et al. “Flimma: a federated and privacy-aware tool for differential gene expression analysis.” In: *Genome biology* 22 (2021), pp. 1–26.

[2] **R. Nasirigerdeh**, R. Torkzadehmahani, J. Baumbach, and D. B. Blumenthal. “On the Privacy of Federated Pipelines.” In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR ’21)*. 2021.

[3] **R. Nasirigerdeh**, R. Torkzadehmahani, J. Matschinske, J. Baumbach, D. Rueckert, and G. Kaissis. “HyFed: A Hybrid Federated Framework for Privacy-preserving Machine Learning.” In: *arXiv preprint arXiv:2105.10545* (2021).

[4] A. Hartebrodt, **R. Nasirigerdeh**, D. B. Blumenthal, and R. Röttger. “Federated principal component analysis for genome-wide association studies.” In: *2021 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2021, pp. 1090–1095.

2020

[1] S. Sadegh, J. Matschinske, D. B. Blumenthal, G. Galindez, T. Kacprowski, M. List, **R. Nasirigerdeh**, M. Oubounyt, A. Pichlmair, T. D. Rose, et al. “Exploring the SARS-CoV-2 virus-host-drug interactome for drug repurposing.” In: *Nature communications* 11.1 (2020), p. 3518.

PART I

INTRODUCTION

Introduction

Machine learning models have achieved growing popularity in a wide range of applications due to their considerable potential for tackling real-world problems [1, 2]. *Deep neural network* (*DNN*) models, in particular, have successfully been employed in numerous domains such as computer vision [3, 4], speech recognition [5], natural language processing [6, 7], and medical imaging [8, 9]. *Regression* models, moreover, are still popular in biomedical applications including *genome-wide association studies* (*GWAS*), which examine millions of single nucleotide polymorphisms (SNPs) to discover potential associations between a particular SNP and disease [10].

DNNs, however, rely on large amounts of data to be trained effectively. This is also the case for GWAS, where larger datasets result in discovering more associations and more accurate genetic predictors [11, 12]. On the other hand, it is extremely difficult to procure such large-scale datasets in a centralized manner. This is because in practice, data is distributed across multiple locations under different administrative domains, and moving the distributed data to a centralized site is close to impossible due to the privacy rules and regulations [13, 14, 15]. We refer to this challenge as large-scale *data availability* challenge.

Federated learning [16] addresses the data availability challenge by enabling multiple clients (e.g. hospitals or mobile devices) to collaboratively train a global model under the coordination of a central server without sharing their private data with third parties [17]. Federated learning, on the other hand, poses new challenges in terms of *utility* and *network communication* [18]. A model trained in a federated fashion might deliver lower accuracy than the model trained on the centralized data, particularly if data is not independent and identically distributed (NonIID) across the clients [19]. Federated learning, moreover, might incur significant communication overhead, requiring a remarkable amount of network traffic to be exchanged between the server and clients [18].

Both centralized and federated learning face the *privacy* challenge, in which the trained model or intermediate model parameters can leak the privacy-sensitive information about a specific individual participating in the dataset [20, 21]. Prior studies show the attacker can determine the presence of an individual in the training dataset from the released centralized model or local model parameters shared with the server in federated training (known as *membership inference attacks*) [22, 23, 24]. The revealed model parameters (e.g. gradients or weights) might also be exploited for reconstructing the training samples (referred to as *reconstruction attacks*) [25, 26].

Differential privacy [27] is the gold standard to address the privacy challenge in both centralized and federated environments. Differential privacy is a theoretical framework and collection of methods to process or release data in a *privacy-preserving* manner. In the context of DNNs, *differentially private learning* aims to limit the information learnt about a specific sample in the training dataset by injecting random noise into the clipped gradients of the model [28]. Differential privacy, however, faces the utility challenge, where the model utility is adversely affected by the gradient clipping and injected noise [29].

In summary, the training environments need to deal with different challenges, depending on if data is centralized or distributed, and whether or not the training procedure is differentially private. Given that, we categorize the learning environments into the following:

- *Centralized learning* (*CL*): The training data is located in a centralized site, and a single model is trained on the *centralized data* without using differential privacy. This training setting is *non-private*.

- *Federated learning* (*FL*): Data is *distributed* across multiple clients, where each client trains a local model on its private data, and only shares the model parameters with the server, which in turn, aggregates the parameters from the clients to obtain the global model. We refer to this environment as *privacy-enhancing* or *privacy-aware* due to the fact that the clients keep their private data on-site, enhancing data governance. However, it is not privacy-preserving because the clients do not capitalize on differential privacy during training.

- *Differentially private learning* (*DP*): The data is *centralized*, and the model is trained on the data using differential privacy. This training environment is *privacy-preserving* due to the privacy guarantee offered by differential privacy.

- *Differentially private federated learning* (*DP-FL*): The training data is *distributed* across clients, and the clients train the local models in a federated and differentially private manner. This environment is indeed *privacy-preserving* because of the privacy guarantee from DP.

Aim of the dissertation: The main goal of this thesis is to make the training procedure more efficient or the most efficient in terms of *utility*, *network communication*, and/or *privacy* in *federated* or *privacy-preserving* learning environments given centralized training as baseline. The cornerstones of the thesis are four peer-reviewed publications, where the author of the thesis is the main contributor (sole first author): (1) *sPLINK* [30], (2) utility-preserving federated learning [31], (3) kernel normalization [32], and (4) kernel normalization for FL, DP, and DP-FL environments [33].

In the first study, we introduce a tool called ***sPLINK*** [30] (*safe PLINK* [34]) for GWAS, which implements the federated versions of the chi-square test, and linear and logistic regression models. We demonstrate that the aforementioned models trained using sPLINK on distributed GWAS data in a federated fashion achieve the same utility (in terms of p-values and set of identified significant SNPs) as the corresponding models trained using PLINK in a centralized manner on the aggregated GWAS data. We theoretically prove that this conclusion holds regardless of the data distribution across the clients. In other words, we show that the federated training procedures for the models deliver *optimal utility*, which is identical to utility from the corresponding centralized learning procedures. They are also *highly efficient* in terms of *network communication* because they need a few communication rounds to compute the statistics. sPLINK, moreover, employs *additive secret sharing* [35] to conceal the original values of the local parameters of the clients from the server, further improving privacy. However, it is still considered as privacy-enhancing, but not privacy-preserving because it does not preserve the privacy of the model *outputs*.

In the utility-preserving federated learning (***UPFL***) study [31], we theoretically prove and experimentally validate that DNN models can also deliver *optimal utility* in federated environments similar to chi-square and regression models provided that specific conditions hold for the training algorithm, model, loss function, and optimizer as the main DNN training components. In more detail, if the (1) training algorithm selects all clients, instruments them to carry out a single local update per communication round, and enforces the server to use sample size based weighted averaging as the aggregation function, (2) model and loss function are batch-independent and deterministic, and (3) optimizer employs a linear momentum function, then the DNN model trained in a federated manner has weights identical to those from centralized training independent of how data is distributed across the clients. The equivalence between the federated and centralized DNN models implies that they indeed achieve identical utility. UPFL, however, is highly inefficient from the network communication perspective, requiring a huge number of communication rounds for model convergence. It is also a *privacy-enhancing* learning environment akin to ordinary FL.

In the UPFL paper, we also investigate the properties of the existing DNN training components to determine which ones satisfy the necessary conditions for UPFL. Our examination shows that, for instance, the *convolutional* and *linear* layers can be incorporated in UPFL because they are batch-independent and deterministic. This is not the case, however, for *batch normalization* (*BatchNorm*) [36], which is a batch-dependent layer, where the normalization statistics of a given sample depends on the other samples in the batch. Interestingly, a component not holding the necessary conditions for UPFL typically causes utility reduction in NonIID federated settings too, although the underlying theoretical analysis of UPFL does not imply it. For example, the BatchNorm layer, *federated averaging* (*FedAvg*) algorithm [16], and *Adam* optimizer [37], which cannot be incorporated in UPFL, indeed deliver lower accuracy in NonIID environments compared to centralized training [19, 38, 39].

sPLINK and UPFL deliver optimal utility in non-privacy-preserving federated environments. The former provides high communication efficiency, while the latter incurs considerable communication overhead. In other words, sPLINK and UPFL *do not* make the training procedure efficient in terms of all three aforementioned perspectives. This is also closely related to the communication-utility-privacy (**CUP**) trade-off, which states it is impossible to improve communication, utility, and privacy simultaneously for *given training components* (i.e. algorithm, model, loss function, and optimizer). Considering that, an interesting question arises:

Can we break the CUP trade-off? If so, how? The CUP trade-off holds for given training components, i.e. the underlying assumption is that the training components remain unchanged. Relaxing this assumption (by replacing a particular component with a more efficient one) makes it possible to improve communication, utility, and privacy at the same time.

In this dissertation, we focus on the model, or more precisely, the normalization layer of the model to break the CUP trade-off. The motivation behind this choice is the contradictory behavior of BatchNorm, as the most widely used normalization layer, in centralized training and the privacy-related domains. While BatchNorm is extremely efficient in CL, it is inapplicable to DP and DP-FL. This is because in differentially private training, per-sample gradients are required, and the gradients of a particular sample are not allowed to be affected by the other samples in the batch. BatchNorm, on the other hand, breaks the independence among the samples in the batch by taking into account the batch dimension during normalization [32]. This makes BatchNorm inapplicable to privacy-preserving learning environments.

There are also batch-independent alternatives to BatchNorm such as *layer nor-*

malization (*LayerNorm*) [40] and *group normalization* (*GroupNorm*) [41]. These layers, however, cannot typically achieve the performance of BatchNorm in centralized training, especially in image classification. Moreover, their performance is not as much as expected in privacy-related domains.

Given that, in our kernel normalization study [32], we propose a novel *batch-independent* normalization layer called **KernelNorm** as an efficient alternative to the existing normalization layers for centralized, federated, and privacy-preserving learning environments. The KernelNorm layer is akin to the pooling layers, except that KernelNorm normalizes the elements specified by the kernel size instead of computing average or maximum of the elements. Additionally, KernelNorm operates over all channels rather than a single channel. The distinguishing characteristic of KernelNorm is the *overlapping* normalization units, which enables it to *extensively* benefit from the spatial correlation among the elements during normalization. KernelNorm, moreover, introduces a regularization effect during training by using slightly randomized normalization statistics (i.e. mean and variance) instead of the original statistics to normalize the elements (partially inspired by BatchNorm).

We also introduce the *kernel normalized convolutional* (**KNConv**) layer as the combination of KernelNorm and the traditional convolutional layer, where it first applies KernelNorm to the input tensor, and then, computes the convolution (dot product) between the normalized tensor and kernel weights. Due to the remarkable computation overhead of this naive form of KNConv, we propose a *computationally efficient* version of KNConv, in which the output of the convolutional layer is adjusted using the mean and variance of the normalization units instead of actually normalizing the elements. As an application of the proposed layers, we incorporate them into kernel normalized residual networks (**KNResNets**), while foregoing the BatchNorm layers. Through extensive experiments in centralized settings, we illustrate KNResNets (1) achieve higher or very competitive performance compared to the batch normalized counterparts, and (2) significantly outperform the other batch-independent (e.g. LayerNorm and GroupNorm) competitors in almost all considered cases. We also demonstrate KNResNet-18 provides higher accuracy than layer and group normalized ResNet-18 in differentially private training on the down-sampled ImageNet dataset [42]. In simple words, we show KernelNorm combines the performance advantage of BatchNorm with the batch-independence benefit of LayerNorm/GroupNorm.

In our last study [33], we extensively investigate the performance of KernelNorm in FL, DP, and DP-FL environments using VGG [43], DenseNet [44], and ResNet [45, 46] models. Our experimental evaluation indicates that kernel normalized models provide considerably higher accuracy and communication efficiency (convergence rate) compared with non-normalized, and layer/group normalized counterparts in all three aforementioned environments. We also propose a kernel normalized ResNet architecture called KNResNet-13, and improve the state-of-the-art accuracy on the CIFAR-10 [47] and Imagenette (a subset of ImageNet) [48] datasets in DP settings.

In summary, KernelNorm is a batch-independent layer, and thus, is applicable to privacy-preserving machine learning. KernelNorm is also a *local* normalization layer, which extensively considers spatial correlation among the elements during normalization. Our extensive experimental results demonstrate the efficiency of kernel normalized models not only in CL, but also in FL, DP, and DP-FL.

Breaking the CUP trade-off using KernelNorm: In a DP-FL environment, we first train ResNet-9-GN (based on GroupNorm) on CIFAR-10, where samples are distributed across clients in a NonIID fashion (more experimental details in Appendix B). The clients employ differential privacy with ε=8.0 and δ=10^{-5} to train the local models. Note that ε and δ are the privacy parameters, whose lower values imply stronger privacy. ResNet-9-GN achieves accuracy of 47.13% in this setting. Next, we replace ResNet-9-GN with the corresponding kernel normalized model, ResNet-9-KN, and train it with privacy parameters of ε=7.0 and δ=10^{-5}. ResNet-9-KN delivers accuracy of 49.95%, implying that ResNet-9-KN improves utility by 2.82% compared to ResNet-9-GN while providing stronger privacy. Moreover, ResNet-9-KN achieves higher communication efficiency than ResNet-9-GN according to Figure 1.1. These experimental results indicate that ResNet-9-KN enhances communication, utility, and privacy simultaneously, and breaks the CUP trade-off.

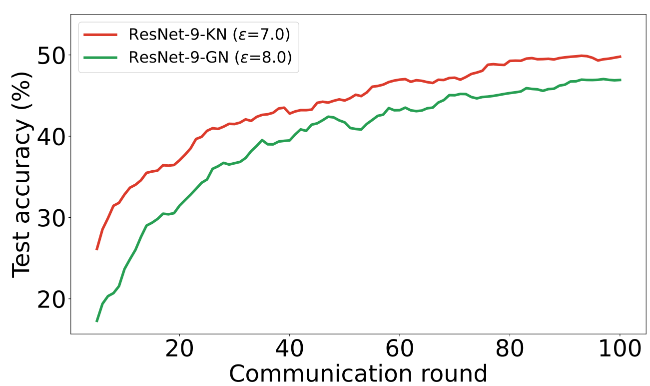


Figure 1.1: Breaking the CUP trade-off: With stronger privacy and in a fewer number of communication rounds, kernel normalized ResNet-9 achieves higher accuracy than group normalized ResNet-9.

Organization

This dissertation is organized in fours parts: Part I includes three chapters, of which the current one is Chapter 1. Chapter 2 provides a brief background on neural networks, centralized training, federated learning, differential privacy, and differentially private federated training, and discusses the related work. Chapter 3 summarizes the key contributions of the dissertation. Part II consists of Chapters 4-7, which present four first-author publications that constitute the core of the thesis. Each chapter presents the corresponding paper in a self-contained section, starting with the synopsis of the paper. Part III comprises Chapter 8, which provides the conclusions of the dissertation, and Chapter 9, which discusses the potential future directions for the thesis. Part IV contains supplementary materials associated with the publications, and the experimental setup for the CUP trade-off experiment.

Method

We provide a brief background on neural network models as well as different training environments including centralized, federated, differentially private, and differentially private federated environments in which the models are trained. In this dissertation, we particularly focus on convolutional neural networks [49], a special type of neural networks which are popular in image vision tasks such as image classification [3] and semantic segmentation [50].

2.1 Convolutional Neural Networks (CNNs)

A CNN model consists of multiple layers including the input layer, hidden layers, and the output layer stacked on top of each other. The data is fed into the input layer, which conducts an initial transformation to change the representation of the data; the hidden layers perform more complex transformations on the data representation obtained from the input layer; the output layer carries out the final prediction of the model. In the following, we overview the widely used layers in convolutional networks. Unless otherwise stated, we assume that the input data is a set of 2-dimensional images, and thus, the input of the layers is a 4-dimensional tensor with batch, channel, height, and width as dimensions.

Convolutional (Conv) layer: The Conv layer is the major building block of CNNs, which takes the number of input channels (filters) ch_{in}, number of output channels ch_{out}, kernel size (i.e. height and width) (k_h, k_w), stride (s_h, s_w), and padding (p_h, p_w) as the main arguments. The Conv layer first pads the input tensor; then, it computes the dot-product between the kernel weights and a subset of elements from the input tensor specified by the kernel window. Next, it slides the kernel window s_w elements along the width dimension and performs the dot-product computation with the new area. If there is not enough elements in the width dimension, it slides the window s_h elements in the height dimension, and repeats the dot-product calculation procedure.

If the input tensor is of shape (m, ch_{in}, h, w), in which m is batch size, ch_{in} is the number of channels, h is height, and w is width, then the output tensor from the Conv layer has the following shape:

$$(m, ch_{out}, \lfloor \frac{h + 2 \cdot p_h - k_h}{s_h} \rfloor + 1, \lfloor \frac{w + 2 \cdot p_w - k_w}{s_w} \rfloor + 1),$$

that is, the Conv layer might change the size of the channel, height, and width dimensions of the input tensor depending on the values of the kernel size, stride, padding, and the number of output channels.

The Conv layer has total of $ch_{in}.ch_{out}.k_h.k_w$ learnable (trainable) parameters. Note that if the bias flag of the Conv layer is set, then it will have ch_{out} additional trainable parameters. It is worth mentioning that the Conv layer has other arguments such as the number of groups, and the dilation value. We do not include them in our description for simplicity purposes. We assume that the value of the aforementioned arguments is unit.

Pooling layers: The pooling layers take kernel size (k_h, k_w), stride (s_h, s_w), and padding (p_h, p_w) as arguments similar to the Conv layer. The difference is that the pooling layers compute the maximum (max-pooling) or average (average-pooling) over the elements specified by the kernel window instead of performing dot-product calculation. Moreover, they operate on a single input channel rather than all channels.

The shape of the output from the pooling layers is as follows:

$$(m, ch_{in}, \lfloor \frac{h + 2 \cdot p_h - k_h}{s_h} \rfloor + 1, \lfloor \frac{w + 2 \cdot p_w - k_w}{s_w} \rfloor + 1),$$

i.e. the pooling layers do not change the size of the channel dimension, but they might modify the width and height of the input tensor. Unlike the Conv layer, the pooling layers have no trainable parameters.

Linear (fully-connected) layer: The fully-connected layer performs a linear transformation on the input tensor. More precisely, if the input X=$(x_1, \ldots, x_l)$ is a one-dimensional tensor of size l, W=$(w_1, \ldots, w_l)$ is the weight tensor of a linear layer with l input neurons and a single output neuron, and b is the bias value, then the output is computed as follows:

$$o = X \cdot W + b = x_1 \cdot w_1 + \ldots + x_l \cdot w_l + b,$$

that is, the output is the dot-product of the input tensor and weights of the linear layer with bias added. This is the simplest form of the linear layer, which is equivalent to the regression model.

The linear layer, in general, takes the number of input neurons n_{in} and output neurons n_{out} as arguments, and performs the dot-product computation independently for each output neuron. If the input tensor is of shape $(m,\, n_{in})$, then the output tensor from the linear layer has the shape of $(m,\, n_{out})$. The layer has total of $n_{out} \cdot (1 + n_{in})$ trainable parameters assuming that the bias flag is set.

Activation layers: The activation layers carry out non-linear transformations on the input tensor. The rectified linear unit (ReLU) activation is the most widely adopted activation layer in CNNs. The ReLU activation maps the non-positive values to zero, but performs identity mapping on the positive values. There are other variants of ReLU such as LeakyReLU [51] and SELU [52], which might improve the performance of the model compared to ReLU in particular tasks. In general, activation layers have no learnable parameters, and their output has identical shape to the input tensor's.

Normalization layers: The normalization layer is another major building block of convolutional networks. The existing state-of-the-art normalization layers are based on standard normalization, i.e zero-mean and unit-variance. In more details, they (1) consider a subset of elements from the input tensor as their normalization unit [32], (2) compute the mean and variance over the elements of the normalization unit, (3) normalize the elements by first subtracting the mean from each element, and then, dividing the result by the square root of the variance:

$$\hat{U} = \frac{U - \mu_U}{\sqrt{\sigma_U^2 + \epsilon}},$$

where $\hat{U}$ is the normalized unit, U is the normalization unit, μ_U and σ_U^2 are the mean and variance of the normalization unit, respectively, and ϵ is a small constant (e.g. 10^{-5}) for numerical stability, (4) concatenate the resulting normalized units together to obtain the whole normalized tensor, and (5) apply the per-channel shift and scale parameters to the whole normalized tensor to compute the final output tensor:

$$Y_i = \alpha_i \cdot \hat{X}_i + \beta_i,$$

where Y is the output tensor, $\hat{X}$ is the whole normalized tensor, and β and α are the per-channel shift and scale parameters, respectively, and i is the channel number.

The normalization layers are different from each other in the normalization units they consider during normalization. In the following, we provide a brief overview on the popular normalization layers from that aspect assuming that the input tensor is of shape (m, ch_{in}, h, w).

BatchNorm [36]: The BatchNorm layer considers all elements in the batch, height, and width dimensions as its normalization unit. Given that, BatchNorm has ch_{in} normalization units. Because BatchNorm employs the batch dimension during normalization, it is not a batch-independent layer.

LayerNorm [40]: The normalization unit of LayerNorm includes all elements from the channel, width, and height dimensions. LayerNorm has, thus, m normalization units. LayerNorm is a batch-independent layer due to the fact that it does not consider the batch dimension for normalization.

InstanceNorm [53]: The InstanceNorm layer incorporates all elements from the height and width dimensions during normalization. In other words, it carries out normalization independently of the batch and channel dimensions. InstanceNorm has $m \cdot ch_{in}$ normalization units.

GroupNorm [41]: The GroupNorm layer employs a subset of elements from the channel dimension, but all elements from the width and height dimensions for normalization. It has $m \cdot g$ normalization units, where g is the number of groups. Note that LayerNorm and InstanceNorm are special cases of GroupNorm, in which g=1 and g=ch_{in}, respectively.

It is worth noting that these normalization layers do not modify the shape of the input tensor. Moreover, shift and scale are trainable parameters, and as a result, they have $2ch_{in}$ learnable parameters. The readers are referred to [32] for more detail on the normalization layers.

CNN architectures: There are various CNN architectures, which use the aforementioned layers in different ways. In the following, we overview some of the well-known CNN architectures:

VGGNets [43]: The VGG architecture employs the Conv layers with kernel size of (3, 3) as the main building blocks. The architecture uses max-pooling with kernel size and stride of (2, 2) to downsample the input tensor. Each layer is only connected to the subsequent layer. In other words, the output of each layer is only given as input to the next layer. It is worth noting that VGGNets inherit many architectural aspects from AlexNet [3], which revealed, for the first time, the true potential of deep CNNs for image classification.

ResNets [45]: Residual networks are based on residual blocks, where the final output of the block is the summation of the input of the block and the output of the last layer in the block. In other words, the output of some layers are used as input not only to the subsequent layer but also to the sum operation at the end of the block. There are two types of residual blocks in ResNets: basic blocks, which consist of two Conv layers with kernel size of (3, 3), and bottleneck blocks that include three Conv layers with kernel sizes of (1, 1), (3, 3), (1, 1), respectively. ResNets employ convolutional residual blocks with stride of (2, 2) to downsample the input tensor.

DenseNets [44]: The DenseNet architecture is based on dense blocks in which the output of a given Conv layer is fed into *all* subsequent Conv layers. The dense blocks can be of type basic or bottleneck. The former includes the Conv layers with kernel size of (3, 3), whereas the latter incorporates two Conv layers with kernel sizes of (1, 1) and (3, 3), respectively. DenseNets use average-pooling with kernel size and stride of (2, 2) for downsampling the input.

EfficientNets [54]: The key idea behind the EfficientNet architecture is "compound scaling", where the number of filters in the Conv layers, the number of layers of the model, and the resolution of the input images are scaled in a balanced fashion. EfficientNets employ *grouped* Conv layers with various kernel sizes including (5, 5) and (1, 1) as major building blocks. The architecture capitalizes on the Conv layers with stride of (2, 2) to downsample the input.

All the aforementioned architectures use a final linear layer to perform the classification task, and their default normalization layer is BatchNorm. VGGNets, ResNets, and DenseNets employ ReLU as the activation layer, whereas the EfficientNet architecture is based on the Sigmoid Linear Unit (SiLU) activation function.

2.2 Centralized Learning (CL)

In a CL environment, a single model is trained on a centralized dataset. CL is considered as non-private because data from multiple sites needs to be moved to a centralized location, which can violate privacy. We use the model performance in the CL setting as our baseline throughout the thesis. In the following, we describe the training procedure for the regression and neural network models in CL environments. Focusing on supervised learning, we presume that $\mathcal{M}_W$ is the model characterized by parameters W, $\mathcal{L}$ is the loss function, and $\mathcal{D} = [S_1, \ldots, S_n]$ is the centralized dataset containing n samples, where S_j=(X_j, y_j) is the j-th sample in the dataset, X_j is the feature values of the sample, and y_j is the corresponding target value.

Linear regression: We capitalize on the *ordinary least squares* (OLS) method [55] to compute the parameters of the linear regression model as follows:

$$W = (X^T X)^{-1}(X^T Y), \tag{2.1}$$

where W is the model parameters, X and Y are the matrices containing the feature and target values of all samples, respectively, T is the transpose operation, and $(\cdot)^{-1}$ indicates the matrix inverse.

Logistic regression: We employ the *Newton-Raphson* method [55] to calculate the parameters of the logistic regression model. The computation of the parameters is performed in an iterative manner as the following:

$$\hat{Y} = \frac{1}{1 + e^{-XW_{i-1}}}, \tag{2.2}$$

$$\nabla = X^T(Y - \hat{Y}), \tag{2.3}$$

$$H = (X^T \circ (\hat{Y} \circ (1 - \hat{Y}))^T)X, \tag{2.4}$$

$$L = \sum (Y \circ \log \hat{Y} + (1 - Y) \circ \log(1 - \hat{Y})), \tag{2.5}$$

$$W_i = W_{i-1} + H^{-1}\nabla, \tag{2.6}$$

where W_i and W_{i-1} are the value of the parameters in the current and previous iterations, respectively, ∇ is the gradient vector, H is the Hessian matrix, L is the log-likelihood value, and $\circ$ indicates the element-wise multiplication. The training process continues until the difference between the log-likelihood values in the current and previous iterations becomes less than a given threshold.

Neural networks: *Gradient descent* is a widely used optimization algorithm for training neural networks. It comes with different versions among which *mini-batch* *gradient descent* (*MBGD*) [56] is the most popular one. The MBGD algorithm first shuffles the dataset, and then divides the dataset samples into mini-batches of size m. For a given mini-batch, it computes the gradient value associated with parameter $w \in W$ as follows:

$$\mathcal{G}_w(W, \mathcal{B}, Y_{\mathcal{B}}) = \frac{\partial \mathcal{L}(Y_{\mathcal{B}}, \mathcal{M}_W(\mathcal{B}))}{\partial w},$$
$$G_i = [g_{i,1}, \ldots, g_{i,m}], \quad g_i = \frac{1}{m}\sum_{j=1}^{m} g_{i,j}, \tag{2.7}$$

where $\mathcal{B}$ is a mini-batch from $\mathcal{D}$, $Y_{\mathcal{B}}$ is the corresponding vector of target values, $\mathcal{G}_w$ is the gradient function associated with parameter w, G_i is the vector of gradient values corresponding to the samples of the mini-batch in iteration i, $g_{i,j}$ is the gradient value associated with j-th sample of the mini-batch, and g_i is the gradient value corresponding to w, which is the average of the gradient values over the mini-batch.

After computing the gradient value, the model parameter is updated by a gradient descent-based optimizer as follows:

$$w_i = w_{i-1} - \eta \mathcal{V}(g_p, \ldots, g_i, \star), \tag{2.8}$$

where w_{i-1} is the value of the parameter in the previous iteration, g_i is the gradient value in the current iteration i, η is learning rate, $\mathcal{V}(g_p, \ldots, g_i, \star)$ is the momentum function that computes the final gradient using the gradients in the current and previous iteration(s) (p is also an iteration number and $p < i$), and $\star$ means the function can take additional arguments.

For instance, the momentum function of the widely adopted SGD optimizer is as the following:

$$\mathcal{V}_{SGD}(g_1, \ldots, g_i, \xi) = \xi^{i-1} g_1 + \ldots + \xi^{i-k} g_k + \ldots + g_i, 1 \leq k < i, \tag{2.9}$$

where ξ is the momentum value. That is, the momentum function of *SGD* takes the momentum value ξ and the gradient values from the first iteration to the current iteration $i > 1$ as inputs, and linearly combines them.

2.3 Federated Learning (FL)

In a FL setting, multiple clients as data holders train a joint (global) model under the orchestration of a server while keeping their data on-site [16]. FL is considered as a privacy-enhancing (or privacy-aware) environment because the raw data is not moved off-site unlike CL. FL, however, is not privacy-preserving because no differential privacy is employed during training. FL settings, in general, can be categorized into *cross-device* or *cross-silo* [17]. In the former, there are a large number of clients with unstable network connection (for instance mobile devices), and a fraction of clients is randomly selected by the server to participate in training. In the latter, there are few clients with reliable network connection (e.g. hospitals), and all clients participate in training in all communication rounds. In the following, we describe the cross-silo federated training process for the regression and neural network models.

Federated linear regression: Each client j computes $\alpha_j{=}X_j^T X_j$ and $\beta_j{=}X_j^T Y_j$ as local parameters, where X_j and Y_j are the feature matrix and target vector of the client's local data. In the aggregation phase, the server first takes sum over the local parameters from all k clients:

$$\alpha = \sum_{j=1}^{j=k} \alpha_j, \tag{2.10}$$

$$\beta = \sum_{j=1}^{j=k} \beta_j, \tag{2.11}$$

then, it calculates the global values of the linear regression parameters as follows:

$$W^g = (\alpha)^{-1}(\beta). \tag{2.12}$$

Federated logistic regression: Each client j calculates gradient (∇_j), Hessian matrix (H_j), and log-likelihood (L_j) values over its local data:

$$\hat{Y}_j = \frac{1}{1 + e^{-X_j W_{i-1}^g}}, \tag{2.13}$$

$$\nabla_j = X_j^T (Y_j - \hat{Y}_j), \tag{2.14}$$

$$H_j = (X_j^T \circ (\hat{Y}_j \circ (1 - \hat{Y}_j))^T) X_j, \tag{2.15}$$

$$L_j = \sum (Y_j \circ \log \hat{Y}_j + (1 - Y_j) \circ \log(1 - \hat{Y}_j)), \tag{2.16}$$

where W_{i-1}^g indicates the global values of the model parameters in iteration $i-1$ obtained from the server. The server adds up the values of the local parameters from the clients to compute the corresponding global values:

$$\nabla = \sum_{j=1}^{j=k} \nabla_j, H = \sum_{j=1}^{j=k} H_j, L = \sum_{j=1}^{j=k} L_j, \tag{2.17}$$

then, it updates the global values of the model accordingly:

$$W_i^g = W_{i-1}^g + H^{-1}\nabla. \tag{2.18}$$

The server also compares the newly computed log-likelihood value to the one from the previous iteration. If their difference is less than a given threshold, it completes the training process.

Federated neural networks: *Federated averaging* (*FedAvg*) [16] is the most commonly used algorithm for training neural networks in FL settings. In FedAvg, each client j trains the model obtained from the server on its local data using the MBGD algorithm, and shares the local parameters $W_{i,j}^l$ or accumulated local gradients $G_{i,j}^l$ and its sample size n_j with the server, which in turn, aggregates the local parameters/gradients from the clients using weighted averaging to compute the global model parameters:

$$W_i^g = \frac{\sum_{j=1}^k n_j W_{i,j}^l}{\sum_{j=1}^k n_j} = W_{i-1}^g - \eta \frac{\sum_{j=1}^k n_j G_{i,j}^l}{\sum_{j=1}^k n_j}, \tag{2.19}$$

where i indicates the communication round.

2.4 Differentially Private Learning (DP)

Differential privacy provides a theoretical framework and a collection of methods for processing and releasing data in a privacy-preserving fashion [27]. Formally, a randomised mechanism $\mathcal{M}$ preserves (ε, δ) differential privacy if, all databases D and D' differing in the data of one individual and all measurable subsets S of the range of $\mathcal{M}$, satisfy the following inequality:

$$\mathbb{P}(\mathcal{M}(D) \in S) \leq e^{\varepsilon} \mathbb{P}(\mathcal{M}(D') \in S) + \delta, \tag{2.20}$$

where $\mathbb{P}$ is the probability of an event, and $\varepsilon \geq 0$ and $0 \leq \delta \leq 1$ are privacy parameters, whose lower values imply stronger privacy.

Algorithm 1: Differentially private SGD (DP-SGD) [28]

Input: Samples $\{S_1, \ldots, S_n\}$, loss function $\mathcal{L}(W) = \frac{1}{n}\sum_j \mathcal{L}(W, x_j)$, learning rate η_i, noise scale σ, group size l, and gradient norm bound C.

$W_0 \leftarrow$ Random initialization

for i **from** 0 **to** $T-1$ **do**

$L_i \leftarrow$ Take a set of random samples with sampling probability l/n

`// Compute per-sample gradients`

for each sample $x_j \in L_i$ **do**

$g_i(x_j) \leftarrow \nabla_{W_i}\mathcal{L}(W_i, x_j)$

`// Clip gradients`

$\bar{g}_i(x_j) \leftarrow g_i(x_j) / \max\left(1, \frac{\|g_i(x_j)\|_2}{C}\right)$

`// Add noise`

$\tilde{g}_i \leftarrow \frac{1}{l}\left(\sum_j \bar{g}_i(x_j) + \mathcal{N}(0, \sigma^2 C^2 I)\right)$

`// Update parameters`

$W_{i+1} \leftarrow W_i - \eta_i \tilde{g}_i$

Output: W_T and calculate the overall privacy cost (ε, δ) using a privacy accountant technique.

In the context of neural networks, a DP environment trains a single model on a centralized dataset using the *differentially private stochastic gradient descent* (*DP-SGD*) algorithm [28], where the role of the database is played by the individual (per-sample) gradients of the loss function with respect to the parameters. As shown in Algorithm 1, DP-SGD (1) computes the per-sample gradients, (2) clips the gradients, (3) adds random noise to the clipped gradients, and (4) update the parameters using the average of the noisy clipped per-sample gradients.

2.5 Differentially Private Federated Learning (DP-FL)

A DP-FL environment consists of multiple clients and a server that coordinates the training procedure akin to FL. In DP-FL, however, the clients employ the DP-SGD algorithm to train the global model on their local data in a differentially private manner, and share differentially private gradients or parameters with the server. Given that, DP-FL is also considered as privacy-preserving similar to DP environments. Note that the server aggregates the private parameters/gradients from the clients using weighted averaging similar to FL.

2.6 Related Work

Given a brief background on different learning environments, we discuss the related work in the areas of federated, differentially private, and differentially private federated training, whose focus is on improving the efficiency of machine learning models similar to this dissertation.

Federated learning (**FL**) [16] enables multiple clients to participate in model training without sharing their private data with third parties. Federated training, however, poses new challenges in terms of utility, network communication, and privacy [18], which have been addressed in a body of prior works.

Many studies in that regard focus on enhancing the *utility* of the model in FL, mainly by proposing new training algorithms as alternatives to *FedAvg*. The *FedProx* algorithm [38] adds a proximal term to the loss function of the clients to act as a regularizer and to enforce the local models of the clients not to be far from the global model. *FedNova* [57] aggregates the local models from the clients by a normalized averaging function instead of weighted averaging to eliminate the inconsistencies between them. *FedMMB* [58] instruments the clients to perform a limited and constant number of local updates per communication round, which results in more frequent aggregation of the local models on the server, improving the global model utility.

Another category of studies aim to enhance *communication efficiency* using *gradient quantization* [59, 60] and/or *gradient sparsification* [61, 62, 63]. In gradient quantization, the gradient values are quantized using a fewer number of bits to reduce the amount of the traffic transferred in the network. Gradient sparsification, on the other hand, does not communicate some of the gradients (e.g. those with very small values) between the server and clients to alleviate the communication overhead.

The other line of work, referred to as *hybrid federated learning* [64], addresses the privacy challenge in FL by combining it with other techniques such as *secure multi-party computation* (*SMPC*) [35], and/or differential privacy[1] [27]. The aim of the hybrid FL is to hide the original values of the local parameters of the clients from third parties including the server, further enhancing privacy in federated environments.

The *HyFed*[2] framework [65] combines the *additive secret sharing* based SMPC with FL, where clients first mask the values of the local parameters with noise, and then share the noisy parameters with server, and the noise values with compensator. The compensator adds up the noise values, and shares the aggregated noise with the server, which in turn, first aggregates the noisy parameters of the clients, and then subtracts the aggregated noise from the noisy aggregated parameters to obtain the final values of the global parameters, which are identical to those from ordinary FL.

The *sPLINK* [30], *Flimma*[3] [66], and *Fever-PCA*[4] [67] tools are based on the HyFed framework. sPLINK implements the chi-square test and linear/logistic regression models for hybrid federated GWAS; Flimma develops the hybrid federated version of linear regression for differential gene expression analysis; Fever-PCA implements the federated principal component analysis (*PCA*) for population stratification in GWAS.

PySyft [68, 69] is a hybrid FL library introduced by *OpenMined*, which provides a rich application programming interface (API) to develop ordinary FL, SMPC-FL, and DP-FL applications. *FeatureCloud*[5] [70] aims to mitigate the complexity of developing and running federated applications (Apps) by providing a platform equipped with an AI store to publish and reuse Apps. The AI store of FeatureCloud consists of a variety of Apps including federated random forest [71], Kaplan-Meier estimator [72], linear regression [30, 66], logistic regression [30], and neural networks. Akin to HyFed, FeatureCloud employs the additive secret sharing method to conceal the original values of the clients' parameters from the server.

Interestingly, the CUP trade-off is identifiable in the aforementioned studies: The `FedProx` and `FedMMB` algorithms enhance utility at the expense of communication efficiency, which is also the case in our UPFL study [31]. Similarly, the gradient quantization and sparsification techniques provide higher communication efficiency

[1]The related work of DP-FL is discussed later in this section.
[2]HyFed is developed by the author of this thesis.
[3]The author of the dissertation is the joint first author in Flimma.
[4]The thesis's author contributed to Fever-PCA.
[5]The dissertation's author contributed to the FeatureCloud platform.

but lower model utility. The SMPC-FL based tools including sPLINK and Flimma improve privacy, but double the communication overhead compared to ordinary FL.

Differentially private learning (**DP**) [27, 28] *preserves* privacy in both centralized and federated environments, but at the cost of utility. Given that, many of the related studies in the DP area focus on improving the utility of differentially private models by proposing new neural network architectures or training procedures.

Klause et al. [73] introduce a 9-layer ResNet architecture, ResNet-9, where the output of the aggregation operation in the residual blocks is further normalized. *Remerscheid et al.* [74] present a DenseNet-based architecture called *SmoothNet*, which leverages higher number of filters in the dense blocks compared to the original DenseNets. *Cheng et al.* [75] propose a framework dubbed *DPNAS* based on the *neural architecture search* technique to automate the design of differentially private models. *De et al.* [76] employ the *augmentation multiplicity* technique [77], which calculates the per-sample gradients by taking average over the gradients from different augmentations of the sample, in DP settings and show that it significantly improves the accuracy of the differentially private model.

Similar to the above-mentioned studies, we propose a novel kernel normalized residual architecture called KNResNet-13 for DP environments as part of our last study (KernelNorm for privacy-related domains) [33]. We show KNResNet-13 outperforms both SmoothNets and ResNet-9, which are based on GroupNorm, in terms of accuracy on CIFAR-10 and Imagenette. Moreover, we capitalize on a modified version of augmentation multiplicity, which incurs much lower computational overhead compared to the original version, to further improve the state-of-the-art accuracy on CIFAR-10.

Differentially private federated learning (**DP-FL**) enforces the clients to employ differential privacy for hiding the original values of their local parameters from third parties. The main advantage of DP-FL is that it provides a formal privacy guarantee thanks to differential privacy, and thus, it is privacy-preserving unlike SMPC-FL. In the following, we briefly discuss the related work in the DP-FL area.

Wei et al. [78] examine the model convergence behavior in DP-FL environments, and show that for a given privacy budget, (1) higher number of clients participated in training leads to faster convergence rate, and (2) there is an optimal number of communication rounds, which results in optimal model convergence rate. *Kaissis et al.* [79] introduce *PriMIA* as an open-source DP-FL framework to train deep convolutional networks for medical imaging applications in a federated fashion, while preserving privacy. *Noble et al.* [80] propose *DP-SCAFFOLD*, which is a variant of the

SCAFFOLD [81] algorithm, aiming to cope with the challenge of data heterogeneity across clients in federated environments under differential privacy constraints.

Akin to *Wei et al.* [78], we investigate the model performance in DP-FL environments, but in the context of normalization layers in our last study [33]. We show that the proposed KernelNorm layer significantly outperforms the competitors including LayerNorm and GroupNorm in terms of both accuracy and communication efficiency for a given privacy budget. Moreover, we illustrate how to break the CUP trade-off in DP-FL settings using a kernel normalized ResNet in this dissertation (Chapter 1).

Summary of Contributions

We make the following contributions in this dissertation:

1. We introduce the sPLINK tool for GWAS, implementing the hybrid federated versions of the chi-square test and linear/logistic regression models [30].

2. We theoretically and experimentally demonstrate that sPLINK achieves optimal utility for the aforementioned models independent of the data distribution across the clients, which is not the case for meta-analysis as the main competitor [30].

3. We analytically prove and experimentally validate that the DNN models can also achieve optimal utility in federated settings similar to the regression models, and pinpoint the necessary conditions to this end [31].

4. We investigate the properties of different training algorithms, model layers, loss functions, and optimizers to determine which one(s) satisfy the necessary conditions for UPFL [31].

5. We propose two batch-independent layers called KernelNorm and KNConv, and incorporate them into KNConvNets in general, and KNResNets in particular while forgoing the BatchNorm layers [32, 33].

6. Through extensive experiments, we show that KNResNets deliver higher or highly competitive accuracy compared to the batch normalized ResNets for image classification and semantic segmentation in centralized training. KNResNets, moreover, significantly outperform the batch-independent competitors including LayerNorm and GroupNorm based counterparts [32].

7. We draw a detailed performance comparison among KernelNorm, LayerNorm, GroupNorm, and NoNorm (no normalization) in FL, DP, and DP-FL environments, and show that KernelNorm based models achieve considerably higher accuracy and communication efficiency (convergence rate) than the competitors in all three considered environments [33].

8. We propose a kernel normalized residual architecture, KNResNet-13, and provide the state-of-the-art accuracy values on CIFAR-10 and Imagenette in DP environments, when trained from scratch [33].

9. Through an elegant experiment, we illustrate that we can break the CUP trade-off using KernelNorm-based models, enhancing communication, utility, and privacy simultaneously in DP-FL environments (Chapter 1).

PART II

PUBLICATIONS

sPLINK: a Hybrid Federated Tool as a Robust Alternative to Meta-analysis in Genome-wide Association Studies

Reza Nasirigerdeh, Reihaneh Torkzadehmahani, Julian Matschinske, Tobias Frisch, Markus List, Julian Späth, Stefan Weiss, Uwe Völker, Esa Pitkänen, Dominik Heider, Nina Kerstin Wenke, Georgios Kaissis, Daniel Rueckert, Tim Kacprowski & Jan Baumbach

Journal: Genome Biology

Synopsis: Genome-wide association studies (GWAS) examine millions of single nucleotide polymorphisms (SNPs) to identify the association between a particular SNP and given disease. Prior studies illustrate that larger GWAS datasets lead to more accurate results. However, it is a daunting challenge to procure large-scale GWAS datasets in a centralized manner. This is because the data is distributed across different sites such as hospitals, and it is almost impossible to move the data to a centralized location due to privacy regulations and concerns. To address this challenge, we introduce a federated tool called ***sPLINK*** (safe *PLINK*), which performs GWAS on distributed datasets without moving the private data off-site. sPLINK implements the federated versions of three popular models in GWAS, i.e. chi-square, linear regression, and logistic regression. sPLINK also employs secure multi-party computation to hide the original values of the local parameters of the clients from third parties including the server. We theoretically show and experimentally validate that sPLINK achieves *ideal utility*, which is identical to utility from centralized training using PLINK, independent of the data distribution across the clients (sites). This is not the case for meta-analysis as our main competitor, which aggregates the statistics from multiple GWAS. Moreover, we demonstrate sPLINK is highly *efficient* from the *network communication* perspective, requiring a few communication rounds to

conduct GWAS. In summary, sPLINK integrates the *privacy-enhancing* property of meta-analysis with the *ideal performance* benefit of centralized learning.

Contributions of thesis author: Leading role in gathering software resources, developing algorithms, implementing the software components, conducting the experiments, writing the manuscript, and significant contribution in scientific findings.

Publisher: Springer Nature

Date: 24.01.2022

Copyright: Open access, the copyright is retained by the authors.

Reprint permission: According to the publisher website, "This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made."

Nasirigerdeh *et al. Genome Biology* (2022) 23:32
https://doi.org/10.1186/s13059-021-02562-1

Genome Biology

METHOD

sPLINK: a hybrid federated tool as a robust alternative to meta-analysis in genome-wide association studies

Reza Nasirigerdeh[1,2*], Reihaneh Torkzadehmahani[1], Julian Matschinske[3], Tobias Frisch[4], Markus List[5], Julian Späth[3], Stefan Weiss[6], Uwe Völker[6], Esa Pitkänen[7,8], Dominik Heider[9], Nina Kerstin Wenke[3], Georgios Kaissis[1,2,12,13], Daniel Rueckert[1,2,12], Tim Kacprowski[10,11†] and Jan Baumbach[3,4†]

*Correspondence:
reza.nasirigerdeh@tum.de
†Tim Kacprowski and Jan Baumbach are joint senior authors.
[1]AI in Medicine and Healthcare, Technical University of Munich, Munich, Germany
[2]Klinikum rechts der Isar, Munich, Germany
Full list of author information is available at the end of the article

Abstract
Meta-analysis has been established as an effective approach to combining summary statistics of several genome-wide association studies (GWAS). However, the accuracy of meta-analysis can be attenuated in the presence of cross-study heterogeneity. We present *sPLINK*, a hybrid federated and user-friendly tool, which performs privacy-aware GWAS on distributed datasets while preserving the accuracy of the results. *sPLINK* is robust against heterogeneous distributions of data across cohorts while meta-analysis considerably loses accuracy in such scenarios. *sPLINK* achieves practical runtime and acceptable network usage for chi-square and linear/logistic regression tests. *sPLINK* is available at https://exbio.wzw.tum.de/splink.

Keywords: sPLINK, PLINK, Federated learning, Genome-wide association studies, GWAS, Meta-analysis, Privacy

Background

Genome-wide association studies (GWAS) test millions of single nucleotide polymorphisms (SNPs) to identify possible associations between a specific SNP and disease [1]. They have led to considerable achievements over the past decade including better comprehension of the genetic structure of complex diseases and the discovery of SNPs playing a role in many traits or disorders [2, 3]. GWAS sample size is an important factor in detecting associations, and larger sample sizes lead to identifying more associations and more accurate genetic predictors [2, 4].

PLINK [5] is a widely used open source software tool for GWAS. The major limitation of *PLINK* is that it can only perform association tests on local data. If multiple cohorts want to conduct collaborative GWAS to take advantage of larger sample sizes, they can pool their data for a joint analysis (Fig. 1a); however, this is close to impossible due to privacy

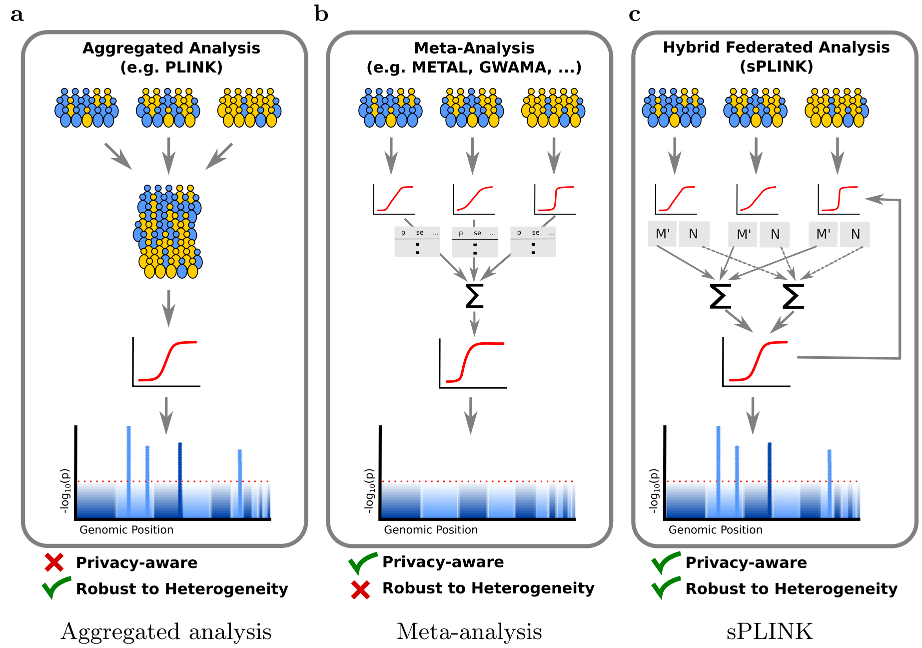


Fig. 1 Comparison of *sPLINK* (**c**), aggregated analysis (**a**), and meta-analysis (**b**) approaches: Aggregated analysis requires cohorts to pool their private data for a joint analysis. The meta-analysis approaches aggregate the summary statistics from the cohorts to estimate the combined *p*-values. In *sPLINK*, the cohorts calculate the model parameters (*M*) from the local data and global model, generate noise (*N*), and make the parameters noisy (*M*′) in an iterative manner. The aggregated noise and noisy parameters are in turn aggregated to update the global model or build the final model. *sPLINK* combines the advantages of the aggregated analysis and meta-analysis, i.e. robustness against heterogeneous data and enhancing the privacy of cohorts' data. Yellow/blue color indicates case/control samples

restrictions and data protection issues, especially concerning genetic and medical data. Hence, the field has established methods for meta-analysis of individual studies, where only the results and summary statistics of the individual analyses have to be exchanged [6] (Fig. 1b).

There are several software packages such as *METAL* [7], *GWAMA* [8], and *PLINK* [5] that implement different meta-analysis models including fixed or random effect models [9]. Although meta-analysis approaches are privacy-aware, i.e. the raw data is not shared with third parities, they suffer from two main constraints: first, they rely on detailed planning and agreement of cohorts on various study parameters such as meta-analysis model (e.g. fixed effect or random effect), meta-analysis tool (e.g., METAL or GWAMA), heterogeneity metric (e.g. Cochran's Q or the I^2 statistic), the covariates to be considered, etc [4]. Second and more importantly, the statistical power of meta-analysis can be adversely affected in the presence of cross-study heterogeneity, leading to inaccurate estimation of the joint results and yielding misleading conclusions [10, 11].

To address the aforementioned shortcomings, privacy-aware collaborative GWAS can be developed using homomorphic encryption (HE) [12], secure multi-party computation (SMPC) [13], and federated learning [14, 15]. In HE, the cohorts encrypt their private data and share it with a single server, which performs operations on the encrypted data from the cohorts to compute the association test results. In SMPC, there are several computing parties and the cohorts extract a separate secret share (anonymized chunk) [16] from the private data and send it to a computing party. The computing parties calculate intermediate results from the secret shares and exchange the intermediate results with each other. Each computing party computes the final results given all intermediate results. In federated learning, the cohorts extract model parameters (e.g. Hessian matrices) from the private data and share the parameters with a central server. The server aggregates the parameters from all cohorts to calculate the association test results.

Kamm et al. [17] and Cho et al. [18] proposed GWAS frameworks based on SMPC. The former developed simple association tests including Cochran–Armitage and chi-square (χ^2) and the latter implemented only the Cochran–Armitage test for trend. Shi et al. [19] presented an SMPC-based logistic regression framework for GWAS. Constable et al. [20] implemented an SMPC-based framework for minor allele frequency and chi-square computation. These frameworks inherit the limitations of SMPC itself: They follow the paradigm of "move data to computation," where they put the processing burden on a few computing parties. Consequently, they are computationally expensive [21] and are not scalable for large-scale GWAS. Moreover, they suffer from the colluding-parties problem [17] in which, if the parties send the secret shares of the cohorts to each other, the whole private data of the cohorts is exposed.

Lu et al. [22], Morshed et al. [23], and Kim et al. [24] developed chi-square, linear regression, and logistic regression tests using HE for GWAS, respectively. Sadat et al. [25] introduced the *SAFETY* framework based on HE and Intel Software Guard Extensions technology, which implements the linkage disequilibrium, Fisher's exact test, Cochran-Armitage test for trend, and Hardy-Weinberg equilibrium statistical tests. Similar to SMPC-based methods, they are not computationally efficient because a single server carries out operations over encrypted data, causing considerable overhead [26]. Additionally, HE-based methods introduce accuracy loss in the association test results [23, 24]. This is because HE only supports addition and multiplication, and as a result, non-linear operations in regression tests should be approximated using those two operations.

To address the computational limitation of HE/SMPC-based methods, the association tests can be implemented in a federated fashion. Federated learning-based methods follow the paradigm of "move computation to data," distributing the heavy computations among the cohorts while performing lightweight aggregation (simple operations such as addition and multiplication of the parameters) at the central server. Wang et al. [27] introduced EXPLORER for distributed logistic regression algorithm. EXPLORER is a model but not a tool for GWAS. Moreover, it does not provide a "guarantee for optimal global solution," implying that its results can be different from the aggregated analysis in general. GLORE [28, 29] implemented a federated logistic regression test but the parameter values computed by each cohort are revealed to the server.

Several hybrid federated frameworks including *HyFed* [30] have been introduced to improve the privacy of federated learning by hiding the local parameters of a cohort from third parties. HyFed is a suitable framework for developing federated GWAS algorithms because it provides enhanced privacy while preserving the accuracy of the results. It also supports federated mode, where different components can run in separate physical machines and securely communicate with each other over the Internet.

In this paper, we present a hybrid federated tool called *sPLINK (safe PLINK)* based on the *HyFed* framework for privacy-aware GWAS. *sPLINK* consists of four main components (Fig. 2): *Web application (WebApp)* to configure the parameters (e.g. association

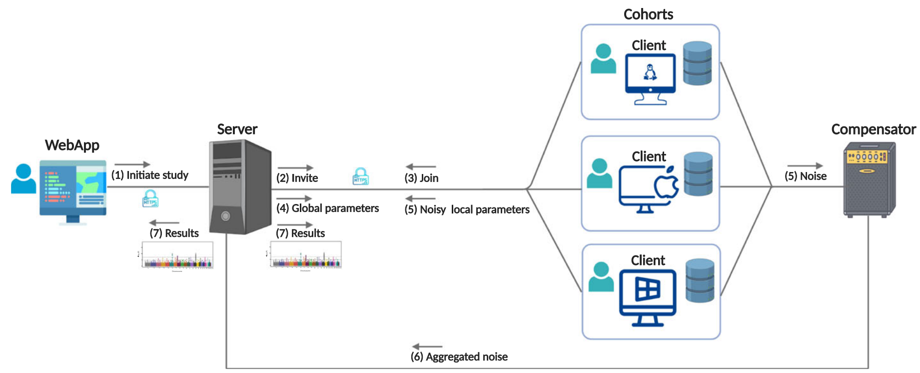


Fig. 2 *Architecture of sPLINK* : (1) The coordinator creates a new project through the WebApp component and (2) invites a set of cohorts to join the project; (3) the cohorts join the project and select the dataset using the client component. The project is started automatically, when all cohorts joined. The computation of the test results is performed in a an iterative manner, where the clients (4) obtain the global parameters from the server, (5) compute the local parameters, mask them with noise, and share the noise and noisy local parameters with the compensator and server, respectively; (6) the compensator aggregates the noise values and sends the aggregated noise to the server; the server calculates the global parameters by aggregating the noisy local parameters and the negative of the aggregated noise; (7) after the computation is done, the cohorts and coordinator can access the results. All communications are performed in a secure channel over HTTPS protocol. The cohorts can use Linux distributions, Microsoft Windows, or MacOS to run the client component

test) of the new study; *client* to compute the local parameters, mask them with noise, and share the noise with *compensator* and noisy local parameters with *server*; *compensator* to aggregate the noise values of the clients and send the aggregated noise to the *server*; *server* to compute the global parameters by adding up the noisy local parameters and the negative of the aggregated noise. Notice that the utility of the global model is preserved because the aggregated noise from the compensator cancels out the accumulated noise from the noisy local parameters during the aggregation.

Unlike *PLINK*, *sPLINK* is applicable to distributed data in a privacy-aware fashion. In *sPLINK*, neither the private data of cohorts leaves the site nor the original values of the local parameters are revealed to the other parties (Fig. 1c). Contrary to the existing HE/SMPC-based methods, *sPLINK* is computationally efficient because heavy computations are distributed across the cohorts while simple aggregation is performed on the server and compensator. Compared to the current federated tools like GLORE, *sPLINK* not only provides enhanced privacy but also supports multiple association tests including logistic and linear regression [31], and chi-square [32] for GWAS.

The advantage of *sPLINK* over the meta-analysis approaches is twofold: usability and robustness against heterogeneity. *sPLINK* is easier to use for collaborative GWAS compared to meta-analysis. In *sPLINK*, a coordinator initiates a collaborative study and invites the cohorts. The only decision the cohorts make is whether or not to join the study. After accepting the invitation, the cohorts just select the dataset they want to employ in the study. More importantly, *sPLINK* is robust to data heterogeneity (phenotype and confounding factors). It gives the same results as aggregated analysis even if the phenotype distribution is imbalanced or if confounding factors are distributed heterogeneously across cohorts. In contrast, meta-analysis tools typically lose statistical power in such imbalanced or heterogeneous scenarios (details in the "Results" section).

Results

We first verify *sPLINK* by comparing its results with those from aggregated analysis conducted with *PLINK* for all three association tests on a real GWAS dataset from the SHIP study [33]. We refer to this dataset as the *SHIP* dataset, which comprises the records of 3699 individuals with *serum lipase activity* as phenotype. The quantitative version represents the square root transformed serum lipase activity, while the dichotomous (binary) version indicates if the serum lipase activity of an individual is above or below the 75th percentile. The *SHIP* dataset contains around 5 million SNPs as well as sex, age, smoking status (current-, ex-, or non-smoker), and daily alcohol consumption (in g/day) as confounding factors (Table 1).

We employ the binary phenotype for logistic regression and the chi-square test, and the quantitative phenotype for linear regression. We incorporate all four confounding factors in the regression models and no confounding factor in the chi-square test. We horizontally (sample-wise) split the dataset into four parts, simulating four different cohorts (Additional file 1: Table S1). *PLINK* computes the statistics for each association test using the whole dataset while *sPLINK* does it in a federated manner using the splits of the individual cohorts. To be consistent with *PLINK*, *sPLINK* calculates the same statistics as *PLINK* for the association tests.

We compute the difference between the *p*-values as well as the Pearson correlation coefficient (ρ) of *p*-values from *sPLINK* and *PLINK*. We use $-log_{10}$(*p*-value) because the *p*-values are typically small and $-log_{10}$(*p*-value) can be a better indicator of small *p*-value differences. According to Fig. 3a–c, the *p*-value difference is zero for most of the SNPs. We also observe that the maximum difference is 0.162 for a SNP in the linear regression. *sPLINK* and *PLINK* report 4.441×10^{-16} and 3.058×10^{-16} as *p*-values for the SNP, respectively. This negligible difference can be attributed to inconsistencies in floating point precision.

The correlation coefficient of *p*-values from *sPLINK* and *PLINK* for all three tests is $0.\overline{99}$, which is consistent with the results of *p*-value difference from Fig. 3a–c. We investigate the overlap of significantly associated SNPs between *sPLINK* and *PLINK*. We

Table 1 Description of datasets

Dataset	# Samples	# SNPs	Adjustments	Phenotype
SHIP[a]	3699	∼5M	Sex, age, smoking status, daily alcohol consumption	SLA[b], dichotomous (75th percentile, 934 cases, 2765 controls)
				SLA, quantitative, Mean±SD[c] 1.23±0.3
COPDGene[d]	5343	∼600K	Sex, age, smoking status, pack years of smoking	COPD[e], dichotomous, (2811 cases, 2532 controls)
				FEV1[f], quantitative, Mean±SD 2.993±0.635
FinnGen	135,615	∼ 1M	Sex and age	Hypertension, dichotomous, (34,257 cases, 101,358 controls)

[a] Study of Health in Pomerania
[b] Serum lipase activity
[c] Standard deviation
[d] Genetic Epidemiology of chronic obstructive pulmonary disease
[e] Chronic obstructive pulmonary disease
[f] Forced expiratory volume in one second

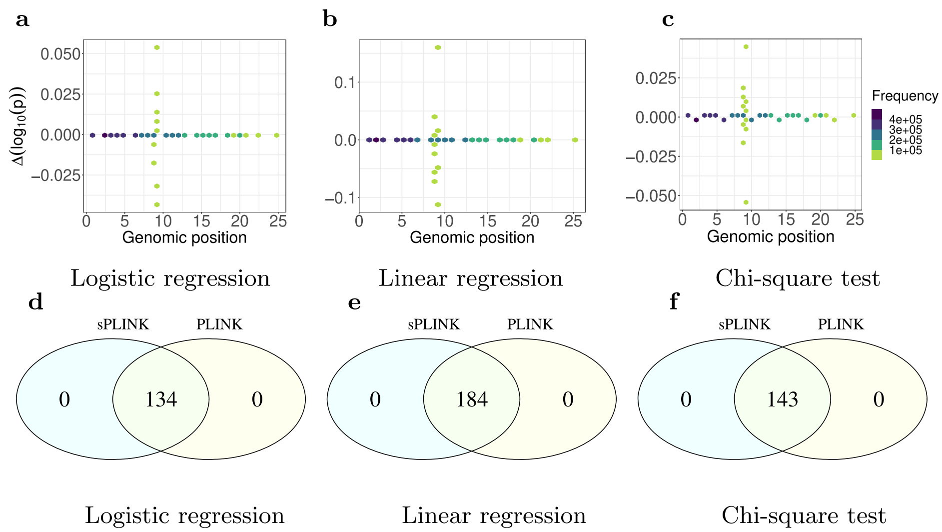


Fig. 3 Δlog_{10}(*p*-value) between *sPLINK* and *PLINK* as well as the set of SNPs identified by *sPLINK* and *PLINK* as significant for logistic regression (**a**, **d**), linear regression (**b**, **e**), and chi-square test (**c**, **f**), respectively. For most of the SNPs, the difference is zero, indicating that *sPLINK* gives the same *p*-values as *PLINK*. The negligible difference between *p*-values for the other SNPs can be attributed to differences in floating point precision. The spikes in some genomic positions are due to the strong association of the corresponding SNPs, which result in higher absolute error. *sPLINK* and *PLINK* also recognize the same set of SNPs as significant. Genomic positions (ticks in **a**–**c**) indicate chromosome numbers. The details of the experiments are available in Additional file 1: Table S1

consider a SNP as significant if its *p*-value is less than 5×10^{-8} (genome-wide significance). *PLINK* and *sPLINK* recognize the same set of SNPs as significant (Fig. 3d–f). Notably, the identified SNPs, e.g. rs8176693 and rs632111, lying in genes ABO (intronic) and FUT2 (3-UTR), respectively, have also been implicated in a previous analysis of this dataset [34]. We also leverage the Bonferroni significance threshold (which is $\approx 1 \times 10^{-8}$ for our tests) to compare the overlapping significant SNPs from *sPLINK* and *PLINK*. The results remain similar and the associated plot is available at Additional file 1: Fig. S1. These results indicate that *p*-values computed by *sPLINK* in a federated manner are the same as those calculated by *PLINK* on the aggregated data (ignoring negligible floating point precision error). In other words, the federated computation in *sPLINK* preserves the accuracy of the results of the association tests.

Next, we compare *sPLINK* with some existing meta-analysis tools, namely *PLINK*, *METAL*, and *GWAMA*. We leverage the *COPDGene* (non-hispanic white ethnic group) [35] and *FinnGen* (data release 3) [36] datasets. The *COPDGene* dataset has an equal distribution of case and control samples unlike the *SHIP* dataset. It contains 5343 samples (ignoring 1327 samples with missing phenotype value) and around 600K SNPs. We utilize chronic obstructive pulmonary disease (COPD) as the binary phenotype and include sex, age, smoking status, and pack years of smoking as confounding factors [37]. *FinnGen* is much larger dataset (in terms of sample size) compared to the *SHIP* and *COPDGene* datasets. It consists of 135,615 samples (ignoring 23 samples with missing phenotype value) and about 1 million SNPs. We use *Hypertension* as the (binary) phenotype and adjust for sex and age as confounding factors (Table 1).

To simulate cross-study heterogeneity [38] on the *COPDGene* dataset, we consider six different scenarios: *Scenario I* (*Balanced*), *Scenario II* (*Slightly Imbalanced*), *Scenario III*

(*Moderately Imbalanced*), *Scenario IV* (*Highly Imbalanced*), *Scenario V* (*Severely Imbalanced*), and *Scenario VI* (*Heterogeneous Confounding Factor*) (Figs. 4a and 5). In each scenario, we partition the dataset into three splits with the same sample size (more details in Additional file 1: Table S2). The distribution of all four confounding factors is homogeneous (similar) across the splits for the first five scenarios. The splits have the same (and balanced) case-control ratio in *Scenario I* and *Scenario VI* but their case-control ratio is different for the imbalanced scenarios (Fig. 4a). In *Scenario VI*, the values of two confounding factors (i.e. smoking status and age) are homogeneously distributed among the splits; however, the distribution of sex and pack years of smoking is slightly and highly heterogeneous across the splits, respectively (Fig. 5). We obtain the summary statistics (e.g. minor allele, odds ratio, and standard error) for each split to conduct meta-analyses. The results are then compared to the federated analysis employing *sPLINK*. Figure 6a shows the Pearson correlation coefficient of $-log_{10}$(p-value) between each tool and the aggregated analysis for all six scenarios. Figure 6c depicts the number of SNPs correctly identified as significant by the tools (true positives).

According to Fig. 6a, the correlation of p-values between *sPLINK* and the aggregated analysis is $\sim$ 1.0 for all six scenarios, implying that *sPLINK* gives the same p-values as the aggregated analysis regardless of how phenotypes or confounding factors have been distributed across the cohorts. In contrast, the correlation coefficient for the meta-analysis tools shrinks with increasing imbalance/heterogeneity, indicating loss of accuracy. Figure 6c illustrates that *sPLINK* correctly identifies all four significant SNPs in all scenarios. In the balanced scenario, almost all meta-analysis tools perform well and recognize all significant SNPs. An exception is *METAL*, which misses one of them. However, they miss more and more significant SNPs as the phenotype imbalance across the splits increases. In the *Highly Imbalanced* and *Severely Imbalanced* scenarios, the meta-analysis tools cannot recognize any significant SNP. This is also the case if the distribution of some confounding factors becomes heterogeneous across the cohorts (*Scenario VI*). We checked the number of SNPs wrongly identified as significant by the tools (false positives) too. *sPLINK* has no false positive in any of the scenarios and the meta-analysis tools introduce zero or one false positive depending on the scenario.

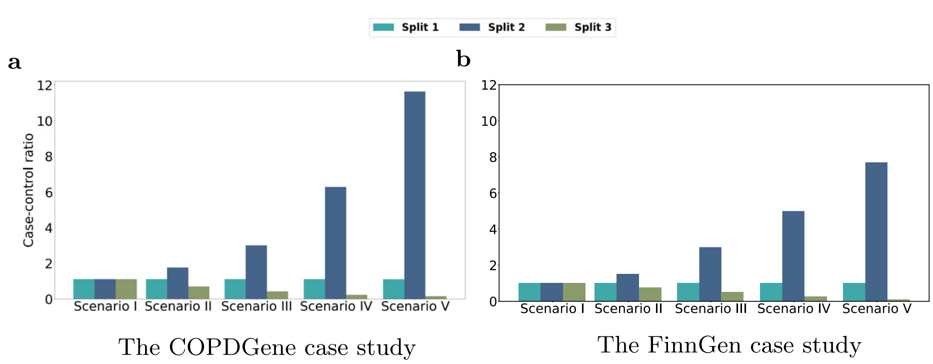


Fig. 4 *Scenario I-V*: The case-control ratio is the same for all splits in the balanced scenario (I) while the splits have different case-control ratios in the imbalanced scenarios (II–V). All three splits have the same sample size in the *COPDGene* dataset as well as the balanced scenario in the *FinnGen* dataset. For the imbalanced scenarios in the *FinnGen* dataset, the splits have different sample sizes

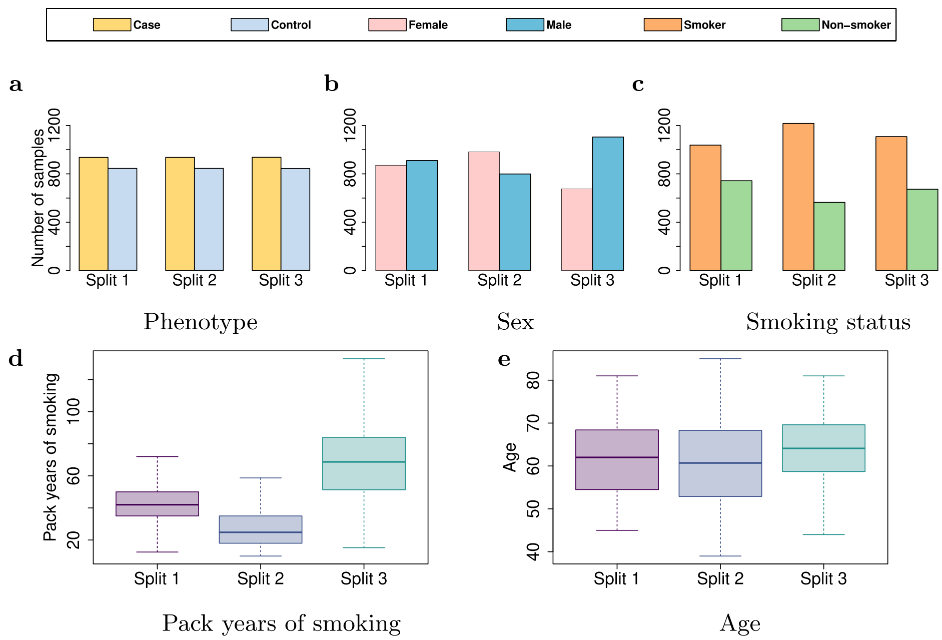


Fig. 5 *Scenario VI (Heterogeneous Confounding Factor)* for the *COPDGene* case study: The phenotype distribution is the same and balanced; the values of smoking status and age are homogeneously distributed; the distribution of sex and pack years of smoking are slightly and highly heterogeneous across the splits, respectively

To show that our findings on the *COPDGene* dataset also hold true for a much larger dataset, we repeat the simulations on the *FinnGen* dataset (more details in Additional file 1: Table S3). Similar to the *COPDGene* case study, we divide the dataset into three splits and define *Scenario I* to *Scenario V*, where the splits have the same case-control ratio (1.0) and sample size (22,838) as in *Scenario I* but different case-control ratios in the remaining scenarios (Fig. 4b); Unlike the *COPDGene* case study in which the sample size of the splits are equal for all scenarios including the imbalanced ones, the splits have different number of samples in the imbalanced scenarios of the *FinnGen* case study. For instance, split1, split2 and split3 have 22,838, 12,561, and 99,345 samples in *Scenario V*, respectively (a split with lower case-control ratio has larger sample size). It implies that the aggregated datasets have different number of samples in the scenarios, and as a result, there are different set of significant SNPs in each scenario of the *FinnGen* case study (total of 110, 116, 199, 304, and 446 significant SNPs in *Scenario I* to *Scenario V*, respectively).

Figures 6b and 6d illustrate the Pearson correlation coefficient and percentage of correctly identified significant SNPs for each scenario on the *FinnGen* case study, respectively. According to Fig. 6b, the correlation coefficient diminishes for the meta-analysis tools as the scenario becomes more and more imbalanced. This is also the case for the percentage of the SNPs correctly identified as significant by each meta-analysis tool (Fig. 6d). These results are consistent with those from the *COPDGene* case study. Moreover, we observed that the meta-analysis tools report high number of false positives (14–88) in *Scenario IV*. Thus, the limitations of meta-analysis tools towards class imbalance observed in the *COPDGene* dataset can be reproduced on a large dataset. However, sPLINK always provides the same results as PLINK with the aggregated analysis (the "Methods" section, Figs. 3 and 6a, c).

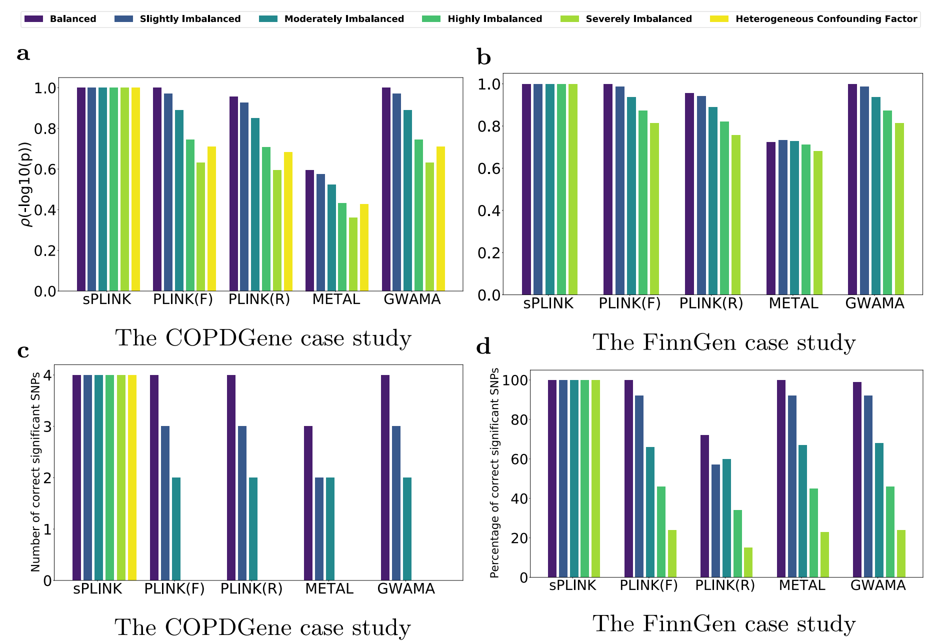


Fig. 6 The Pearson correlation coefficient (ρ) of $-log_{10}$(p-value) between each tool and aggregated analysis (**a**, **b**) and the number (**c**) and the percentage (**d**) of SNPs correctly identified as significant (true positives) by each tool. *F* and *R* stand for fixed-effect and random-effect, respectively. The details of the experiments are available in Additional file 1: Table S2, and Table S3

We also leverage the Spearman correlation to check whether or not the meta-analysis tools maintain the ordering of significance compared to the aggregated analysis. Our results show that this is not the case, and the Spearman correlation values for the meta-analysis tools reduce as the phenotype imbalance across the splits increases, similar to the results from Fig. 6, where the Pearson correlation is used. The corresponding plot can be found in Additional file 1: Figure S2.

Table 2 shows a concise comparison between *sPLINK* and the state-of-the-art approaches. Unlike *PLINK*, *sPLINK* is privacy-aware, where the private data never leaves the cohorts. *sPLINK* is also robust against the imbalance/heterogeneity of phenotype/confounding factor distributions across the cohorts. *sPLINK* always delivers the same p-values as aggregated analysis and correctly identifies all significant SNPs independent of the phenotype or confounding factor distribution in the cohorts. In contrast, meta-analysis tools lose their statistical power in imbalanced phenotype scenarios, missing some or all significant SNPs. This is also the case if the phenotype distribution is balanced but the values of confounding factor(s) have heterogeneously been distributed across the datasets. Compared to the existing SMPC/HE-based approaches, *sPLINK* is computationally efficient and supports multiple association tests including chi-square and linear/logistic regression. *sPLINK* provides enhanced privacy by hiding the model parameters of each cohort from the third parties while federated learning-based frameworks such as GLORE reveal them to the server.

Finally, we measure the runtime and network bandwidth usage of *sPLINK* for each association test using the COPDGene dataset partitioned into three splits of the same sample

Table 2 Comparison between *sPLINK* and the state-of-the-art approaches

Tool/Study	Privacy-aware	Robust to heterogeneity	Computationally efficient	Linear regression	Logistic regression
PLINK	✗	✓	✓	✓	✓
Meta-analysis	✓	✗	✓	✓	✓
Kamm et al. [17]	✓	✓	✗	*	✗
Cho et al. [18]	✓	✓	✗	*	✗
Morshed et al. [23]	✓	✗	✗	✓	✗
Kim et al. [24]	✓	✗	✗	✗	✓
GLORE [28]	✓	✓	✓	✗	✓
sPLINK	✓	✓	✓	✓	✓

*The study supports the Cochran–Armitage test, which is computationally comparable to linear regression

size. We use *COPD* in chi-square as well as logistic regression and *FEV1* in linear regression as phenotype. We include age, sex, smoking status, and pack years of smoking as confounding factors only for the regression tests. The server and WebApp packages are installed on a physical machine located at *Freising* (*Germany*) while the compensator is running on a machine at *Odense* (*Denmark*). Three commodity laptops located at *Munich* or *Freising* are running the client package and host the splits. They communicate with the server and compensator through the Internet. The system specification of the machines and laptops as well as the details of the experiments can be found in Additional file 1: Table S4 and S5.

Figure 7a plots the *sPLINK's* runtime for each association test. *sPLINK* computes the results for chi-square, linear regression, and logistic regression in 8 min, 20 min, and 75 min, respectively. Sending parameters from the clients to the server and compensator contributes the most in sPLINK's runtime. Compared to Kamm et al. [17], *sPLINK* is almost 13 times faster for chi-square test (8 min vs. 110 min[1]) with less powerful hardware, larger sample size (5343 vs. 1080), and more number of SNPs ($\sim$ 580K vs. $\sim$ 263K).

Figure 7b depicts the network usage of *sPLINK*. The clients, server, and compensator exchange total of 0.967 GB, 2.49 GB, and 11.06 GB traffic in chi-square, linear regression, and logistic regression, respectively. Logistic regression has higher volume of traffic

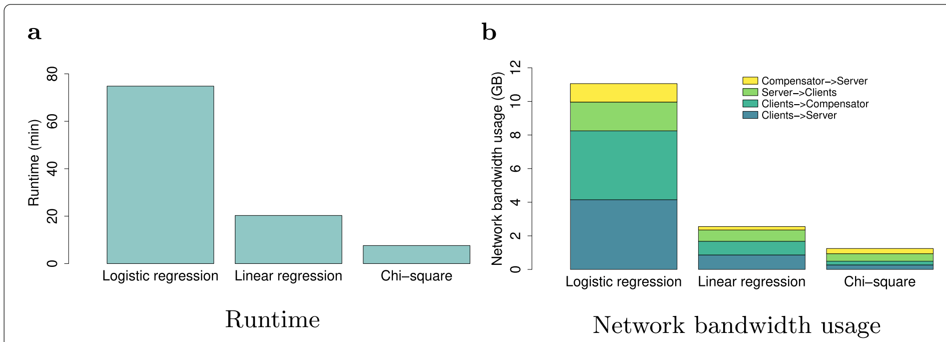


Fig. 7 Runtime and network bandwidth consumption of *sPLINK*. Logistic regression is the most time-consuming association test and exchanges the highest traffic over the network due to the iterative nature of the algorithm. The experimental setup can be found in Additional file 1: Table S5

[1]The best result from Kamm et al. [17] has been considered.

exchange because the computation of beta coefficients are performed in an iterative fashion. A fair comparison between *sPLINK* and SMPC-based frameworks from the network communication aspect is tricky. However, in general, (hybrid) federated learning-based approaches consume more network bandwidth than SMPC-based ones.

We also conduct a set of experiments to investigate how the runtime and network bandwidth consumption of *sPLINK* change with varying number of samples, SNPs, and clients. The results demonstrate that the traffic exchanged over the network is independent of the sample size and linearly increases with the number of SNPs and clients (as expected). Moreover, runtime is not affected much by the sample size thanks to the multi-threading capability of *sPLINK*'s client package, and linearly/non-linearly increases with the number of SNPs/clients. The corresponding plots are available in Additional file 1: Fig. S3, S4, and S5.

Discussion

We first provide a general discussion on the privacy of the existing tools for collaborative GWAS including *sPLINK*. To be more accurate, we draw a distinction between the privacy-aware and privacy-preserving definitions [39]. In a privacy-aware approach, it is not required to share the private data with a third party. A privacy-aware approach is privacy-preserving if the approach offers a privacy guarantee that captures the privacy risk associated with individual samples in the dataset. Given that, meta-analysis, SMPC, HE, federated learning, and hybrid federated learning based on SMPC are privacy-aware because they do not share the raw data with a third party. In meta-analysis/federated learning, the summary statistics/model parameters of each cohort are shared with a third party. In SMPC-based hybrid federated learning, the aggregated (global) parameters are revealed to the server and cohorts. These approaches, including HE and SMPC, reveal the final model too. However, these methods are not privacy-preserving because none of them provides a privacy guarantee indicating to what extent the revealed information leaks the private data of a particular sample in the dataset. To our knowledge, differential privacy (DP) [40] and DP-based hybrid federated learning can offer such a guarantee at the cost of the utility of the model and are considered as privacy-preserving approaches.

While privacy-aware approaches do not offer a privacy guarantee, they might provide stronger/weaker privacy compared to each other based on the amount and nature of the information they share with third parties. For instance, HE-based methods provide stronger privacy because they only reveal the final model (results) while other privacy-aware approaches disclose not only the final results but also other information such as summary statistics or local parameters. Similarly, *sPLINK* provides enhanced privacy in comparison with existing federated learning based tools such as GLORE. This is because GLORE discloses the local parameters of each cohort to the server, which is not revealed in *sPLINK*.

sPLINK is a privacy-aware tool, assuming honest-but-curious server, compensator, and clients, which (I) follow the protocol as it is; for instance, the server always sends the global beta values resulted from the aggregation but not the beta values tampered with such as all zeros to the clients, and (II) do not collude with each other, e.g. the compensator never shares the individual noise values of the clients with the server and similarly, the server does not send the noisy local parameters to the compensator, but (III) they try to reconstruct the raw data using the model parameters. Additionally, (IV) there are at least three different cohorts participating in the study, and their client components as well as the server and compensator components are running in separate physical machines.

Given these assumptions, we discuss the privacy of the masking mechanism of *sPLINK* (inherited from *HyFed*) for the supported association tests. To this end, we use the information theoretic criterion called *mutual information* between two random variables X and Y [30, 41]:

$$I(X, Y) = H(X) - H(X|Y)$$

where $H(X)$ and $H(X|Y)$ indicate the entropy of X and the conditional entropy of X given Y, respectively. The mutual information measures (in bits) the decrease in uncertainty about X having the knowledge of Y. In *sPLINK*, the noisy local parameter M'_L is a secret share from the local parameter M_L (the secret), and random variables X and Y indicate the distributions of M_L and M'_L, respectively.

The local parameter M_L of a client is either a non-negative integer (e.g. sample count, allele count, or contingency table) or floating-point number (e.g. Hessian or covariance matrix) in the association tests. For non-negative integers, *sPLINK* capitalizes on *additive secret sharing* based on *modular arithmetic* over the finite field $\mathbb{Z}_p$={0, 1, $p-1$}, in which p is a *prime* number [13]. For floating-point numbers, *sPLINK* employs *real value secret sharing* based on Gaussian (Normal) distribution [42, 43] (more details in "Methods" section).

For non-negative integers, noise N_L is generated from a uniform distribution over $\mathbb{Z}_p$, and M'_L is the modular addition of M_L and N_L: $M'_L = (M_L + N_L) \mod p$. For this scheme, it has been shown that the knowledge of Y (noisy local parameter) provides no information about X (local parameter), which means the mutual information between them is zero: $I(X, Y) = 0$ [13, 16]. Notice that this is the case for any value of prime number p.

For floating-point numbers, noise N_L is generated using Gaussian distribution with variance of σ_N^2. Assuming that the variance of X is $\sigma_{M_L}^2$, the mutual information between X and Y is maximum if Y follows the Gaussian distribution (variance $\sigma_{M_L}^2 + \sigma_N^2$) [43]. Thus, the upper bound on the mutual information between X and Y is:

$$I(X, Y) = \frac{1}{2}\log_2(1 + \frac{\sigma_{M_L}^2}{\sigma_N^2})$$

That is, the amount of reduction in uncertainty about the local parameters having the knowledge of the noisy local parameters depends on the relative variance of the corresponding distributions. Therefore, using larger values for variance in the Gaussian random generator will provide lower information leakage. The value of mean for the Gaussian random generator does not remarkably impact the privacy and can be set to zero [43], which is the case for *sPLINK*. The default value of σ_N^2 is 10^{12} for *sPLINK*, which is large enough for typical GWAS, but it can be set to higher values if needed to ensure that $\frac{\sigma_{M_L}^2}{\sigma_N^2}$ remains small.

Notice that although *sPLINK* significantly enhances the privacy of data compared to existing federated learning tools by hiding the local parameters of clients from a third party, it does not eliminate the possibility of data reconstruction using the aggregated parameters or final results. For example, the $X^T X$ parameter (covariance matrix) in the linear regression algorithm can be exploited to determine the sex of the patients if the total number of samples across all cohorts is comparable to the number of the confounding factors. However, for a reliable GWAS study, the total sample size is considerably larger than the number of confounding factors, and therefore, the reconstruction of the cohorts' private data from the aggregated parameters can be difficult (but still possible) in practice. A similar argument is also applicable to meta-analysis approaches, which reveal the summary statistics of each cohort to a third party.

The value of prime number p impacts the correctness of the masking mechanism. To ensure the correctness, overflow must not occur in $\sum_{i=1}^{i=K} N_{L_i}$ and $\sum_{i=1}^{i=K} M'_{L_i}$ calculations, and $\sum_{i=1}^{i=K} M_{L_i} < p$. *sPLINK* uses the default value of $p = 2^{54} - 33$, which is the largest prime number than can fit in 54-bit integer. A higher value of p can be employed to handle larger integer values but at the expense of a lower number of clients [30]. Likewise, too large values of variance σ_N^2 (e.g. 10^{30}) can impact the precision of the results. With default values of p and σ_N^2, however, our experiments indicate that there are no statistically significant differences between the results from *sPLINK* with and without the masking mechanism for all three association tests (the experimental setup of Fig. 7 is used in the experiments).

sPLINK currently supports chi-square and linear/logistic regression tests, but it can be extended to compute other useful statistics in GWAS such as minor allele frequency (MAF), Hardy-Weinberg equilibrium (HWE), and linkage disequilibrium (LD) between SNPs in a privacy-aware manner. The federated computation of the aforementioned statistics in *sPLINK* is expected to be straightforward because they are based on the allele frequencies, and *sPLINK* already calculates the minor and major allele counts in the *Non-missing count* step of its computational workflow (the "Methods" section). Moreover, population stratification using the principal component analysis (PCA) will be addressed in the future version of *sPLINK* due to the complexity of the problem. *sPLINK*'s implementation of the association tests is horizontally-federated, where the datasets have different samples but the same features (i.e. SNP and confounding factors). However, correcting for population structure using *sPLINK* requires a vertically-federated [44] PCA algorithm because the eigenvectors should be computed from the sample by sample covariance matrix, and therefore, the samples and features swap roles in the federated PCA (SNPs are considered as samples and patients as features) [45]. Vertical federated learning algorithms are still understudied, and they are considered more complicated than the horizontal algorithms.

Additionally, the federated PCA algorithm should be an iterative, randomized algorithm [46] so that it can handle large GWAS datasets with a practical amount of main memory. The iterative nature of the algorithm will present network and runtime challenges because it might need dozens or hundreds of iterations and exchange huge traffic over the network to converge to the final eigenvectors. From the privacy perspective, a recent study [45] demonstrates that even if we assume the federated PCA and linear regression algorithms individually provide perfect privacy, federated population stratification in GWAS, where the eigenvectors are used as the confounding factors in the association test, does not necessarily offer perfect privacy. Consequently, the server can reconstruct the SNP or binary confounding factor values in polynomial time. To tackle this issue, they suggested that the final eigenvectors should be computed at the clients and the model parameter values should be hidden from the server. The federated population stratification in *sPLINK* should be implemented taking into account those suggestions.

We showed that *sPLINK* is robust against an important source of data heterogeneity, namely the heterogeneous distribution of the phenotype or confounding factor values across the distributed datasets of the cohorts. Population heterogeneity across the cohorts is another source of data heterogeneity in GWAS, which is commonly tackled by population stratification using the PCA algorithm. *sPLINK* currently does not address this kind of data heterogeneity but the future versions of the tool will support population stratification to this end.

Conclusions

We introduce *sPLINK*, a user-friendly, hybrid federated tool for GWAS. *sPLINK* enhances the privacy of the cohorts' data without sacrificing the accuracy of the test results. It supports multiple association tests including chi-square, linear regression, and logistic regression. *sPLINK* is consistent with *PLINK* in terms of the input data formats and results. We compare *sPLINK* to aggregated analysis with *PLINK* as well as meta-analysis with *METAL*, *GWAMA*, and *PLINK*. While *sPLINK* is robust against the heterogeneity of phenotype or confounding factor distributions across separate datasets, the statistical power of the meta-analysis tools is declined in imbalanced/heterogeneous scenarios. We argue that *sPLINK* is easier to use for collaborative GWAS compared to meta-analysis approaches thanks to its straightforward functional workflow. We also show that *sPLINK* achieves practical runtime, in order of minutes or hours, and acceptable network usage. *sPLINK* is an open-source tool and its source code is publicly available under the Apache License Version 2.0. *sPLINK* is a novel and robust alternative to meta-analysis, which performs collaborative GWAS in a privacy-aware manner. It has the potential to immensely impact the statistical genetics community by addressing current challenges in GWAS including cross-study heterogeneity and, thus, to replace meta-analysis as the gold standard for collaborative GWAS.

Methods

Federated learning [14, 15] is a type of distributed learning, where multiple cohorts collaboratively learn a joint (global) model under the orchestration of a central server [47]. The cohorts never share their private data with the server or the other cohorts. Instead, they extract local parameters from their data and send them to the server. The server aggregates the local parameters from all cohorts to compute the global model parameters (or global results), which in turn, are shared with all cohorts. While federated learning is privacy-aware, where the private data of the cohorts is not shared with the server, studies [48, 49] have shown that for some models such as deep neural networks, the raw data can be reconstructed from the parameters shared by the cohorts.

To improve the privacy of federated learning, privacy-enhancing technologies (PETs) such as DP, HE, or SMPC can be combined with federated learning to avoid revealing the original values of the local parameters to third parties including the server [50]. DP-based hybrid federated learning approaches can provide a privacy guarantee but their final results might be considerably impacted by the random noise employed for the perturbation of the model. HE-based aggregation methods can incur remarkable computational overhead because they require the cohorts to encrypt/decrypt the local/global model parameters and the server to perform the aggregation over the encrypted parameters. SMPC-based hybrid federated learning methods [30, 51] increase the network bandwidth usage but does not adversely affect the final results. *HyFed* is an open-source hybrid federated framework, which combines federated learning with additive secret sharing-based SMPC to enhance the privacy of the federated algorithms while preserving the utility (performance) of the global model. HyFed provides a generic API (application programming interface) to develop federated machine learning algorithms. It supports the federated mode of operation, where different components of the framework can be installed in separate physical machines and securely communicate with each other through the Internet.

sPLINK implements a hybrid federated approach using the *HyFed* API to enhance the privacy of data. *sPLINK* works with distributed GWAS data, where samples are individuals and features are SNPs and categorical or quantitative phenotypic variables. While the samples are different across the cohorts, the feature space is the same because *sPLINK* only considers SNPs and phenotypic variables that are common among all datasets (horizontal or sample-based federated learning)[44]. The client package of *sPLINK* is installed on the local machine of each cohort with access to the private data. The compensator is running in a separate machine. sPLINK's server and WebApp packages are installed on a central server.

In *sPLINK*, the original values of the parameters computed from the private data in one cohort is not revealed to the server, compensator, or other cohorts, improving the privacy of the cohorts' data. *sPLINK* provides the chunking capability to handle large datasets containing millions of SNPs. The chunk size (configured by the coordinator) specifies how many SNPs should be processed in parallel. Larger chunk sizes allow for more parallelism, and therefore less running time in general but require more computational resources (e.g. CPU and main memory) from the local machines of the cohorts, the server, and compensator. *sPLINK*'s client package is multi-threaded, where the number of cores is configurable by the participants. This makes the computation of the model parameters in the cohorts very fast, especially for large datasets. While we provide a readily usable web service running at *exbio server* (https://exbio.wzw.tum.de/splink) and online compensator at *compbio server* (https://compensator.compbio.sdu.dk), the server, WebApp, and compensator packages can, of course, be deployed on customized physical machines.

The *functional workflow* of *sPLINK* is comprised of the following steps:

1. **Project creation**: The coordinator creates the project (new study) through the Web interface. To this end, she/he first specifies the project name, association test name, chunk size, and the list of confounding features (only for regression tests), and then, generates a unique project token for each cohort.
2. **Cohort invitation**: The coordinator sends the project ID (automatically generated) and token to each participant (a human entity interacting with the client package in a cohort) through a secure channel such as email for inviting the cohorts to the project.
3. **Cohort joining**: The participants use their corresponding username, password, project ID, and token to join the project. After joining, they can view the general information of the project such as the coordinator, server/compensator name/URL, and etc. If they agree to proceed, they choose the dataset they want to employ in the study. To be consistent with *PLINK*, *sPLINK* supports *.bed* (value of SNPs), *.fam* (sample IDs as well as sex and phenotype values), *.bim* (chromosome

number, name, and base-pair distance of each SNP), *.cov* (value of confounding factors), and *.pheno* (phenotype values that should be used instead of those in *.fam* file) file formats as specified in the *PLINK* manual [52]. For linear regression, phenotype values must be quantitative while for logistic regression and chi-square, phenotype values have to be binary (control/case are encoded as 1/2).

4. **Federated computation**: In *sPLINK*, the association test results are computed by the client package (running on the local machines of cohorts), server package (running in the central server), and compensator (running in its own machine) in a federated manner. The computation is iterative and consists of six general steps:

   (a) **Get global parameters**: All clients obtain the required global parameters M_G from the server.
   (b) **Compute local parameters**: Each client i computes the local parameters M_{L_i} using the local data and global parameters.
   (c) **Mask local parameters**: Each client i generates random noise N_{L_i} with the same shape as M_{L_i}, and masks M_{L_i} with N_{L_i} to obtain the noisy local parameters M'_{L_i}.
   (d) **Share noisy local parameters and noise**: Each client i shares M'_{L_i} and N_{L_i} with the server and compensator, respectively.
   (e) **Aggregate noise**: The compensator computes the aggregated noise N given the noise values from the clients and sends the aggregated noise N to the server.
   (f) **Compute global parameters**: The server calculates (unmasks) the global parameters given the noisy local parameters and the negative of the aggregated noise.

5. **Result download**: The final results are automatically downloaded for the cohorts but the coordinator needs to download them manually through the web interface. Similar to *PLINK*, *sPLINK* reports minor allele name (*A1*) and p-value (P) for all three association tests, chi-square (*CHISQ*), odds ratio (*OR*), minor allele frequency in cases (*F_A*), and minor allele frequency in controls (*F_U*) for chi-square test, and the number of non-missing samples (*NMISS*), beta (*BETA*), and t-statistic (*STAT*) for linear and logistic regression tests.

sPLINK inherits its masking mechanism from *HyFed*, which masks the local parameters with non-negative integer and floating-point values in different ways. For a local parameter with a non-negative integer value, *sPLINK* considers a finite field $\mathbb{Z}_p$={0, 1, $p-1$} (p is a *prime* number) [13], where each client i generates a uniform random integer from $\mathbb{Z}_p$ as noise N_{L_i} and masks its local parameter M_{L_i} with N_{L_i} by performing the *modular addition* over $\mathbb{Z}_p$: $M'_{L_i} = (M_{L_i} + N_{L_i}) \mod p$. Notice that $M_{L_i}, N_{L_i}, M'_{L_i} \in \mathbb{Z}_p$. For M_{L_i} with a floating-point value, each client i generates noise N_{L_i} using Gaussian random generator with zero-mean and variance σ_N^2, and masks M_{L_i} with N_{L_i} using the ordinary addition: $M'_{L_i} = M_{L_i} + N_{L_i}$.

The compensator computes the aggregated noise N by taking sum over the noise values of the clients using the modular or ordinary addition depending on the data type of the noise: if N_{L_i} is non-negative integer, then $N = (\sum_{i=1}^{i=K} N_{L_i}) \mod p$; if N_{L_i} is floating-point type, then $N = \sum_{i=1}^{i=K} N_{L_i}$. To calculate the global parameters with non-negative integer values, the server first computes the aggregated noisy parameter by taking sum over the noisy local parameters using the modular addition, and then subtracts the aggregated noise from the aggregated noisy parameter using the modular subtraction: $M_G = (((\sum_{i=1}^{i=K} M'_{L_i}) \mod p) - N) \mod p$. For model parameters with floating-point values, the server adds up the noisy local parameters and the negative of the aggregated noise using the ordinary addition: $M_G = \sum_{i=1}^{i=K} M'_{L_i} - N$.

The *computational workflow* of *sPLINK* involves seven steps common among all association tests as well as a couple of steps specific to each association test (Fig. 8). In the first three steps (i.e. *Init*, *SNP name*, and *Allele name*) as well as the sixth step (*Minor allele*), the clients only communicate with the server, where the name of the SNPs and alleles (which are not considered private) are directly shared with the server. In the remaining steps, the compensator is involved and clients mask the local parameters with noise to hide their original values from the server. The formulas associated with the steps indicate how the clients compute local parameters and how the server calculates the global parameters using the noisy local parameters of the clients and the aggregated noise from the compensator. In the following, we provide an overview of each step:

1. **Init**: Each client i opens the files of the dataset selected by the participant to be employed in the study and creates its phenotype vector (Y_i) and feature matrix (X_i), which includes the value of SNPs and confounding factors. It is worth noting that there is a separate feature matrix for each SNP but the phenotype vector is the same for all SNPs. Assume a dataset containing three SNPs named *SNP1*, *SNP2*, and *SNP3* and *age* and *sex* as confounding features. There will be three different feature matrices, one feature matrix per SNP. For instance, the feature matrix of *SNP1* has three columns including *SNP1*, *age*, and *sex* values. Phenotype vector and feature matrix are the private data of the cohorts. They cannot be shared with the server, compensator, or the other cohorts. The aggregation process in the server just makes sure that all clients successfully initialized their data.
2. **SNP name**: Each client shares the SNP names with the server. In the aggregation process, the server computes the intersection of all SNP names. Only common SNPs are considered in the computation of the association test results.
3. **Allele name**: Each client sends the allele names (e.g. G,A) of each SNP to the server. In the aggregation process, the server ensures that all cohorts employ the same allele names for the SNPs. Notice that the clients sort the allele names to avoid revealing which one is minor or major allele.
4. **Sample count**: Each client i calculates its local sample count T_i (number of samples in its dataset including missing samples, which is the size of vector Y_i). The server computes the corresponding global sample count: $T = (((\sum_{i=1}^{i=K} T'_i) \mod p) - N_T) \mod p$, where T'_i is the noisy local sample count of client i: $T'_i = (T_i + N_i) \mod p$ and N_T is the aggregated noise from the compensator: $N_T = (\sum_{i=1}^{i=K} N_i) \mod p$.
5. **Non-missing count**: In this step, SNPs are split into chunks which can be processed in parallel. The chunking capability is provided to handle very large datasets containing millions of SNPs. The clients compute the non-missing sample count by filtering out the missing samples (value of -9 is considered as missing). Likewise, they calculate the local allele count by counting the number of alleles in

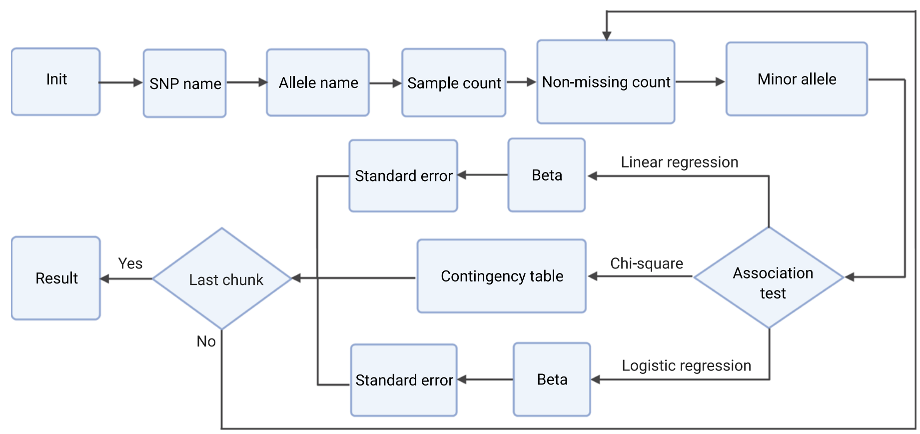


Fig. 8 Computational workflow of *sPLINK*: The first six steps and the last step are common among all association tests. Contingency table is specific to the chi-square test while Beta and Standard error are regression test related steps

each SNP. In the aggregation process, the server computes the global non-missing sample count (n) and allele count using the corresponding noisy parameters and the aggregated noise similar to the sample count step. Finally, the server determines the global minor allele based on the values of the global allele counts.

6. **Minor allele**: The clients compare their local minor allele with the global minor allele. If they are the same, they do nothing. Otherwise, they update the mapping of SNP values read from .bed file. Each SNP value can be 0, 1, 2, or 3 (missing value). These values are encoded based on the minor allele name. If the minor allele is changed, the value of the SNP needs to be swapped if it is 0 or 2. Thus, if a client's minor allele is different from global minor allele, it inverses the mapping of SNP values ($0 \rightarrow 2$ and $2 \rightarrow 0$). The aggregation in the server makes sure that all clients successfully completed this step.
7. **Association test specific steps**: In the following, we elaborate on the steps specific to each association test. Regarding regression tests, *sPLINK* implements the federated versions of ordinary least squares linear regression and Newton-Raphson method based logistic regression.
   Chi-square: The only test-specific step for the chi-square test is *Contingency table*, where each client i computes its local contingency table containing minor allele frequency for cases (t_i), minor allele frequency for controls (r_i), major allele frequency for cases (q_i), and major allele frequency for controls (s_i). The server aggregates the noisy contingency tables from the clients (t'_i, r'_i, q'_i, and s'_i are the elements of the table) and the corresponding aggregated noise from the compensator (N_t, N_r, N_q, and N_s) to compute the global (observed) contingency table (Table 3). It also calculates the expected contingency table based on the observed contingency table (Table 4).
   Given the observed contingency table (O) and the expected contingency table (E), the server computes odds ratio (OR), χ^2, and p-value (P) as follows:

$$\mathrm{OR} = \frac{t \times s}{q \times r} \tag{1}$$

Table 3 Global (observed) contingency table

	Minor allele	Major allele	Total
Case	$t = (((\sum_{i=1}^{i=K} t'_i) \mod p) - N_t) \mod p$	$q = (((\sum_{i=1}^{i=K} q'_i) \mod p) - N_q) \mod p$	$t+q$
Control	$r = (((\sum_{i=1}^{i=K} r'_i) \mod p) - N_r) \mod p$	$s = (((\sum_{i=1}^{i=K} s'_i) \mod p) - N_s) \mod p$	$r+s$
Total	$t+r$	$q+s$	2n

$$\chi^2 = \sum \frac{(E-O)^2}{E} \tag{2}$$

$$P = 1 - F_t(\chi^2, 1) \tag{3}$$

where F_t is the cumulative distribution function (CDF) of χ^2 distribution (degree of freedom is 1).

Linear regression: *Beta* and *Standard error* are two steps specific to linear regression test. In the *Beta* step, each client i computes $X_i^T X_i$ and $X_i^T Y_i$, where X_i^T is the transpose of X_i. In the aggregation process, the server performs the following calculations (K is the number of clients):

$$X^T X = \sum_{i=1}^{i=K} (X_i^T X_i)' - N_{X^T X} \tag{4}$$

$$X^T Y = \sum_{i=1}^{i=K} (X_i^T Y_i)' - N_{X^T Y} \tag{5}$$

$$\beta = (X^T X)^{-1} (X^T Y) \tag{6}$$

where $(X_i^T X_i)'$ and $(X_i^T Y_i)'$ are the noisy local parameters from the clients, $N_{X^T X}$ and $N_{X^T Y}$ are the corresponding aggregated noise from the compensator, and $()^{-1}$ indicates the inverse matrix.

In the *Standard error* step, each client i calculates the local sum square error (SSE) E_i by having the global β vector.

$$\hat{Y}_i = X_i \beta \tag{7}$$

$$E_i = \sum (Y_i - \hat{Y}_i)^2 \tag{8}$$

and then the server calculates the global standard error vector (SE) as follows:

$$E = \sum_{i=1}^{i=K} E'_i - N_E \tag{9}$$

$$\text{VAR} = \left(\frac{E}{n-m-1}\right)(X^T X)^{-1} \tag{10}$$

$$\text{SE} = \sqrt{\text{diag}(\text{VAR})} \tag{11}$$

Table 4 Expected contingency table

	Minor allele	Major allele
Case	$\frac{(t+q)\times(t+r)}{2n}$	$\frac{(t+q)\times(q+s)}{2n}$
Control	$\frac{(r+s)\times(t+r)}{2n}$	$\frac{(r+s)\times(q+s)}{2n}$

where E_i' and N_E are the noisy SSE values and the corresponding aggregated noise, respectively; n is the global non-missing sample count, m is the number of features (1 + number of confounding factors), and *diag* is the main diagonal of the matrix. Given the standard error vector, the server computes the *T statistic* (T) and *p-value* (P) as follows:

$$T = \frac{\beta}{\mathrm{SE}} \tag{12}$$

$$\mathrm{DF} = n - m - 1 \tag{13}$$

$$P = 2 \times (1 - F_t(|T|, \mathrm{DF})) \tag{14}$$

in which *DF* is degree of freedom and F_t is the CDF of T distribution.

Logistic regression: Similar to linear regression, logistic regression has two specific steps: *Beta* and *Standard error*. However, the *Beta* step is iterative in logistic regression (maximum number of iterations is specified by the coordinator and its default value is 20). In each iteration, each client i computes local gradient (∇_i), Hessian matrix (H_i) and log-likelihood (L_i) as follows:

$$\hat{Y}_i = \frac{1}{1 + e^{-X_i\beta}} \tag{15}$$

$$\nabla_i = X_i^T(Y_i - \hat{Y}_i) \tag{16}$$

$$H_i = (X_i^T \circ (\hat{Y}_i \circ (1 - \hat{Y}_i))^T)X_i \tag{17}$$

$$L_i = \sum (Y_i \circ \log \hat{Y}_i + (1 - Y_i) \circ \log(1 - \hat{Y}_i)) \tag{18}$$

where β is the global beta vector from the previous iteration and $\circ$ indicates element-wise multiplication.

The server aggregates the noisy local gradients (∇_i'), Hessian matrices (H_i') and log-likelihood values (L_i') from K clients and the associated aggregated noise values N_∇, N_H, N_L as follows:

$$\nabla = \sum_{i=1}^{i=K} \nabla_i' - N_\nabla \tag{19}$$

$$H = \sum_{i=1}^{i=K} H_i' - N_H \tag{20}$$

$$L = \sum_{i=1}^{i=K} L_i' - N_L \tag{21}$$

Then, it updates the β values accordingly:

$$\beta_{\mathrm{new}} = \beta_{\mathrm{old}} + H^{-1}\nabla \tag{22}$$

where β_{old} is the β value from the previous iteration. The server also compares the newly computed log-likelihood value (L) with the one from previous iteration (L_{old}). If their difference is less than a pre-specified threshold, β values converged, and therefore, it stops updating beta.

In the *Standard error* step, the server shares the global β values with the clients. Each client i computes its local Hessian matrix (H_i) using the global β. The server gets the noisy local Hessian matrices from K clients and the aggregated noise from the compensator and applies the following formula to obtain the global standard error vector (SE):

$$\mathrm{SE} = \sqrt{\mathrm{diag}\Big(\big(\sum_{i=1}^{i=K} H_i' - N_H\big)^{-1}\Big)} \tag{23}$$

Having standard error values, the server calculates T statistics and p-value (P) as follows:

$$T = \frac{\beta}{\mathrm{SE}} \tag{24}$$

$$P = 1 - F_t(|T|^2, 1) \tag{25}$$

where F_t is CDF of χ^2 distribution (degree of freedom is 1).

8. **Result**: The computation of association test results have been completed for all chunks and the results are shared with all cohorts.

The client and server components of *sPLINK* has been written using the Python API of the HyFed framework [53]. The WebApp component has been implemented using Angular and HTML/CSS. *sPLINK* employs the algorithm-agnostic compensator of the HyFed framework. The *pandas* package [54] is used in the client component to open the dataset files while *NumPy* [55] is leveraged to pre-process the data and to compute the local parameters. In the server component, the *NumPy* and *SciPy* [56] packages are used for aggregation and computing p-values.

Supplementary Information

The online version contains supplementary material available at https://doi.org/10.1186/s13059-021-02562-1.

Additional file 1: Experimental details. Table S1. The SHIP case study. **Table S2.**. The COPDGene case study. **Table S3.** The FinnGen case study. **Supplementary results. Figure S1.** The significant SNPs overlapped between sPLINK and PLINK for the SHIP case study considering Bonferroni significance threshold. **Figure S2.** The Spearman rank correlation coefficient between the *p*-values from each tool and the aggregated analysis for the COPDGene and FinnGen case studies. **Figure S3.** Runtime and network bandwidth usage of sPLINK with varying number of SNPs. **Figure S4.** Runtime and network bandwidth usage of sPLINK with varying number of samples. **Figure S5.** Runtime and network bandwidth usage of sPLINK with varying number of clients. **Experimental setup. Table S4.** The system specification of the physical machines and laptops used to measure the runtime and network bandwidth usage of sPLINK. **Table S5.** The experimental setup used for measuring the runtime and network bandwidth usage of sPLINK.

Additional file 2: Review history.

Acknowledgements

We would like to thank Anne Hartebrodt and Richard Röttger for providing and setting up the workstation for the compensator component at the University of Southern Denmark. SHIP is part of the Community Medicine Research net of the University of Greifswald, Germany (https://www.community-medicine.de), which is funded by the Federal Ministry of Education and Research (grants no. 01ZZ9603, 01ZZ0103, and 01ZZ0403), the Siemens AG, the Ministry of Cultural Affairs as well as the Social Ministry of the Federal State of Mecklenburg-West Pomerania, and the network 'Greifswald Approach to Individualized Medicine (GANI_MED)' funded by the Federal Ministry of Education and Research (grant 03IS2061A). ExomeChip data have been supported by the Federal Ministry of Education and Research (grant no. 03Z1CN22) and the Federal State of Mecklenburg-West Pomerania. Patients and control subjects in FinnGen provided informed consent for biobank research, based on the Finnish Biobank Act. Alternatively, older research cohorts, collected prior the start of FinnGen (in August 2017), were collected based on study-specific consents and later transferred to the Finnish biobanks after approval by Fimea, the National Supervisory Authority for Welfare and Health. Recruitment protocols followed the biobank protocols approved by Fimea. The Coordinating Ethics Committee of the Hospital District of Helsinki and Uusimaa (HUS) approved the FinnGen study protocol Nr HUS/990/2017. The FinnGen project is approved by Finnish Institute for Health and Welfare (THL), approval number THL/2031/6.02.00/2017, amendments THL/1101/5.05.00/2017, THL/341/6.02.00/2018, THL/2222/6.02.00/2018, THL/283/6.02.00/2019 Digital and population data service agency VRK43431/2017-3, VRK/6909/2018-3, the Social Insurance Institution (KELA) KELA 58/522/2017, KELA 131/522/2018 and Statistics Finland TK-53-1041-17. The Biobank Access Decisions for FinnGen samples and data utilized in FinnGen Data Freeze 3 include: THL Biobank BB2017_55, BB2017_111, BB2018_19, BB_2018_34, BB_2018_67, BB2018_71, Red Cross Blood Service Biobank 7.12.2017, Helsinki Biobank HUS/359/2017, Auria Biobank AB17-5154, Biobank Borealis of Northern Finland_2017_1013, Biobank of Eastern Finland 1186/2018, Finnish Clinical Biobank Tampere MH0004, Central Finland Biobank 1-2017. The FinnGen project is funded by two grants from Business Finland (HUS 4685/31/2016 and UH 4386/31/2016) and eleven industry partners (AbbVie Inc, AstraZeneca UK Ltd, Biogen MA Inc, Celgene Corporation, Celgene International II Sàrl, Genentech Inc, Merck Sharp & Dohme Corp, Pfizer Inc., GlaxoSmithKline, Sanofi, Maze Therapeutics Inc., Janssen Biotech Inc). Following biobanks are acknowledged for collecting the FinnGen project samples: Auria Biobank (https://www.auria.fi/biopankki), THL Biobank (https://www.thl.fi/biobank), Helsinki Biobank (https://www.helsinginbiopankki.fi), Biobank Borealis of Northern Finland (https://www.ppshp.fi/Tutkimus-ja-opetus/Biopankki/Pages/Biobank-Borealis-briefly-in-English.aspx), Finnish Clinical Biobank Tampere (https://www.tays.fi/en-US/Research_and_development/Finnish_Clinical_Biobank_Tampere), Biobank of Eastern Finland (https://www.ita-suomenbiopankki.fi/en), Central Finland Biobank (https://www.ksshp.fi/fi-FI/Potilaalle/Biopankki), Finnish Red Cross Blood Service Biobank (https://www.veripalvelu.fi/verenluovutus/biopankkitoiminta) and Terveystalo Biobank (https://www.terveystalo.com/fi/Yritystietoa/Terveystalo-Biopankki/Biopankki/). All Finnish Biobanks are members of BBMRI.fi infrastructure (https://www.bbmri.fi). Figures 2 and 8 were created with BioRender.com.

Peer review information
Barbara Cheifet was the primary editor of this article and managed its editorial process and peer review in collaboration with the rest of the editorial team.

Review history
The review history is available as Additional file 2.

Authors' contributions
R.N., R.T., T.F., T.K., and J.B. conceived and designed the study. R.N. and R.T. developed the federated algorithms. R.N., R.T., and J.M. implemented the client and server components. J.M., R.N., R.T., T.F., and J.S. implemented the WebApp component. T.K. and R.N. performed the aggregated and federated association tests on the SHIP dataset. R.N., T.F., T.K., M.L., J.B., and S.W. conducted the meta-analysis on the COPDGene case study. R.N. and E.P. performed the meta-analysis on the FinnGen dataset. R.N., J.S., J.M., and R.T. conducted the performance measurements. R.N. and R.T. prepared the original draft. G.K. and D.R. provided critical feedback on the design and implementation of the tool from the privacy perspective. M.L., T.K., N.K.W., D.H., U.V., and J.B. helped with the manuscript revising. T.K., J.B., and M.L. assisted in the improvement of the tool. The authors read and approved the final manuscript.

Funding
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826078. This reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains. This work was supported by the BMBF-funded de.NBI Cloud within the German Network for Bioinformatics Infrastructure (de.NBI) (031A537B, 031A533A, 031A538A, 031A533B, 031A535A, 031A537C, 031A534A, 031A532B). Open Access funding enabled and organized by Projekt DEAL.

Availability of data and materials
The SHIP dataset [33] is accessible to researchers after completing a web-based request form at http://ship.community-medicine.de and approval. The COPDGene dataset [35] is publicly available (dbGaP accession number phs000179.v1.p1). The FinnGen dataset [36] is available for researchers by requesting access to the FinnGen Sandbox environment, and after completing Sandbox training on how to deal with personal data, and passing an exam about data security (https://www.finngen.fi/en). The sPLINK tool is available online at https://exbio.wzw.tum.de/splink. The source code of sPLINK is publicly available at GitHub (https://github.com/tum-aimed/splink) and Zenodo (DOI: 10.5281/zenodo.5735472) [57] under the Apache License Version 2.0.

Declarations

Ethics approval and consent to participate
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Author details
[1] AI in Medicine and Healthcare, Technical University of Munich, Munich, Germany. [2] Klinikum rechts der Isar, Munich, Germany. [3] Chair of Computational Systems Biology, University of Hamburg, Hamburg, Germany. [4] Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark. [5] Chair of Experimental Bioinformatics, TUM School of Life Sciences, Technical University of Munich, Munich, Germany. [6] Department of Functional Genomics, University Medicine Greifswald, Greifswald, Germany. [7] Institute for Molecular Medicine Finland (FIMM), Helsinki Institute of Life Science (HiLIFE), University of Helsinki, Helsinki, Finland. [8] Applied Tumor Genomics Research Program, Research Programs Unit, Faculty of Medicine, University of Helsinki, Helsinki, Finland. [9] Department of Mathematics and Computer Science, University of Marburg, Marburg, Germany. [10] Division Data Science in Biomedicine, Peter L. Reichertz Institute for Medical Informatics of TU Braunschweig and Hannover Medical School, Brunswick, Germany. [11] Braunschweig Integrated Centre of Systems Biology (BRICS), Brunswick, Germany. [12] Biomedical Image Analysis Group, Imperial College London, London, UK. [13] OpenMined, Oxford, UK.

Received: 25 November 2020 Accepted: 2 December 2021
Published online: 24 January 2022

References

1. Fareed M, Afzal M. Single nucleotide polymorphism in genome-wide association of human population: A tool for broad spectrum service. Egypt J Med Human Genet. 2013;14(2):123–34.
2. Visscher PM, Wray NR, Zhang Q, Sklar P, McCarthy MI, Brown MA, Yang J. 10 years of gwas discovery: biology, function, and translation. Am J Hum Genet. 2017;101(1):5–22.
3. Visscher PM, Brown MA, McCarthy MI, Yang J. Five years of gwas discovery. Am J Hum Genet. 2012;90(1):7–24.
4. De R, Bush W, Moore J. Bioinformatics challenges in genome-wide association studies (gwas). Methods Mol Biol (Clifton, NJ). 2014;1168:63–81.
5. Purcell S, Neale B, Todd-Brown K, Thomas L, Ferreira MA, Bender D, Maller J, Sklar P, De Bakker PI, Daly MJ, et al. Plink: a tool set for whole-genome association and population-based linkage analyses. Am J Hum Genet. 2007;81(3): 559–75.
6. Evangelou E, Ioannidis JP. Meta-analysis methods for genome-wide association studies and beyond. Nat Rev Genet. 2013;14(6):379–89.
7. Willer CJ, Li Y, Abecasis GR. Metal: fast and efficient meta-analysis of genomewide association scans. Bioinformatics. 2010;26(17):2190–1.
8. Mägi R, Morris AP. Gwama: software for genome-wide association meta-analysis. BMC Bioinformatics. 2010;11(1):288.
9. Lunetta KL. Methods for meta-analysis of genetic data. Curr Protoc Human Genet. 2013;77(1):1–24.
10. Cantor RM, Lange K, Sinsheimer JS. Prioritizing gwas results: a review of statistical methods and recommendations for their application. Am J Hum Genet. 2010;86(1):6–22.
11. de Vlaming R, Okbay A, Rietveld CA, Johannesson M, Magnusson PK, Uitterlinden AG, van Rooij FJ, Hofman A, Groenen PJ, Thurik AR, et al. Meta-gwas accuracy and power (metagap) calculator shows that hiding heritability is partially due to imperfect genetic correlations across studies. PLoS Genet. 2017;13(1):e1006495.
12. Gentry C. Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing; 2009. p. 169–78.
13. Cramer R, Damgård IB, Nielsen JB. Secure Multiparty Computation and Secret Sharing. Cambridge: Cambridge University Press; 2015.
14. McMahan B, Moore E, Ramage D, Hampson S, y Arcas BA. Communication-efficient learning of deep networks from decentralized data. In: Artificial Intelligence and Statistics. Fort Lauderdale: PMLR; 2017. p. 1273–82.
15. Konečnỳ J, McMahan HB, Yu FX, Richtárik P, Suresh AT, Bacon D. Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492. 2016. https://arxiv.org/abs/1610.05492.
16. Shamir A. How to share a secret. Commun ACM. 1979;22(11):612–3.
17. Kamm L, Bogdanov D, Laur S, Vilo J. A new way to protect privacy in large-scale genome-wide association studies. Bioinformatics. 2013;29(7):886–93.
18. Cho H, Wu DJ, Berger B. Secure genome-wide association analysis using multiparty computation. Nat Biotechnol. 2018;36(6):547–51.
19. Shi H, Jiang C, Dai W, Jiang X, Tang Y, Ohno-Machado L, Wang S. Secure multi-party computation grid logistic regression (smac-glore). BMC Med Inf Dec Making. 2016;16(3):89.
20. Constable SD, Tang Y, Wang S, Jiang X, Chapin S. Privacy-preserving gwas analysis on federated genomic datasets. BMC Med Inf Dec Making. 2015;15:1–9.
21. Alexandru AB, Pappas GJ. Secure multi-party computation for cloud-based control. In: Privacy in Dynamical Systems. Singapore: Springer; 2020. p. 179–207.
22. Lu W-J, Yamada Y, Sakuma J. Privacy-preserving genome-wide association studies on cloud environment using fully homomorphic encryption. BMC Med Inf Dec Making. 2015;15:1–8.
23. Morshed T, Alhadidi D, Mohammed N. Parallel linear regression on encrypted data. In: 2018 16th Annual Conference on Privacy, Security and Trust (PST). Los Alamitos: IEEE Computer Society; 2018. p. 1–5.
24. Kim M, Song Y, Wang S, Xia Y, Jiang X, et al. Secure logistic regression based on homomorphic encryption: Design and evaluation. JMIR Med Inf. 2018;6(2):8805.
25. Sadat MN, Al Aziz MM, Mohammed N, Chen F, Jiang X, Wang S. Safety: secure gwas in federated environment through a hybrid solution. IEEE/ACM Trans Comput Biol Bioinforma. 2018;16(1):93–102.
26. Chialva D, Dooms A. Conditionals in homomorphic encryption and machine learning applications. arXiv preprint arXiv:1810.12380. 2018. https://arxiv.org/abs/1810.12380.
27. Wang S, Jiang X, Wu Y, Cui L, Cheng S, Ohno-Machado L. Expectation propagation logistic regression (explorer): distributed privacy-preserving online model learning. J Biomed Inf. 2013;46(3):480–96.
28. Wu Y, Jiang X, Kim J, Ohno-Machado L. Grid binary logistic regression (glore): building shared models without sharing data. J Am Med Inf Assoc. 2012;19(5):758–64.
29. Jiang W, Li P, Wang S, Wu Y, Xue M, Ohno-Machado L, Jiang X. Webglore: a web service for grid logistic regression. Bioinformatics. 2013;29(24):3238–40.
30. Nasirigerdeh R, Torkzadehmahani R, Matschinske J, Baumbach J, Rueckert D, Kaissis G. HyFed: A Hybrid Federated Framework for Privacy-preserving Machine Learning. arXiv preprint arXiv:2105.10545. 2021. https://arxiv.org/abs/2105.10545.
31. Hastie T, Tibshirani R, Friedman J. The Elements of Statistical Learning. Cambridge: Springer; 2009.
32. McHugh ML. The chi-square test of independence. Biochemia Med Biochemia Med. 2013;23(2):143–9.
33. Völzke H, Alte D, Schmidt CO, Radke D, Lorbeer R, Friedrich N, Aumann N, Lau K, Piontek M, Born G, et al. Cohort profile: the study of health in pomerania. Int J Epidemiol. 2011;40(2):294–307.
34. Weiss FU, Schurmann C, Guenther A, Ernst F, Teumer A, Mayerle J, Simon P, Völzke H, Radke D, Greinacher A, et al. Fucosyltransferase 2 (fut2) non-secretor status and blood group b are associated with elevated serum lipase activity in asymptomatic subjects, and an increased risk for chronic pancreatitis: a genetic association study. Gut. 2015;64(4):646–56.

35. COPDGene. http://www.copdgene.org/. Accessed 30 Nov 2021.
36. FinnGen Documentation of R3 release. https://r3.finngen.fi/about. Accessed 30 Nov 2021.
37. Pillai SG, Ge D, Zhu G, Kong X, Shianna KV, Need AC, Feng S, Hersh CP, Bakke P, Gulsvik A, et al. A genome-wide association study in chronic obstructive pulmonary disease (copd): identification of two major susceptibility loci. PLoS Genet. 2009;5(3):e1000421.
38. Pei Y-F, Tian Q, Zhang L, Deng H-W. Exploring the major sources and extent of heterogeneity in a genome-wide association meta-analysis. Ann Hum Biol. 2016;80(2):113–22.
39. Lyu L, Yu H, Yang Q. Threats to federated learning: A survey. arXiv preprint arXiv:2003.02133. 2020. https://arxiv.org/abs/2003.02133.
40. Dwork C. Differential privacy. In: International Colloquium on Automata, Languages, and Programming. Berlin: Springer; 2006. p. 1–12.
41. Cover TM. Elements of Information Theory. New York: John Wiley & Sons; 1999.
42. Dibert A, Csirmaz L. Infinite secret sharing–examples. J Math Cryptol. 2014;8(2):141–68.
43. Tjell K, Wisniewski R. Privacy in Distributed Computations based on Real Number Secret Sharing. arXiv preprint arXiv:2107.00911. 2021. https://arxiv.org/abs/2107.00911.
44. Yang Q, Liu Y, Chen T, Tong Y. Federated machine learning: Concept and applications. ACM Trans Intell Syst Technol (TIST). 2019;10(2):1–19.
45. Nasirigerdeh R, Torkzadehmahani R, Baumbach J, Blumenthal DB. On the privacy of federated pipelines. In: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '21). New York: Association for Computing Machinery; 2021.
46. Galinsky KJ, Bhatia G, Loh P-R, Georgiev S, Mukherjee S, Patterson NJ, Price AL. Fast principal-component analysis reveals convergent evolution of adh1b in europe and east asia. Am J Hum Genet. 2016;98(3):456–472.
47. Kairouz P, McMahan HB, Avent B, Bellet A, Bennis M, Bhagoji AN, Bonawitz K, Charles Z, Cormode G, Cummings R, et al. Advances and open problems in federated learning. arXiv preprint arXiv:1912.04977. 2019. https://arxiv.org/abs/1912.04977.
48. Zhu L, Han S. Deep leakage from gradients. In: Federated Learning. Cham: Springer; 2020. p. 17–31.
49. Melis L, Song C, De Cristofaro E, Shmatikov V. Exploiting unintended feature leakage in collaborative learning. In: 2019 IEEE Symposium on Security and Privacy (SP). Manhattan: IEEE; 2019. p. 691–706.
50. Torkzadehmahani R, Nasirigerdeh R, Blumenthal DB, Kacprowski T, List M, Matschinske J, Späth J, Wenke NK, Bihari B, Frisch T, et al. Privacy-preserving Artificial Intelligence Techniques in Biomedicine. arXiv preprint arXiv:2007.11621. 2020. https://arxiv.org/abs/2007.11621.
51. Bonawitz K, Ivanov V, Kreuter B, Marcedone A, McMahan HB, Patel S, Ramage D, Segal A, Seth K. Practical secure aggregation for privacy-preserving machine learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. New York: Association for Computing Machinery; 2017. p. 1175–91.
52. PLINK data formats. http://zzz.bwh.harvard.edu/plink/data.shtml. Accessed 30 Nov 2021.
53. HyFed API. https://github.com/tum-aimed/hyfed. Accessed 30 Nov 2021.
54. pandas: Python Data Analysis Library. https://pandas.pydata.org/. Accessed 30 Nov 2021.
55. Harris CR, Millman KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, Wieser E, Taylor J, Berg S, Smith NJ, Kern R, Picus M, Hoyer S, van Kerkwijk MH, Brett M, Haldane A, del Río JF, Wiebe M, Peterson P, Gérard-Marchant P, Sheppard K, Reddy T, Weckesser W, Abbasi H, Gohlke C, Oliphant TE. Array programming with NumPy. Nature. 2020;585(7825):357–62.
56. Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, van der Walt SJ, Brett M, Wilson J, Millman KJ, Mayorov N, Nelson ARJ, Jones E, Kern R, Larson E, Carey CJ, Polat İ, Feng Y, Moore EW, VanderPlas J, Laxalde D, Perktold J, Cimrman R, Henriksen I, Quintero EA, Harris CR, Archibald AM, Ribeiro AH, Pedregosa F, van Mulbregt P, SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nat Methods. 2020;17:261–272.
57. Nasirigerdeh R, Torkzadehmahani R, Matschinske J, Frisch T, List M, Späth J, Weiss S, Völker U, Pitkänen E, Heider D, Wenke NK, Kaissis G, Rueckert D, Kacprowski T, Baumbach J. splink: a hybrid federated tool as a robust alternative to meta-analysis in genome-wide association studies. Zenodo. 2021. https://doi.org/10.5281/zenodo.5735472.

Publisher's Note

Utility-preserving Federated Learning

Reza Nasirigerdeh, Daniel Rueckert & Georgios Kaissis

Workshop: In Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec'23)

Synopsis: We theoretically prove and experimentally show that deep neural network (DNN) models trained on distributed data in a federated fashion can achieve the same utility as those trained on the corresponding centralized data provided that particular conditions are satisfied for the training algorithm, model, loss function, and optimizer as the main components of DNN training. More precisely, if the (1) DNN model and loss function are batch-independent and deterministic, (2) training algorithm selects all clients, instruments them to perform a single local update per communication round, and enforces the server to aggregate the local parameters from the clients using sample size based weighted averaging, and (3) optimizer employs a linear momentum function, then the models from the federated and centralized training are equivalent, and thus, they provide identical utility. We refer to a training environment satisfying the aforementioned conditions as ***utility-preserving federated learning*** (***UPFL***). Next, we evaluate the properties of the existing DNN training components to determine which one(s) can be incorporated in UPFL. Our evaluations indicate that, for instance, the *federated averaging* algorithm, which performs multiple local updates per round, does not hold the necessary conditions for UPFL. This is also the case for the *Adam* optimizer and its variants that use non-linear momentum functions as well as *batch normalization*, which is not a batch-independent layer. The *federated full gradient descent* algorithm, on the other hand, can be incorporated in UPFL. Moreover, the popular loss functions such as *cross-entropy*, the SGD optimizer, and widely used layers including *convolutional* and *linear* layers also meet the necessary conditions for UPFL. The main limitation of UPFL is remarkable communication overhead. In other words, UPFL delivers ideal utility at the expense of network communication efficiency.

Contributions of thesis author: Leading role in conceiving and designing the study, gathering software resources, developing algorithms, implementing the software components, conducting the experiments, writing the manuscript, and significant contribution in scientific findings.

Publisher: Association for Computing Machinery (ACM)

Date: 26.11.2023

Reprint permission: This paper is licences to Association for Computing Machinery (ACM). According to this license, "All ACM published authors of magazine articles, journal articles, and conference papers retain the right to post the pre-submitted (also known as pre-prints), submitted, accepted, and peer-reviewed versions of their work in any and all of the following sites: Author's Homepage and Author's Institutional Repository." Note that the accepted version of the paper is included in the dissertation. Please see Appendix C for the complete description of "Publication Rights and Licensing Policy" of ACM.

Utility-preserving Federated Learning

Reza Nasirigerdeh
Technical University of Munich
Helmholtz Munich
Munich, Germany
reza.nasirigerdeh@tum.de

Daniel Rueckert
Technical University of Munich
Munich, Germany
Imperial College London
London, United Kingdom
daniel.rueckert@tum.de

Georgios Kaissis
Technical University of Munich
Helmholtz Munich
Munich, Germany
g.kaissis@tum.de

ABSTRACT

We investigate the concept of *utility-preserving federated learning* (UPFL) in the context of deep neural networks. We theoretically prove and experimentally validate that UPFL achieves the same accuracy as centralized training independent of the data distribution across the clients. We demonstrate that UPFL can fully take advantage of the momentum and weight decay techniques compared to centralized training, but it incurs substantial communication overhead. Ordinary federated learning, on the other hand, provides much higher communication efficiency, but it can partially benefit from the aforementioned techniques to improve utility. Given that, we propose a method called *weighted gradient accumulation* to gain more benefit from the momentum and weight decay akin to UPFL, while providing practical communication efficiency similar to ordinary federated learning.

CCS CONCEPTS

• **Distributed machine learning** → **Federated learning**; • **Machine learning** → *Deep neural networks*.

KEYWORDS

Federated learning, Utility-preserving federated learning, Weighted gradient accumulation

ACM Reference Format:
Reza Nasirigerdeh, Daniel Rueckert, and Georgios Kaissis. 2023. Utility-preserving Federated Learning. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec '23), November 30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3605764.3623908

1 INTRODUCTION

Deep neural networks (DNNs) have successfully been applied to a diverse range of applications including computer vision [4], natural language processing [20], and biomedicine [26]. DNNs, however, depend on large-scale datasets to effectively train the model, which is challenging to procure in a centralized fashion because of the privacy concerns and regulations [6]. *Federated learning* (FL) [17] addresses this issue by enabling clients as data holders to collaboratively train a global model under the orchestration of a central server without sharing their private data with a third party [10]. FL, on the other hand, has faced several challenges including utility (accuracy) and network communication. FL might deliver lower accuracy compared to centralized training, especially if the data is not independent and identically distributed (NonIID) across the clients [8, 17]. FL can also incur high communication overhead, exchanging considerable amount of traffic over the network [13].

Prior studies on the utility challenge mainly focus on narrowing the accuracy gap between federated and centralized training. `FedProx` [14] is a slightly modified version of `FederatedAveraging` (`FedAvg`), the de facto standard training algorithm in FL, which adds a proximal term to the local loss functions of the clients to act as a regularizer and enforce the local models not to be far from the global one. `FedNova` [27] is another variant of `FedAvg`, which aggregates the local updates by a normalized averaging method to eliminate the inconsistencies between local updates. `FedOpt` [24] introduces variants of the adaptive optimizers including Adam [11], which are more efficient than the original counterparts in FL. Although these methods further enhance performance in federated environments, *they do not deliver the same utility as centralized training*.

In this study, we *theoretically* show FL can achieve utility *identical to* that from the centralized training provided that particular conditions are satisfied for the model, training algorithm, optimizer, and loss function as the major components in DNN training. In more detail, if the (1) model and loss function are *batch-independent* and *deterministic*, (2) training algorithm selects all clients in each communication round, enforces the clients to carry out a *single local update* per round, and employs sample-size based *weighted averaging* at the server, and (3) optimizer computes the final gradient values using a *linear combination* of the gradient values in the current and previous iterations (based on momentum), then the federated and centralized models are *equivalent*, and as a result, they achieve the same utility *regardless of data distribution across the clients*. We refer to a federated environment consisting of components satisfying the aforementioned properties as **utility-preserving federated learning (UPFL)**.

Next, we investigate the aforementioned properties for well-known (1) training algorithms such as `FedAvg` and `FedProx`, *federated full gradient descent* (`FedFGD`), and *federated single mini-batch* (`FedSMB`) [18] as its variant, (2) optimizers including SGD, Adam, and its variants, (3) loss functions such as cross-entropy and focal loss [16], and (4) model layers including the convolutional layer, batch normalization (BatchNorm) [9], and group normalization

AISec '23, November 30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0260-0/23/11…$15.00
https://doi.org/10.1145/3605764.3623908

(GroupNorm) [28]. Our analysis indicates that a federated environment consisting of the FedFGD or FedSMB algorithm, SGD optimizer, cross-entropy or any other batch-independent loss function, and a model with widely-used layers except BatchNorm is utility-preserving. We also experimentally validate the theoretical results on CIFAR-10 [12] and Imagenette [7].

Our theoretical analysis and experimental evaluation provides new insights into the utility challenge in federated environments: (I) UPFL incorporates many DNN training components initially designed for centralized training such as the SGD optimizer, cross-entropy loss function, and models consisting of convolutional, GroupNorm, and linear layers; (II) the main difference between UPFL and ordinary FL is the training algorithm. While the former is based on FedFGD or FedSMB, the latter leverages FedAvg or its variants. Interestingly, the factor that distinguishes FedFGD/FedSMB from FedAvg is the number of local updates per communication round. The former algorithms perform a *single* local update, whereas the latter one carries out *multiple* local updates per round.

Given that, we thoroughly investigate the impact of the number of local updates per round on communication efficiency and utility. Our results indicate that (I) UPFL (single local update per round) requires a huge number of communication rounds for model convergence; ordinary FL (multiple local updates per round), on the other hand, dramatically enhances the communication efficiency compared to UPFL; (II) UPFL can *fully* take advantage of the momentum and weight decay techniques to enhance model accuracy, whereas ordinary FL can *partially* benefit from the aforementioned techniques. Considering this observation, an interesting question arises: *How can a federated training algorithm benefit from momentum and weight decay considerably (ideally fully) with practical communication efficiency (multiple local updates per round)?*

As a first step towards addressing this question, we present a method called **weighted gradient accumulation** (**WGA**), where the local gradients in the initial updates have more weights than those in the final updates during gradient accumulation at the clients. The logic behind this idea is that the local models are closer to the global model during initial local updates than the final ones. We show WGA achieves higher accuracy gain using momentum and weight decay compared to ordinary FL with a comparable number of communication rounds.

In summary, we make the following contributions in this paper:

- We investigate the concept of UPFL in the context of deep learning, and theoretically prove that it achieves the same utility as centralized training regardless of the data distribution across the clients.
- We experimentally validate the theoretical results on two different datasets.
- We illustrate UPFL can fully benefit from momentum and weight decay, but incurs considerable communication overhead. Ordinary FL, on the other hand, significantly improves communication efficiency, but can partially take advantage of the before-mentioned techniques.
- We introduce the WGA method to provide more accuracy gain from momentum and weight decay compared to ordinary FL with competitive communication efficiency.

2 METHOD

We first provide preliminary material on *gradient descent*, and centralized and federated training. Next, we present the properties that DNN training components should hold to be incorporated in UPFL, and formally prove UPFL achieves the same utility as centralized training. Finally, we analyze the characteristics of different training algorithms, optimizers, loss functions, and model layers to determine whether they satisfy the necessary conditions for UPFL.

2.1 Preliminaries

Gradient descent is the most commonly used optimization algorithm for training DNNs. Assume that w is a model parameter, g is the corresponding gradient, w_i and g_i are the values of w and g in iteration i, respectively, η is learning rate, and $\mathcal{V}(g_p, \ldots, g_i, \star)$ is the *momentum function* that computes the final gradient value (p is also an iteration number and $p \leq i$). The gradient descent-based optimizers update w as follows:

$$w_i = w_{i-1} - \eta \cdot \mathcal{V}(g_p, \ldots, g_i, \star) \tag{1}$$

where $\star$ means the function can take additional arguments.

Gradient descent comes with different versions including `full gradient descent` (FGD) and `mini-batch gradient descent` (MBGD). In FGD, the gradients are calculated using *all samples* of the dataset, whereas MBGD computes the gradients using a *mini-batch of samples* from the dataset. Focusing on *supervised learning* tasks, let $\mathcal{D} = [S_1, \ldots, S_n]$ be a dataset of n samples, where S_j indicates the j-th sample of the dataset, $Y = [y_1, \ldots, y_n]$ be a vector of target values associated with the samples of $\mathcal{D}$, $\mathcal{M}_W$ be a model characterized by a vector of parameters W, and $\mathcal{L}$ be a loss function. The FGD *gradient function* corresponding to parameter $w \in W$ is:

$$\begin{gathered}\mathcal{G}_w(W, \mathcal{D}, Y) = \frac{\partial \mathcal{L}(Y, \mathcal{M}_W(\mathcal{D}))}{\partial w}, \\ G_i = [g_{i,1}, \ldots, g_{i,n}], \quad g_i = \frac{1}{n} \sum_{j=1}^{n} g_{i,j},\end{gathered} \tag{2}$$

where $g_{i,j}$ is the gradient value associated with j-th sample of $\mathcal{D}$ in iteration i, and G_i is a vector of gradient values corresponding to all samples of $\mathcal{D}$.

The MBGD algorithm first shuffles the dataset, and then divides it into mini-batches of m samples. Assuming $\mathcal{B}$ is a mini-batch from $\mathcal{D}$, and $Y_{\mathcal{B}}$ is the corresponding vector of target values, we have:

$$\begin{gathered}\mathcal{G}_w(W, \mathcal{B}, Y_{\mathcal{B}}) = \frac{\partial \mathcal{L}(Y_{\mathcal{B}}, \mathcal{M}_W(\mathcal{B}))}{\partial w}, \\ G_i = [g_{i,1}, \ldots, g_{i,m}], \quad g_i = \frac{1}{m} \sum_{j=1}^{m} g_{i,j}\end{gathered} \tag{3}$$

Notice that FGD performs a single iteration (i.e. parameter update) per *epoch*, while MBGD carries out $\lceil \frac{n}{m} \rceil$ iterations per epoch, where epoch is the number of iterations required to employ all samples of the dataset during training.

A *federated training algorithm* consists of a *client selection* procedure, *local optimization* method on the client side and an *aggregation function* on the server side. Each selected client j can apply FGD, MBGD, or their variants as the local optimization method to its dataset for computing $w^l_{i,j}$ (the *local value* of parameter w) or $g^l_{i,j}$ (the local value of gradient g) in iteration i. The aggregation function $\mathcal{A}$ instruments the server to aggregate the local values of the parameter/gradient from k clients in order to compute the *global value* of the parameter:

$$w_i = \mathcal{A}(w_{i,1}^l, \ldots, w_{i,k}^l, \star) = w_{i-1} - \eta \cdot \mathcal{A}(g_{i,1}^l, \ldots, g_{i,k}^l, \star) \quad (4)$$

Weighted averaging based on the train sample size is the widely used aggregation function in FL. This function takes the weighted average over the local values of the parameter/gradient from the clients, in which the train sample size of a client determines its relative weight during averaging:

$$\mathcal{A}(w_{i,1}^l, \ldots, w_{i,k}^l, n_1, \ldots, n_k) = \frac{\sum_{j=1}^k n_j \cdot w_{i,j}^l}{\sum_{j=1}^k n_j} \quad (5)$$

$$\mathcal{A}(g_{i,1}^l, \ldots, g_{i,k}^l, n_1, \ldots, n_k) = \frac{\sum_{j=1}^k n_j \cdot g_{i,j}^l}{\sum_{j=1}^k n_j}, \quad (6)$$

where n_j is the train sample size of client j. In the remainder of the paper, we refer to the aggregation functions in equations 5 and 6 simply as *weighted averaging*.

2.2 Utility-Preserving Federated Learning

We define a set of *properties* that training components should satisfy for UPFL, and prove models trained by UPFL and centralized training are equivalent.

Batch-independence: Let $\mathcal{X} = [X_1, \ldots, X_m]$ be a batch of input values, $\mathcal{F}$ be the mapping function of a DNN layer with parameters W, and $\mathcal{Y} = [Y_1, \ldots, Y_m]$ be the output of the layer, where $\mathcal{Y} = \mathcal{F}_W(\mathcal{X})$. The *layer is batch-independent* if the output of the layer for a particular input is *independent* of the other input values in the batch, i.e. $Y_i = \mathcal{F}_W(X_i)$ for $i = \{1, \ldots, m\}$ or in other words, $\mathcal{Y} = [\mathcal{F}_W(X_1), \ldots, \mathcal{F}_W(X_m)]$. A *model* is batch-independent if all the constituent layers of the model are batch-independent.

Similarly, let $\mathcal{L}(Y, \hat{Y})$ be a loss function, and $Y = [y_1, \ldots, y_m]$ and $\hat{Y} = [\hat{y}_1, \ldots, \hat{y}_m]$ be batches of the target and predicted output values, respectively. The *loss function is batch-independent* if it computes the distance between a particular target and predicted value *independently* of the other values in the batches; that is, $\mathcal{L}(Y, \hat{Y}) = [\mathcal{L}(y_1, \hat{y}_1), \ldots, \mathcal{L}(y_m, \hat{y}_m)]$.

Determinism: A *layer* is *deterministic* if applying the layer to the same input always produces the same output. In other words, the mapping function of the layer is not a randomized function. A *model* is deterministic if all layers of the model are deterministic.

Momentum function linearity: The *momentum function* of an *optimizer* is *linear* if the function linearly combines the gradient values from iteration p to current iteration i for obtaining the final gradient value to update the parameter:

$$\mathcal{V}(g_p, \ldots, g_i, \alpha_p, \ldots, \alpha_i) = \alpha_p \cdot g_p + \ldots + \alpha_i \cdot g_i, \quad (7)$$

where $\alpha_p, \ldots, \alpha_i$ are constant values.

Proposition 2.1. *Federated learning and centralized training using full gradient descent are equivalent, that is the parameters from the federated and centralized models are identical in each iteration, if the (1) model is batch-independent and deterministic, (2) loss function is batch-independent, (3) optimizer uses a linear momentum function, and (4) training algorithm selects all clients, the clients perform a single local update per communication round using all samples of their datasets, and the server employs weighted averaging as the aggregation function.*

Proof. The proof can be found in Appendix A. □

The *equivalence* between the federated and centralized training implies the corresponding models achieve the same utility. In other words, federated learning *fully preserves the utility* compared to centralized training.

2.3 Suitability Analysis

We explore the properties of well-known training algorithms, optimizers, model layers, and loss functions to determine which one can be incorporated in UPFL.

2.3.1 Federated Training Algorithms. The algorithms differ from each other in the number of local updates per round and the aggregation method assuming that all clients are selected for training in each communication round. `FedAvg` is the de facto standard algorithm for FL, which instruments the clients to use `MBGD` for local optimization and server to employ weighted averaging as the aggregation function. Each client j with train sample size n_j performs $\tau_j = e \cdot \lceil \frac{n_j}{m} \rceil$ local updates in each round, where e is the number of local epochs, and m is the batch size. `FedProx` and `FedNova` are modified versions of `FedAvg`, but the number of local updates per round in both algorithms is the same as `FedAvg`.

In `FedAvg` and its variants, the batch size and number of local updates per round are coupled to each other because the batch size determines both the number of training samples in the batch and the number of local updates per round. The algorithm proposed by [18] (we refer to it as *federated constant-mini-batches* (`FedCMB`) addresses this issue by specifying the number of local updates (or mini-batches) and batch size using two different hyper-parameters, where the clients perform a constant (and multiple) number of local updates per round independently of the given batch size.

`FedFGD`, on the other hand, enforces the clients to conduct a single local update using all samples of their local datasets in each round and leverages weighted averaging as the aggregation function at the server. `FedSMB` [18] is a variant of `FedFGD` in which the clients perform one local update per round using a single *mini-batch of samples* instead of the whole dataset.

Given that, `FedFGD` satisfies the necessary conditions outlined in Proposition 2.1, which is not the case for `FedAvg`, `FedProx`, and `FedNova` because they conduct *multiple* local updates per round. Regarding `FedSMB`, we can replace $\mathcal{D}_j^l$ (the local dataset of client j) with $\mathcal{B}_j^l$ (a mini-batch of size m from $\mathcal{D}_j^l$), and $\mathcal{D}$ (centralized dataset) with $\bigcup_{j=1}^k \mathcal{B}_j^l$ (the aggregation of the local mini-batches of k clients) in the proof of Proposition 2.1 (Appendix A). For a given communication round, the federated training with `FedSMB` of batch size m becomes equivalent to the centralized training with `MBGD` of batch size $m \cdot k$, where k is the number of clients. Notice that an additional assumption should be made here: the train sample sizes of the clients are identical and divisible by the mini-batch size m.

Corollary 2.2. `FedFGD` *and* `FedSMB` *can be incorporated as training algorithms in UPFL.*

2.3.2 Optimizers. The stochastic gradient descent based optimizers including *SGD*, *Adam*, and its variants employ different momentum functions to compute the final gradient value for updating the model parameters. The *SGD* optimizer calculates the final gradient value as follows:

$$v_i = \xi \cdot v_{i-1} + g_i,$$

where ξ is the momentum value, and $v_0 = 0$. In other words, the momentum function of *SGD* takes the momentum value ξ and the gradient values from the first iteration to the current iteration $i > 1$ as inputs, and linearly combines them:

$$\mathcal{V}_{SGD}(g_1, \ldots, g_i, \xi) = \xi^{i-1} \cdot g_1 + \ldots + \xi^{i-k} \cdot g_k + \ldots + g_i, 1 \leq k \leq i.$$

Given Equation 7, the momentum function of *SGD* is linear, where $\alpha_k = \xi^{i-k}$.

The *Adam* optimizer computes the final gradient value g_f as follows:

$$v_i = \beta_1 \cdot v_{i-1} + (1 - \beta_1) \cdot g_i, \qquad \gamma_i = \beta_2 \cdot \gamma_{i-1} + (1 - \beta_2) \cdot (g_i)^2$$

$$\bar{v_i} = \frac{v_i}{1 - (\beta_1)^i}, \quad \bar{\gamma_i} = \frac{\gamma_i}{1 - (\beta_2)^i}, \qquad g_f = \frac{\bar{v_i}}{\sqrt{\bar{\gamma_i}} + \epsilon},$$

where the coefficients β_1 and β_2 are hyper-parameters, and $v_0 = \gamma_0 = 0$. According to the formulas, Adam's momentum function is not linear (notice $(g_i)^2$ and $\sqrt{\bar{\gamma_i}}$). Likewise, the other *adaptive* optimizers including AdaMax [11], Adadelta [29], and Adagrad [2] employ non-linear momentum functions.

Corollary 2.3. *SGD employs a linear momentum function, and thus, SGD satisfies the necessary condition for UPFL.*

2.3.3 Loss Functions. A loss function $\mathcal{L}(y, \hat{y})$ provides a criterion that measures the distance between the value predicted by the model, i.e. $\hat{y}$, and the target value y for a particular input x. Given a batch-independent model, the binary cross-entropy loss function, for example, calculates the loss value as follows:

$$\mathcal{L}(y, \hat{y}) = -(y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y}))$$

$$\hat{\mathcal{Y}} = [\hat{y}_1, \ldots, \hat{y}_m],\ \mathcal{L}(\mathcal{Y}, \hat{\mathcal{Y}}) = [\mathcal{L}(y_1, \hat{y}_1), \ldots, \mathcal{L}(y_m, \hat{y}_m)]$$

This indicates binary cross-entropy is batch-independent. This is also the case for other popular loss functions such as multi-class cross-entropy and focal loss.

Corollary 2.4. *All widely-used loss functions including cross-entropy and focal loss are batch-independent. Thus, they can be employed as loss function in UPFL.*

2.3.4 Model Layers. A model layer can be considered as a function that maps a given input to output. For instance, the mapping function of BatchNorm is as follows:

$$\mu_X = \frac{1}{m} \cdot \sum_{i=1}^{m} X_i, \quad \sigma_X^2 = \frac{1}{m} \cdot \sum_{i=1}^{m} (X_i - \mu_X)^2,$$

$$\hat{X}_i = \frac{X_i - \mu_X}{\sqrt{\sigma_X^2 + \epsilon}}, \quad Y_i = \gamma \cdot \hat{X}_i + \beta = \mathcal{F}_{\gamma,\beta}(X_1, \ldots, X_m),$$

where $[X_1, \ldots, X_m]$ is the input batch of size m, X_i is the i^{th} input element in the batch, Y_i is the corresponding output, μ_X and σ_X^2 are the mean and variance of the input batch, respectively, ϵ is a small constant for numerical stability, and γ and β are the BatchNorm's learnable parameters.

As another example, the output of each neuron in the fully-connected layer is a linear transformation of the input:

$$Y_i = w \cdot X_i + b = \mathcal{F}_{w,b}(X_i),$$

where w and b are the learnable parameters of the layer.

According to the equations, the output of BatchNorm for a particular input element depends on the other input values in the batch, and consequently, the BatchNorm layer is not batch-independent. The fully-connected layer, on the other hand, computes the output for each element independently of the other elements in the batch. Thus, the fully-connected layer is batch-independent. The batch-independence property also holds for other layers widely used in image vision such as convolutional, max/average-pooling, GroupNorm, as well as activation functions including ReLU. These layers are deterministic too.

Corollary 2.5. *Most of the widely-adopted layers in image vision such as the convolutional, fully-connected (linear), max/average-pooling, and GroupNorm, and all activation functions including ReLU are batch-independent and deterministic, and as a result, they can be incorporated in UPFL.*

Note that if a component does not satisfy the necessary conditions for UPFL, the weights from the federated and centralized models become different (Proof of Proposition 2.1 in Appendix A). Theoretically, this does not imply the federated model delivers lower utility compared to the centralized model. However, the component typically causes utility reduction in practice under NonIID settings as also shown in prior studies [15, 17, 24]. For instance, `FedAvg`, which performs multiple local updates per round, achieves lower accuracy than centralized training [14, 17]. BatchNorm, which is not a batch-independent layer, dramatically reduces utility in federated environments [15]. Adaptive optimizers including Adam, which are not based on a linear momentum function, result in significant accuracy reduction in FL compared to centralized training [24].

3 EXPERIMENTAL VALIDATION

We experimentally validate the theoretical results from Section 2. In the following, we first describe the datasets, models, and training procedures used in the experiments (more details in Appendix B), and then provide the results.

Datasets. The CIFAR-10 dataset [12] includes 50000 train and 10000 test samples of shape 32×32 from 10 classes. The Imagenette dataset [7] is a subset of Imagenet [1], containing 9469 train and 3925 test samples from 10 "easily classified" classes. The feature values of the samples in both datasets are divided by 255. The samples of Imagenette are resized to 128×128.

Models. We employ the VGG-6 architecture from [21, 25] and the original implementation of ResNet-18 [5] from PyTorch [22]. VGG-6 consists of the convolutional, max-pooling, average-pooling, and fully-connected layers. ResNet-18 includes GroupNorm in addition to the aforementioned layers. Both models use ReLU as the activation function.

Centralized training. The VGG-6 and ResNet-18 models are trained on 25000 training samples from CIFAR-10 and 5000 training samples from Imagenette, respectively (due to the memory limitation regarding `FGD`). The loss function is cross-entropy; optimizer is SGD with momentum of 0.9. For the `FGD` algorithm, the VGG-6

Table 1: Mean square error (MSE) between the model weights from the centralized and FedFGD-based UPFL. The centralized and federated models have the same weights, ignoring the numerical errors during computations; IID: 10 classes per client; Moderately-NonIID: 5 classes per client; Extremely-NonIID: 1 class per client.

(a) VGG-6-CIFAR-10

Iteration/Round	IID	Moderately-NonIID	Extremely-NonIID
1	4×10^{-20}	4×10^{-20}	4×10^{-20}
10	2×10^{-17}	2×10^{-17}	2×10^{-17}
100	1×10^{-15}	1×10^{-15}	8×10^{-16}
1000	1×10^{-8}	3×10^{-8}	1×10^{-8}
10000	5×10^{-7}	6×10^{-7}	3×10^{-7}

(b) ResNet-18-Imagenette

Iteration/Round	IID	Moderately-NonIID	Extremely-NonIID
1	2×10^{-19}	2×10^{-19}	2×10^{-19}
10	2×10^{-15}	1×10^{-15}	2×10^{-15}
100	1×10^{-11}	2×10^{-11}	4×10^{-12}
1000	1×10^{-9}	2×10^{-9}	1×10^{-9}
2500	3×10^{-9}	3×10^{-9}	2×10^{-9}

Table 2: Test accuracy of the models from the centralized and UPFL. The federated models achieve very close accuracy values compared to the corresponding centralized models independent of data distribution across the clients.

(a) FGD/FedFGD

Model	Dataset	Centralized	IID	Moderately-NonIID	Extremely-NonIID
VGG-6	CIFAR-10	64.66±0.40	64.59±0.28	64.55±0.30	64.68±0.28
ResNet-18	Imagenette	65.69±0.10	65.68±0.29	65.47±0.49	65.67±0.17

(b) MBGD/FedSMB

Model	Dataset	Centralized	IID	Moderately-NonIID	Extremely-NonIID
VGG-6	CIFAR-10	75.41±0.16	75.42±0.10	75.44±0.32	75.45±0.26
ResNet-18	Imagenette	62.34±0.25	62.46±0.25	62.66±0.88	62.26±0.69

and ResNet-18 models are trained for 10000 and 2500 iterations, respectively. The initial learning rates are 0.01 and 0.001, which are reduced by factor of 0.99 every 20 and 5 iterations for the VGG-6-CIFAR-10 and ResNet-18-Imagenette case studies, respectively. For the MBGD algorithm, VGG-6 and ResNet-18 are trained with learning rate of 0.0125 for 50 epochs (i.e. 12500 and 2500 iterations, respectively) with batch size of 100.

Federated training. The federated environments consist of 10 clients, where the centralized CIFAR-10 and Imagenette datasets (with 25000 and 5000 training samples, respectively) have evenly been distributed across the clients. We consider three different class distributions among the clients: (1) IID, where each client has samples from all 10 classes, (2) moderately NonIID, in which each client has samples from only 5 classes, and (3) extremely NonIID, where the clients have samples only from a single class. The federated training algorithms are FedFGD and FedSMB, and all clients are selected in each communication round. The loss function, optimizer, models, learning rate decay procedure, and number of communication rounds (iterations) are the same as the corresponding centralized training. Moreover, the global models are initialized with the same weights as the associated centralized models. Note that the federated environments are utility-preserving according to Section 2.

Results. Table 1 lists the mean square error (MSE) between the model weights from the centralized and FedFGD-based UPFL for both VGG-6-CIFAR-10 and ResNet-18-Imagenette case studies. According to the table, the MSE values are close to zero, and as a result, the weight of the models are identical up to numerical precision. The insignificant difference between the model weights is due to the numerical errors from gradient computation, weighted averaging, and etc. Given that, the centralized and federated models have the same weights, which validates Proposition 2.1.

Table 2 shows the test accuracy values achieved by the models trained using FGD and MBGD in centralized setting and using FedFGD and FedSMB in utility-preserving federated setting. The federated models deliver accuracy values highly close to those from the corresponding centralized models regardless of the data distribution across the clients. This indicates UPFL preserves utility compared to centralized training (indeed implied by our theoretical analysis).

4 PRACTICAL APPLICATION

According to our theoretical analysis and experimental validation, (1) UPFL achieves the same utility as centralized training, (2) many of the components popular in centralized training such as the cross-entropy loss function, SGD optimizer, and convolutional layer can be incorporated into UPFL too, and (3) one of the main characteristics that differentiates UPFL from ordinary FL is the training algorithm, or more precisely, the number of local updates per communication round in the training algorithm. In UPFL, the clients perform exactly one local update per round, whereas ordinary FL enforces the clients to conduct multiple local updates per round. Given that, we delve more deeply into the impact of the number of local updates on utility and communication efficiency. Here is a brief description of the datasets, models, and training procedures employed in the experiments (more details in Appendix B).

Datasets. CIFAR-100 includes samples from 100 classes, making it a more challenging dataset than its CIFAR-10 counterpart. We also created our own ImageNet subset, *ImageNet-50*, which is more difficult to classify compared to Imagenette. ImageNet-50 contains 25000 training (500 samples per class) and 5000 (100 samples per class) test samples of shape 160×160 from 50 classes, which are easy-to-classify using pretrained ResNet-18/34/50, DenseNet-121/161/169, and EfficientNet-b0/b1. For data augmentation, the train samples of CIFAR-100 are randomly cropped after 4×4 padding, horizontally flipped, and normalized using mean and standard deviation (ST) of the dataset. Similarly, we apply horizontal flipping, and random cropping of shape 128×128 to the train samples of ImageNet-50, and normalize them using mean and ST of ImageNet.

Models. We adopt the GroupNorm-based VGG-11 and ResNet-18 as models. Both models contain only batch-independent and deterministic layers.

Centralized training. We train VGG-11 and ResNet-18 on CIFAR-100 and ImageNet-50, respectively, in three different configurations: (1) zero-momentum and zero-weight-decay, (2) momentum of 0.9 and zero-weigh-decay, and (3) momentum of 0.9 and weight decay of 0.0005. The loss function is cross-entropy, optimizer is SGD, and training algorithm is MBGD with batch size of 100.

Federated training. The federated environments in both VGG-11-CIFAR-100 and ResNet18-ImageNet-50 case studies include 10 clients. In the former, each client has 5000 samples from 10 classes, wheres the clients have 2500 samples from 5 classes in the latter. Thus, the class distribution across the clients can be contemplated as highly NonIID in both cases. The optimizer, loss function, and configurations (i.e. momentum and weight decay) are the same as those in centralized training. We consider three different number of local updates per round: single, few, and many, where the clients perform 1, 5, and 100 local updates per round, respectively. The data augmentation at the clients is the same as centralized training.

Our observations show that applying momentum and weight decay on the client-side does not provide an accuracy gain if clients perform multiple local updates per round, and thus, we apply them on the server-side as follows:

$$g_i = \frac{\sum_{j=1}^{k} n_{i,j} \cdot g_{i,j}^{l}}{\sum_{j=1}^{k} n_{i,j}}, \quad u_i = \xi_s \cdot u_{i-1} + g_i + \lambda_s \cdot w_{i-1}, \quad w_i = w_{i-1} - \eta \cdot u_i,$$

where $g_{i,j}^{l}$ is the accumulated gradient value from client j in round i, $n_{i,j}$ is the number of samples used for training, ξ_s and λ_s are the server-side momentum and weight decay, respectively, and u_0=0.

Results. Tables 3-4 list the test accuracy values and communication rounds for different number of local updates per round and batch sizes, respectively. As shown in the tables, (1) UPFL (single local update per round) *fully* takes advantage of momentum and weight decay to improve accuracy compared to centralized training, but incurs substantial communication overhead, (2) ordinary FL (multiple local updates per round) remarkably enhances communication efficiency; however, it *partially* benefits from momentum and weight decay, (3) smaller batch sizes deliver higher accuracy than larger ones. In other words, UPFL sacrifices network efficiency to achieve ideal utility and to benefit from momentum and weight decay fully. Ordinary FL, on the other hand, aims to improve communication efficiency, which leads to utility reduction and partial benefit from the aforementioned techniques.

Because both utility and network communication are crucial factors in FL, this question arises: *How can a federated training algorithm take advantage of momentum and weight decay considerably (ideally completely) while maintaining practical communication efficiency by performing multiple local updates per round?*

As an initial step towards addressing that question, we propose a method called *weighted gradient accumulation* (WGA), in which the local gradients of the clients from initial iterations (updates) have more weights than those in the final iterations during gradient accumulation on the client side. The logic behind WGA is that the local models in the initial iterations are closer to the global model than those in final ones. Assuming that a given client performs τ local updates per round, WGA computes the final accumulated gradient as follows:

$$g_{acc}^{l} = \alpha_1 \cdot g_1^{l} + \ldots + \alpha_\tau \cdot g_\tau^{l}, \ \textit{where}\ \alpha_1 > \alpha_2 > \ldots > \alpha_\tau = 1 \quad (8)$$

In our experiments, for instance, we set α_1=τ, and α_i=$(i-2)\cdot \frac{1-\frac{\tau}{2}}{\tau-2} + \frac{\tau}{2}$ for $2 \le i \le \tau$. That is, the coefficients of the first and second iterations are τ and $\frac{\tau}{2}$, respectively, and are linearly reduced from $\frac{\tau}{2}$ to 1 in the remaining iterations.

Table 5 and Figure 1 show the test accuracy and communication efficiency of WGA (combined with FedCMB) and FedCMB (τ=5) as baseline, respectively. WGA provides more accuracy gain using the momentum and weight decay techniques, and enhances communication efficiency compared to FedCMB.

5 RELATED WORK

Some of the related work on the utility challenge in FL are experimental works that investigate the model performance in different federated settings. The study from [8] performs extensive experiments using various datasets to understand the impact of NonIID data on the performance of the federated models. It concludes that distributed learning over NonIID data is a burdensome problem whose difficulty highly depends on the degree of the data heterogeneity. Similarly, the work by [23] evaluates the impact of the data heterogeneity on FL for medical imaging, and suggest several strategies to mitigate the utility reduction in the NonIID federated scenarios. Our analysis, on the other hand, indicates that the adverse impact of NonIID data on utility is only the case for ordinary FL, where clients perform multiple local updates per round. In other words, NonIID data is not a challenge for UPFL, which can provide the same accuracy as the centralized training regardless of data distribution across clients.

Another category of studies including FedProx [14] and FedNova [27] propose new training algorithms to mitigate the performance issues of FedAvg in the NonIID federated environments as we discussed in Section (1). Although they partially alleviate the utility problem of FedAvg, they cannot still achieve the same utility as centralized training. Our study, however, theoretically and experimentally demonstrate that a model trained under the UPFL environment can provide utility identical to centralized training.

The other line of work focuses on simple algorithms in NonIID federated environments. *sPLINK* [19] develops federated versions of the chi-square, linear, and logistic regression algorithms. Similarly, *Flimma* [30] and *Fever-PCA* [3] implement the federated linear regression and principal component analysis algorithms. These studies indeed demonstrate that the federated algorithms are equivalent to the corresponding centralized counterparts, and therefore, they provide the same utility as centralized training. Our study takes a similar approach, but in the context of deep neural networks, which is more challenging and complicated than the aforementioned methods.

Table 3: Test accuracy versus the number of local updates per round (τ) and batch size (m) with and without server-side momentum (ξ_s) and weight decay (λ_s): Single/multiple local update(s) per round can fully/partially benefit from momentum and weight decay; Smaller batch sizes achieve higher accuracy; Δ: accuracy gain from momentum and weight decay.

(a) VGG-11-CIFAR-100

Environment	Algorithm	τ	m	(ξ_s=0, λ_s=0)	(ξ_s=0.9, λ_s=0)	(ξ_s=0.9, λ_s=5e-4)	Δ
Centralized	MBGD	NA	100	66.47±0.07	68.55±0.15	72.55±0.09	6.08
UPFL	FedSMB	1	10	66.54±0.41	68.65±0.19	72.50±0.21	5.96
FL	FedCMB	5	10	66.36±0.35	67.33±0.01	68.71±0.19	2.35
FL	FedAvg	5	1000	62.26±0.09	63.06±0.31	62.35±0.06	0.80
FL	FedCMB	100	10	66.78±0.14	66.66±0.54	67.19±0.47	0.41
FL	FedAvg	100	50	64.68±0.11	64.49±0.31	64.86±0.39	0.18

(b) ResNet-18-ImageNet-50

Environment	Algorithm	τ	m	(ξ_s=0, λ_s=0)	(ξ_s=0.9, λ_s=0)	(ξ_s=0.9, λ_s=5e-4)	Δ
Centralized	MBGD	NA	100	64.58±0.25	67.73±0.13	74.13±0.17	9.55
UPFL	FedSMB	1	10	64.55±0.06	67.83±0.20	74.20±0.13	9.65
FL	FedCMB	5	10	64.35±0.18	65.62±0.61	66.67±0.60	2.32
FL	FedAvg	5	500	60.95±0.69	62.36±0.37	62.48±0.46	1.53
FL	FedCMB	100	10	65.19±0.80	65.23±0.31	65.68±0.52	0.49
FL	FedAvg	100	25	64.95±0.20	65.65±0.22	65.08±0.37	0.70

Table 4: Communication rounds versus local updates per round (τ): More local updates per round requires fewer communication rounds for model convergence, improving communication efficiency.

(a) VGG-11-CIFAR-100

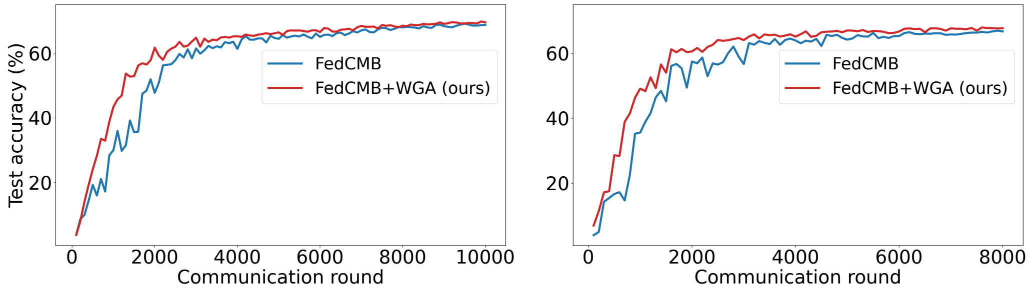
Environment	Algorithm	τ	m	(ξ_s=0, λ_s=0)	(ξ_s=0.9, λ_s=0)	(ξ_s=0.9, λ_s=5e-4)
UPFL	FedSMB	1	10	90000	55000	90000
FL	FedCMB	5	10	10000	9000	10000
FL	FedAvg	5	1000	5000	4000	5000
FL	FedCMB	100	10	2500	2000	2500
FL	FedAvg	100	50	2500	2000	2500

(b) ResNet-18-ImageNet-50

Environment	Algorithm	τ	m	(ξ_s=0, λ_s=0)	(ξ_s=0.9, λ_s=0)	(ξ_s=0.9, λ_s=5e-4)
UPFL	FedSMB	1	10	98000	60000	98000
FL	FedCMB	5	10	8000	7000	8000
FL	FedAvg	5	500	4000	3500	4000
FL	FedCMB	100	10	4000	4000	4000
FL	FedAvg	100	25	3500	3500	3500

Table 5: Weighted gradient accumulation (WGA) improves test accuracy, and benefits more from momentum and weight decay compared to the baseline.

Algorithm	Model	Dataset	τ	ξ_s	λ_s	Accuracy	Δ
FedCMB	VGG-11	CIFAR-100	5	0.0	0.0	66.36±0.35	—
FedCMB	VGG-11	CIFAR-100	5	0.9	5e-4	68.71±0.19	2.35
FedCMB+WGA (ours)	VGG-11	CIFAR-100	5	0.9	5e-4	**69.34**±0.08	**2.98**
FedCMB	ResNet-18	ImageNet-50	5	0.0	0.0	64.35±0.18	—
FedCMB	ResNet-18	ImageNet-50	5	0.9	5e-4	66.67±0.60	2.32
FedCMB+WGA (ours)	ResNet-18	ImageNet-50	5	0.9	5e-4	**67.77**±0.13	**3.42**



(a) VGG-11-CIFAR-100 **(b) ResNet-18-ImageNet-50**

Figure 1: Weighted gradient accumulation (WGA) enhances communication efficiency compared to the baseline.

6 CONCLUSION AND FUTURE WORK

In the context of deep learning, we define *utility-preserving federated learning* as an environment in which the DNN model is batch-independent and deterministic, loss function is batch-independent too, optimizer employs linear momentum function, and federated training algorithm selects all clients to participate in training, instruments them to perform a single local update per round, and enforces the server to apply weighted averaging during aggregation. Next, we theoretically prove and experimentally validate UPFL can provide the same utility as centralized training, and thus, it preserves the model utility compared to centralized training.

Our analysis shows the main property that distinguishes UPFL from ordinary FL is the number of local updates per round. The clients in UPFL carry out exactly one local update per round, while ordinary FL instruments the clients to perform multiple local updates per round. Our evaluations demonstrate the former incurs considerable communication overhead, but it can fully benefit from the momentum and weight decay techniques to improve accuracy. The latter remarkably enhances communication efficiency; however, it can partially take advantage of the before-mentioned techniques. Given that, we propose *weighted gradient accumulation* to combine the benefits of UPFL and ordinary FL, and illustrate it can benefit from momentum and weight decay more akin to UPFL, while providing higher communication efficiency similar to ordinary FL.

We focus on cross-silo federated learning [10], where all clients are selected in each round. Theoretical analysis and empirical study of UPFL and WGA for cross-device FL, in which a fraction of clients are chosen to participate in training, is an interesting direction for future works.

ACKNOWLEDGMENTS

This work has been funded by the German Federal Ministry of Education and Research and the Bavarian State Ministry for Science and the Arts. The authors of this work take full responsibility for its content.

REFERENCES

[1] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. IEEE, 248–255.

[2] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research* 12, 7 (2011).

[3] Anne Hartebrodt, Reza Nasirigerdeh, David B Blumenthal, and Richard Röttger. 2021. Federated principal component analysis for genome-wide association studies. In *2021 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1090–1095.

[4] Mahmoud Hassaballah and Ali Ismail Awad. 2020. *Deep learning in computer vision: principles and applications*. CRC Press.

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

[6] Eric Horvitz and Deirdre Mulligan. 2015. Data, privacy, and the greater good. *Science* 349, 6245 (2015), 253–255.

[7] Jeremy Howard. 2019. imagenette. https://github.com/fastai/imagenette/

[8] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip Gibbons. 2020. The non-iid data quagmire of decentralized machine learning. In *International Conference on Machine Learning*. PMLR, 4387–4398.

[9] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*. PMLR, 448–456.

[10] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2019. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977* (2019).

[11] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[12] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[13] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine* 37, 3 (2020), 50–60.

[14] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated Optimization in Heterogeneous Networks. In *Proceedings of Machine Learning and Systems*, Vol. 2. 429–450.

[15] Xiaoxiao Li, Meirui JIANG, Xiaofei Zhang, Michael Kamp, and Qi Dou. 2021. FedBN: Federated Learning on Non-IID Features via Local Batch Normalization. In *International Conference on Learning Representations*.

[16] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*. 2980–2988.

[17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 1273–1282.

[18] Reza Nasirigerdeh, Mohammad Bakhtiari, Reihaneh Torkzadehmahani, Amirhossein Bayat, Markus List, David B Blumenthal, and Jan Baumbach. 2020. Federated Multi-Mini-Batch: An Efficient Training Approach to Federated Learning in Non-IID Environments. *arXiv preprint arXiv:2011.07006* (2020).

[19] Reza Nasirigerdeh, Reihaneh Torkzadehmahani, Julian Matschinske, Tobias Frisch, Markus List, Julian Späth, Stefan Weiss, Uwe Völker, Esa Pitkänen, Dominik Heider, et al. 2022. sPLINK: a hybrid federated tool as a robust alternative to meta-analysis in genome-wide association studies. *Genome Biology* 23, 1 (2022), 1–24.

[20] Daniel W Otter, Julian R Medina, and Jugal K Kalita. 2020. A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems* 32, 2 (2020), 604–624.

[21] Dae Young Park, Moon-Hyun Cha, Daesin Kim, Bohyung Han, et al. 2021. Learning student-friendly teacher networks for knowledge distillation. *Advances in Neural Information Processing Systems* 34 (2021).

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga,

Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035.

[23] Liangqiong Qu, Niranjan Balachandar, and Daniel L Rubin. 2021. An Experimental Study of Data Heterogeneity in Federated Learning Methods for Medical Imaging. *arXiv preprint arXiv:2107.08371* (2021).

[24] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. 2021. Adaptive Federated Optimization. In *International Conference on Learning Representations*. https://openreview.net/forum?id=LkFG3lB13U5

[25] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*.

[26] Michael Wainberg, Daniele Merico, Andrew Delong, and Brendan J Frey. 2018. Deep learning in biomedicine. *Nature biotechnology* 36, 9 (2018), 829–838.

[27] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. 2020. Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., 7611–7623.

[28] Yuxin Wu and Kaiming He. 2018. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*. 3–19.

[29] Matthew D Zeiler. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701* (2012).

[30] Olga Zolotareva, Reza Nasirigerdeh, Julian Matschinske, Reihaneh Torkzadehmahani, Mohammad Bakhtiari, Tobias Frisch, Julian Späth, David B Blumenthal, Amir Abbasinejad, Paolo Tieri, et al. 2021. Flimma: a federated and privacy-aware tool for differential gene expression analysis. *Genome biology* 22, 1 (2021), 1–26.

A PROOFS

Additivity. A *gradient function* has the *additive* property with respect to input batch if the underlying loss function and model are batch-independent and deterministic.

Proof. Let $\mathcal{L}$ be the loss function, $\mathcal{M}_W$ be the model, and w be a learnable parameter of the model. Because $\mathcal{L}$ is batch-independent and $\mathcal{M}_W$ is deterministic and batch-independent, the resulting gradient function $\mathcal{G}_w$ is deterministic and batch-independent too. Therefore, $\mathcal{G}_w$ calculates the gradient value for each sample deterministically, and independently of the other samples in the batch, and as a result, the gradient values can be grouped into arbitrary batches. That is, if $\mathcal{B}, \mathcal{B}_1, \ldots, \mathcal{B}_k$ are batches of sizes $m, m_1, \ldots, m_k$, respectively, and $\mathcal{B} = \mathcal{B}_1 \bigcup \ldots \bigcup \mathcal{B}_k$ and $\mathcal{B}_p \bigcap \mathcal{B}_q = \emptyset$ for $p \neq q$, we have:

$$\mathcal{G}_w(W, \mathcal{B}, Y_{\mathcal{B}}) = \frac{\partial \mathcal{L}(Y_{\mathcal{B}}, \mathcal{M}_W(\mathcal{B}))}{\partial w}$$
$$= \frac{\partial \mathcal{L}(Y_1, \mathcal{M}_W(S_1))}{\partial w} \bigcup \ldots \bigcup \frac{\partial \mathcal{L}(Y_m, \mathcal{M}_W(S_m))}{\partial w}$$
$$= \bigcup_{r=1}^{m_1} \frac{\partial \mathcal{L}(Y_r, \mathcal{M}_W(S_r))}{\partial w} \bigcup \ldots \bigcup_{r=m-m_k+1}^{m} \frac{\partial \mathcal{L}(Y_r, \mathcal{M}_W(S_r))}{\partial w}$$
$$= \frac{\partial \mathcal{L}(Y_{\mathcal{B}_1}, \mathcal{M}_W(\mathcal{B}_1))}{\partial w} \bigcup \ldots \bigcup \frac{\partial \mathcal{L}(Y_{\mathcal{B}_k}, \mathcal{M}_W(\mathcal{B}_k))}{\partial w}$$
$$= \mathcal{G}_w(W, \mathcal{B}_1, Y_{\mathcal{B}_1}) \bigcup \ldots \bigcup \mathcal{G}_w(W, \mathcal{B}_k, Y_{\mathcal{B}_k}) \qquad \square$$

□

Proposition 2.1. Federated learning and centralized training using full gradient descent are equivalent if the (1) model is batch-independent and deterministic, (2) loss function is batch-independent, (3) optimizer uses a linear momentum function, and (4) training algorithm selects all clients, the clients perform a single local update per communication round using all samples of their local datasets, and the server employs weighted averaging as the aggregation function.

Proof. Assume a centralized environment with dataset $\mathcal{D} = [S_1, ..., S_n]$ of n samples and the corresponding target vector $Y = [y_1, ..., y_n]$, and a federated setting consisting of k clients, where $\mathcal{D}_j^l$, Y_j^l, and n_j indicate the local dataset, target vector, and train sample size of client j, respectively. Moreover, the aggregation of the clients' data is the same as the centralized data: $\mathcal{D} = \bigcup_{j=1}^{k} \mathcal{D}_j^l$, $Y = \bigcup_{j=1}^{k} Y_j^l$, $n = \sum_{j=1}^{k} n_j$, $\mathcal{D}_p^l \bigcap \mathcal{D}_q^l = \emptyset$ and $Y_p^l \bigcap Y_q^l = \emptyset$ for $p \neq q$.

In the centralized setting, model $\mathcal{M}_W$ characterized by a vector of parameters W is trained on dataset $\mathcal{D}$. In the federated environment, the clients train the same model $\mathcal{M}_W$ on their local datasets. The initial value of $w \in W$ for both federated and centralized models are identical. The clients and centralized training utilize the same optimizer O and loss function $\mathcal{L}$.

Let $w \in W$ be a model parameter, w_i^f and w_i^c be the (global) value of w from the federated and centralized training in iteration i, respectively. We prove *by induction* that $w_i^f = w_i^c$ for every iteration $i \geq 1$.

◇ *Base case (i=0)*: $w_0^f = w_0^c$ according to the assumption that the initial values of w_i^f and w_i^c are identical.

◇ *Inductive hypothesis*: Assume $w_1^f = w_1^c, \ldots, w_{i-1}^f = w_{i-1}^c$.

◇ *Inductive step*: Based on the properties of the federated training algorithm, all k clients are selected in each round, and each selected client j first computes the local values of gradients for its n_j train samples, then calculates the local value of gradient in the current iteration by taking average over the gradients from all n_j samples, and finally computes the local value of w:

$$G_{i,j}^l = [g_{i,j,1}, \ldots, g_{i,j,n_j}], \quad g_{i,j}^l = \frac{1}{n_j} \sum_{h=1}^{n_j} g_{i,j,h}$$
$$w_{i,j}^l = w_{i-1}^f - \eta \cdot (\alpha_p \cdot g_{p,j}^l + \ldots + \alpha_i \cdot g_{i,j}^l)$$

Notice that the momentum function of the optimizer is *linear*; thus, a linear combination of the final gradient values from iterations p to i is employed to calculate the parameter's local value.

On the server, the global value of w is calculated by aggregating the local values of w from all k clients using *weighted averaging*:

$$w_i^f = \frac{1}{\sum_{j=1}^{k} n_j} \cdot (n_1 \cdot w_{i,1}^l + \ldots + n_k \cdot w_{i,k}^l)$$
$$= \frac{1}{n} \cdot n_1 \cdot (w_{i-1}^f - \eta \cdot (\alpha_p \cdot g_{p,1}^l + \ldots + \alpha_i \cdot g_{i,1}^l)) + \ldots$$
$$\ldots + \frac{1}{n} \cdot n_k \cdot (w_{i-1}^f - \eta \cdot (\alpha_p \cdot g_{p,k}^l + \ldots + \alpha_i \cdot g_{i,k}^l))$$
$$= w_{i-1}^f - \frac{1}{n} \cdot \eta \cdot \alpha_p \cdot (n_1 \cdot g_{p,1}^l + \ldots + n_k \cdot g_{p,k}^l) - \ldots$$
$$\ldots - \frac{1}{n} \cdot \eta \cdot \alpha_i \cdot (n_1 \cdot g_{i,1}^l + \ldots + n_k \cdot g_{i,k}^l)$$
$$= w_{i-1}^f - \eta \cdot \alpha_p \cdot \frac{1}{n} \cdot (\sum_{h=1}^{n_1} g_{p,1,h} + \ldots + \sum_{h=1}^{n_k} g_{p,k,h}) - \ldots$$
$$\ldots - \eta \cdot \alpha_i \cdot \frac{1}{n} \cdot (\sum_{h=1}^{n_1} g_{i,1,h} + \ldots + \sum_{h=1}^{n_k} g_{i,k,h})$$

The aggregation of the clients' data is identical to the centralized data, model is *deterministic* and *batch-independent*, and loss function is *batch-independent.* Thus, the gradient function holds the *additive property*, and the gradient values from the clients in a particular iteration can be grouped into a single batch of gradients corresponding to all samples:

$$= w_{i-1}^f - \eta \cdot \alpha_p \cdot \frac{1}{n} \cdot (g_{p,1} + \ldots + g_{p,n}) - \ldots$$
$$\ldots - \eta \cdot \alpha_i \cdot \frac{1}{n} \cdot (g_{i,1} + \ldots + g_{i,n})$$
$$= w_{i-1}^f - \eta \cdot (\alpha_p \cdot \frac{1}{n} \cdot \sum_{s=1}^{n} g_{p,s} + \cdots + \alpha_i \cdot \frac{1}{n} \cdot \sum_{s=1}^{n} g_{i,s})$$
$$= w_{i-1}^f - \eta \cdot (\alpha_p \cdot g_p + \ldots + \alpha_i \cdot g_i)$$

Based on the inductive hypothesis, $w_{i-1}^f = w_{i-1}^c$, so:

$$w_i^f = w_{i-1}^c - \eta \cdot (\alpha_p \cdot g_p + \ldots + \alpha_i \cdot g_i) = w_i^c$$

□

B REPRODUCIBILITY

Table 6: Experimental settings associated with result tables; FGD: full gradient descent; MBGD: mini-batch gradient descent; WGA: weighted gradient accumulation; N: number of training samples; m: batch size; η: learning rate; ω: learning rate decay factor; κ: learning rate decay period (iterations); ξ_s: server-side momentum; λ_s: server-side weight decay; τ: number of local updates per round;

(a) Centralized

Table #	Config	Model	Dataset	N	Algorithm	m	Iterations	η	ω	κ
1a & 2a	—	VGG-6	CIFAR-10-Subset	25000	FGD	—	10000	0.01	0.99	20
1b & 2a	—	ResNet-18	Imagenette-Subset	5000	FGD	—	2500	0.001	0.99	5
2b	—	VGG-6	CIFAR-10-Subset	25000	MBGD	100	12500	0.0125	—	—
2b	—	ResNet-18	Imagenette-Subset	5000	MBGD	100	2500	0.0125	—	—
3a & 4a	(ξ_s=0,λ_s=0)	VGG-11	CIFAR-100	50000	MBGD	100	90000	0.05	0.99	500
3a & 4a	(ξ_s=0.9,λ_s=0)	VGG-11	CIFAR-100	50000	MBGD	100	55000	0.05	0.99	500
3a & 4a	(ξ_s=0.9,λ_s=5e-4)	VGG-11	CIFAR-100	50000	MBGD	100	90000	0.025	0.99	500
3b & 4b	(ξ_s=0.0,λ_s=0.0)	ResNet-18	ImageNet-50	25000	MBGD	100	98000	0.05	0.5	18750
3b & 4b	(ξ_s=0.9,λ_s=0.0)	ResNet-18	ImageNet-50	25000	MBGD	100	60000	0.025	0.5	18750
3b & 4b	(ξ_s=0.9,λ_s=5e-4)	ResNet-18	ImageNet-50	25000	MBGD	100	98000	0.05	0.5	18750

(b) Utility-preserving federated learning

Table #	Config	Model	Dataset	N	Algorithm	m	Iterations	η	ω	κ
1a & 2a	—	VGG-6	CIFAR-10-Subset	25000	FedFGD	—	10000	0.01	0.99	20
1b & 2a	—	ResNet-18	Imagenette-Subset	5000	FedFGD	—	2500	0.001	0.99	5
2b	—	VGG-6	CIFAR-10-Subset	25000	FedSMB	10	12500	0.0125	—	—
2b	—	ResNet-18	Imagenette-Subset	5000	FedSMB	10	2500	0.0125	—	—
3a & 4a	(ξ_s=0,λ_s=0)	VGG-11	CIFAR-100	50000	FedSMB	10	90000	0.05	0.99	500
3a & 4a	(ξ_s=0.9,λ_s=0)	VGG-11	CIFAR-100	50000	FedSMB	10	55000	0.05	0.99	500
3a & 4a	(ξ_s=0.9,λ_s=5e-4)	VGG-11	CIFAR-100	50000	FedSMB	10	90000	0.025	0.99	500
3b & 4b	(ξ_s=0.0,λ_s=0.0)	ResNet-18	ImageNet-50	25000	FedSMB	10	98000	0.05	0.5	18750
3b & 4b	(ξ_s=0.9,λ_s=0.0)	ResNet-18	ImageNet-50	25000	FedSMB	10	60000	0.025	0.5	18750
3b & 4b	(ξ_s=0.9,λ_s=5e-4)	ResNet-18	ImageNet-50	25000	FedSMB	10	98000	0.05	0.5	18750

(c) Ordinary federated learning

Table #	Config	Model	Dataset	N	Algorithm	τ	m	Iterations	η	ω	κ
3a & 4a	(ξ_s=0,λ_s=0)	VGG-11	CIFAR-100	50000	FedCMB	5	10	10000	0.025	0.99	100
3a & 4a	(ξ_s=0.9,λ_s=0)	VGG-11	CIFAR-100	50000	FedCMB	5	10	9000	0.01	0.99	100
3a & 4a	(ξ_s=0.9,λ_s=5e-4)	VGG-11	CIFAR-100	50000	FedCMB	5	10	10000	0.025	0.99	100
3a & 4a	(ξ_s=0,λ_s=0)	VGG-11	CIFAR-100	50000	FedAvg	5	1000	5000	0.025	0.99	50
3a & 4a	(ξ_s=0.9,λ_s=0)	VGG-11	CIFAR-100	50000	FedAvg	5	1000	4000	0.025	0.99	50
3a & 4a	(ξ_s=0.9,λ_s=5e-4)	VGG-11	CIFAR-100	50000	FedAvg	5	1000	5000	0.025	0.99	50
3a & 4a	(ξ_s=0,λ_s=0)	VGG-11	CIFAR-100	50000	FedCMB	100	10	2500	0.025	0.99	25
3a & 4a	(ξ_s=0.9,λ_s=0)	VGG-11	CIFAR-100	50000	FedCMB	100	10	2000	0.025	0.99	25
3a & 4a	(ξ_s=0.9,λ_s=5e-4)	VGG-11	CIFAR-100	50000	FedCMB	100	10	2500	0.025	0.99	25
3a & 4a	(ξ_s=0,λ_s=0)	VGG-11	CIFAR-100	50000	FedAvg	100	50	2500	0.025	0.99	25
3a & 4a	(ξ_s=0.9,λ_s=0)	VGG-11	CIFAR-100	50000	FedAvg	100	50	2000	0.025	0.99	25
3a & 4a	(ξ_s=0.9,λ_s=5e-4)	VGG-11	CIFAR-100	50000	FedAvg	100	50	2500	0.025	0.99	25
3b & 4b	(ξ_s=0.0,λ_s=0.0)	ResNet-18	ImageNet-50	25000	FedCMB	5	10	8000	0.025	0.5	1500
3b & 4b	(ξ_s=0.9,λ_s=0.0)	ResNet-18	ImageNet-50	25000	FedCMB	5	10	7000	0.025	0.5	1500
3b & 4b	(ξ_s=0.9,λ_s=5e-4)	ResNet-18	ImageNet-50	25000	FedCMB	5	10	8000	0.025	0.5	1500
3b & 4b	(ξ_s=0.0,λ_s=0.0)	ResNet-18	ImageNet-50	25000	FedAvg	5	500	4000	0.05	0.5	500
3b & 4b	(ξ_s=0.9,λ_s=0.0)	ResNet-18	ImageNet-50	25000	FedAvg	5	500	3500	0.05	0.5	500
3b & 4b	(ξ_s=0.9,λ_s=5e-4)	ResNet-18	ImageNet-50	25000	FedAvg	5	500	4000	0.05	0.5	500
3b & 4b	(ξ_s=0.0,λ_s=0.0)	ResNet-18	ImageNet-50	25000	FedCMB	100	10	4000	0.025	0.5	600
3b & 4b	(ξ_s=0.9,λ_s=0.0)	ResNet-18	ImageNet-50	25000	FedCMB	100	10	4000	0.025	0.5	600
3b & 4b	(ξ_s=0.9,λ_s=5e-4)	ResNet-18	ImageNet-50	25000	FedCMB	100	10	4000	0.025	0.5	600
3b & 4b	(ξ_s=0.0,λ_s=0.0)	ResNet-18	ImageNet-50	25000	FedAvg	100	25	3500	0.025	0.5	500
3b & 4b	(ξ_s=0.9,λ_s=0.0)	ResNet-18	ImageNet-50	25000	FedAvg	100	25	3500	0.025	0.5	500
3b & 4b	(ξ_s=0.9,λ_s=5e-4)	ResNet-18	ImageNet-50	25000	FedAvg	100	25	3500	0.025	0.5	500
5	(ξ_s=0.9,λ_s=5e-4)	VGG-11	CIFAR-100	50000	FedCMB+WGA	5	10	10000	0.01	0.99	100
5	(ξ_s=0.9,λ_s=5e-4)	ResNet-18	ImageNet-50	25000	FedCMB+WGA	5	10	8000	0.025	0.5	1500

Kernel Normalized Convolutional Networks

Reza Nasirigerdeh, Reihaneh Torkzadehmahani, Daniel Rueckert & Georgios Kaissis

Journal: Transactions on Machine Learning Research

Synopsis: Existing convolutional neural networks (CNNs) frequently rely upon batch normalization (*BatchNorm*) to be trained effectively. BatchNorm can significantly enhance the performance of CNNs by smoothening the optimization landscape, and alleviating the problem of vanishing gradients. BatchNorm, however, performs poorly with small batch sizes, and is inapplicable to differentially private learning. These limitations are due to the fact that BatchNorm breaks the independence among the samples in the batch, and consequently, it is not a batch-independent layer. To overcome the drawbacks of BatchNorm, batch-independent normalization layers such as layer normalization (*LayerNorm*) and group normalization (*GroupNorm*) have been introduced. These layers, however, do not typically achieve the performance of BatchNorm in centralized training, and their efficiency is not as expected in differentially private learning. To address these problems, we propose ***kernel normalization*** (*KernelNorm*) and ***kernel normalized convolutional*** (*KNConv*) layers, and incorporate them into kernel normalized convolutional networks (*KNConvNets*) as the main building blocks. We implement KNConvNets corresponding to the state-of-the-art ResNets, *KNResNets*, while forgoing BatchNorm layers. Through extensive experiments, we illustrate KNResNets provide higher or competitive performance compared to the BatchNorm counterparts in image classification and semantic segmentation. They also significantly outperform their batch-independent competitors including LayerNorm and GroupNorm in centralized and differentially private training. Given that, KernelNorm combines the batch-independence property of LayerNorm/GroupNorm with the performance advantage of BatchNorm.

Contributions of thesis author: Leading role in conceiving and designing the study, gathering software resources, developing algorithms, implementing the software components, conducting the experiments, writing the manuscript, and significant contribution in scientific findings.

Publisher: Transactions on Machine Learning Research (TMLR)

Date: 02.03.2024

Kernel Normalized Convolutional Networks

Reza Nasirigerdeh *reza.nasirigerdeh@tum.com*
Technical University of Munich
Helmholtz Munich

Reihaneh Torkzadehmahani *reihaneh.torkzadehmahani@tum.de*
Technical University of Munich

Daniel Rueckert *daniel.rueckert@tum.de*
Technical University of Munich
Imperial College London

Georgios Kaissis *g.kaissis@tum.de*
Technical University of Munich
Helmholtz Munich

Reviewed on OpenReview: *https://openreview.net/forum?id=Uv3XVAEgG6*

Abstract

Existing convolutional neural network architectures frequently rely upon batch normalization (BatchNorm) to effectively train the model. BatchNorm, however, performs poorly with small batch sizes, and is inapplicable to differential privacy. To address these limitations, we propose the **kernel normalization** (**KernelNorm**) and **kernel normalized convolutional** layers, and incorporate them into kernel normalized convolutional networks (**KNConvNets**) as the main building blocks. We implement KNConvNets corresponding to the state-of-the-art ResNets while forgoing the BatchNorm layers. Through extensive experiments, we illustrate that KNConvNets achieve higher or competitive performance compared to the BatchNorm counterparts in image classification and semantic segmentation. They also significantly outperform their batch-independent competitors including those based on layer and group normalization in non-private and differentially private training. Given that, KernelNorm combines the batch-independence property of layer and group normalization with the performance advantage of BatchNorm [1].

1 Introduction

Convolutional neural networks (CNNs) (LeCun et al., 1989) are standard architectures in computer vision tasks such as image classification (Krizhevsky et al., 2012; Sermanet et al., 2014) and semantic segmentation (Long et al., 2015b). Deep CNNs including ResNets (He et al., 2016a) achieved outstanding performance in classification of challenging datasets such as ImageNet (Deng et al., 2009). One of the main building blocks of these CNNs is *batch normalization* (BatchNorm) (Ioffe & Szegedy, 2015). The BatchNorm layer considerably enhances the performance of deep CNNs by smoothening the optimization landscape (Santurkar et al., 2018), and addressing the problem of vanishing gradients (Bengio et al., 1994; Glorot & Bengio, 2010).

BatchNorm, however, has the disadvantage of breaking the independence among the samples in the batch (Brock et al., 2021b). This is because BatchNorm carries out normalization along the batch dimension (Figure 1a), and as a result, the normalized value associated with a given sample depends on the statistics of the other samples in the batch. Consequently, the effectiveness of BatchNorm is highly dependent on

[1] The code is available at: `https://github.com/reza-nasirigerdeh/norm-torch`

batch size. With large batch sizes, the batch normalized models are trained effectively due to more accurate estimation of the batch statistics. Using small batch sizes, on the other hand, BatchNorm causes reduction in model accuracy (Wu & He, 2018) because of dramatic fluctuations in the batch statistics. BatchNorm, moreover, is inapplicable to *differential privacy* (DP) (Dwork & Roth, 2014). For the theoretical guarantees of DP to hold for the training of neural networks (Abadi et al., 2016), it is required to compute the gradients individually for each sample in a batch, clip the per-sample gradients, and then average and inject random noise to limit the information learnt about any particular sample. Because per-sample (individual) gradients are required, the gradients of a given sample are not allowed to be influenced by other samples in the batch. This is not the case for BatchNorm, where samples are normalized using the statistics computed over the other samples in the batch. Consequently, BatchNorm is inherently incompatible with DP.

To overcome the limitations of BatchNorm, the community has introduced *batch-independent* normalization layers including layer normalization (LayerNorm) (Ba et al., 2016), instance normalization (InstanceNorm) (Ulyanov et al., 2016), group normalization (GroupNorm) (Wu & He, 2018), positional normalization (PositionalNorm) (Li et al., 2019), and local context normalization (LocalContextNorm) (Ortiz et al., 2020), which perform normalization independently for each sample in the batch. These layers do not suffer from the drawbacks of BatchNorm, and might outperform BatchNorm in particular domains such as generative tasks (e.g. LayerNorm in Transformer models (Vaswani et al., 2017)). For image classification and semantic segmentation, however, they typically do not achieve performance comparable with BatchNorm's in non-private (without DP) training. In DP, moreover, these batch-independent layers might not provide the accuracy gain we expect compared to non-private learning. This motivates us to develop alternative layers, which are batch-independent but more efficient in both non-private and differentially private learning.

Our main contribution is to propose two novel *batch-independent* layers called **kernel normalization** (**KernelNorm**) and the **kernel normalized convolutional** (**KNConv**) layer to further enhance the performance of deep CNNs. The distinguishing characteristic of the proposed layers is that they *extensively* take into account the *spatial correlation* among the elements during normalization. KernelNorm is similar to a pooling layer, except that it normalizes the elements specified by the kernel window instead of computing the average/maximum of the elements, and it operates over all input channels instead of a single channel (Figure 1g). KNConv is the combination of KernelNorm with a convolutional layer, where it applies KernelNorm to the input, and feeds KernelNorm's output to the convolutional layer (Figure 2). From another perspective, KNConv is the same as the convolutional layer except that KNConv first normalizes the input elements specified by the kernel window, and then computes the convolution between the normalized elements and kernel weights. In both aforementioned naive forms, however, KNConv is computationally inefficient because it leads to extremely large number of normalization units, and therefore, considerable computational overhead to normalize the corresponding elements. To tackle this issue, we present **computationally-efficient KNConv** (Algorithm 1), where the output of the convolution is adjusted using the mean and variance of the normalization units. This way, it is not required to normalize the elements, improving the computation time by orders of magnitude.

As an application of the proposed layers, we introduce **kernel normalized convolutional networks** (**KNConvNets**) corresponding to residual networks (He et al., 2016a), referred to as **KNResNets**, which employ KernelNorm and computationally-efficient KNConv as the main building blocks while forgoing the BatchNorm layers (Section 3). Our last contribution is to draw performance comparisons among KNResNets and the competitors using several benchmark datasets including CIFAR-100 (Krizhevsky et al., 2009), ImageNet (Deng et al., 2009), and Cityscapes (Cordts et al., 2016). According to the experimental results (Section 4), KNResNets deliver significantly higher accuracy than the BatchNorm counterparts in image classification on CIFAR-100 using a small batch size. KNResNets, moreover, achieve higher or competitive performance compared to the batch normalized ResNets in classification on ImageNet and semantic segmentation on CityScapes. Furthermore, KNResNets considerably outperform GroupNorm and LayerNorm based models for almost all considered case studies in non-private and differentially private learning. Considering that, KernelNorm combines the performance advantage of BatchNorm with the batch-independence benefit of LayerNorm and GroupNorm.

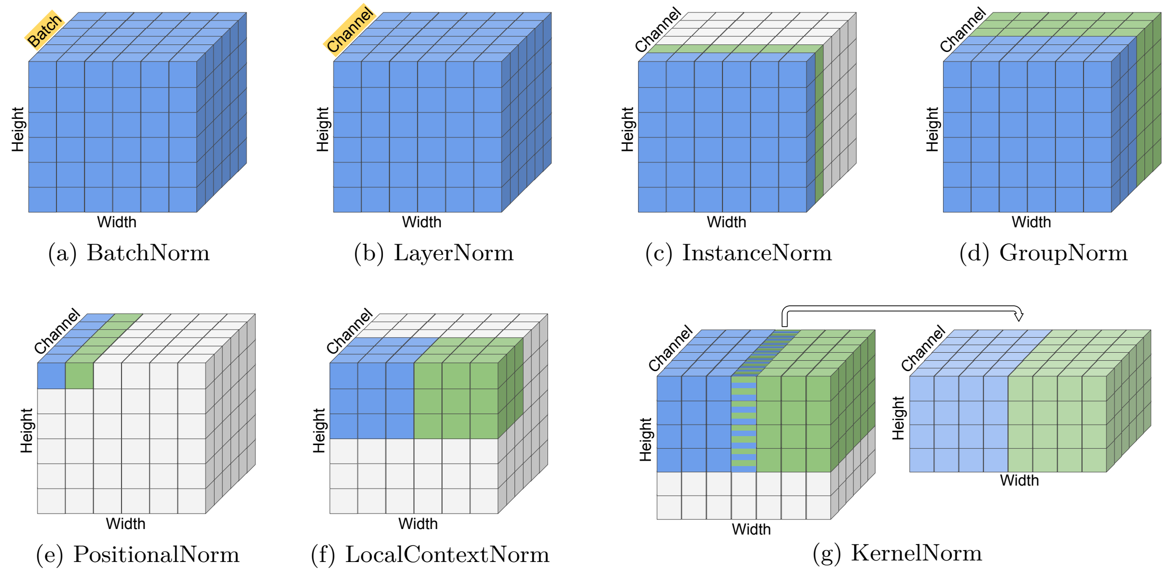


Figure 1: **Normalization layers** differ from one another in their normalization unit (highlighted in blue and green). The normalization layers in (a)-(f) establish a *one-to-one correspondence* between the input and normalized elements (i.e. no overlap between the normalization units, and no ignorance of an element). The proposed **KernelNorm** layer does not impose such one-to-one correspondence: Some elements (dash-hatched area) are common among the normalization units, contributing more than once to the output, while some elements (uncolored ones) are ignored during normalization. Due to this unique property of overlapping normalization units, KernelNorm *extensively* incorporates the spatial correlation among the elements during normalization (akin to the convolutional layer), which is not the case for the other normalization layers.

2 Normalization Layers

Normalization methods can be categorized into *input normalization* and *weight normalization* (Salimans & Kingma, 2016; Bansal et al., 2018; Wang et al., 2020; Qi et al., 2020). The former techniques perform normalization on the input tensor, while the latter ones normalize the model weights. The aforementioned layers including BatchNorm, and the proposed KernelNorm layer as well as *divisive normalization* (Heeger, 1992; Bonds, 1989), (Ren et al., 2017) and *local response normalization* (LocalResponseNorm) (Krizhevsky et al., 2012) belong to the category of input normalization. Weight standardization (Huang et al., 2017b; Qiao et al., 2019) and normalizer-free networks (Brock et al., 2021a) fall into the category of weight normalization.

In the following, we provide an overview on the existing normalization layers closely related to KernelNorm, i.e. the layers which are based on input normalization, and employ standard normalization (zero-mean and unit-variance) to normalize the input tensor. For the sake of simplicity, we focus on 2D images, but the concepts are also applicable to 3D images. For a 2D image, the input of a layer is a 4D tensor of shape (n, c, h, w), where n is batch size, c is the number of input channels, h is height, and w is width of the tensor. Normalization layers differ from one another in their *normalization unit*, which is a group of input elements that are normalized together with the mean and variance of the unit.

The normalization unit of **BatchNorm** (Figure 1a) is a 3D tensor of shape (n, h, w), implying that BatchNorm incorporates all elements in the batch, height, and width dimensions during normalization. **LayerNorm**'s normalization unit (Figure 1b) is a 3D tensor of shape (c, h, w), i.e. LayerNorm considers all elements in the channel, height, and width dimensions for normalization. The normalization unit of **InstanceNorm** (Figure 1c) is a 2D tensor of shape (h, w), i.e. all elements of the height and width dimensions are taken into account during normalization.

GroupNorm's normalization unit (Figure 1d) is a 3D tensor of shape (c_g, h, w), where c_g indicates the channel group size. Thus, GroupNorm incorporates all elements in the height and width dimensions and a subset of elements specified by the group size in the channel dimension during normalization. **PositionalNorm**'s normalization unit (Figure 1e) is a 1D tensor of shape c, i.e. PositionalNorm performs channel-wise normalization. The normalization unit of **LocalContextNorm** (Figure 1f) is a 3D tensor of shape (c_g, r, s), where c_g is the group size, and (r, s) is the window size. Therefore, LocalContextNorm considers a subset of elements in the height, width, and channel dimensions during normalization.

BatchNorm, LayerNorm, InstanceNorm, and GroupNorm consider *all elements* in the height and width dimensions for normalization, and thus, they are referred to as *global normalization* layers. PositionalNorm and LocalContextNorm, on the other hand, are called *local normalization* layers (Ortiz et al., 2020) because they incorporate a *subset of elements* from the aforementioned dimensions during normalization. In spite of their differences, the aforementioned normalization layers including BatchNorm have at least one thing in common: There is a *one-to-one correspondence* between the original elements in the input and the normalized elements in the output. That is, there is exactly one normalized element associated with each input element. Therefore, these layers do not modify the shape of the input during normalization.

3 Kernel Normalized Convolutional Networks

The KernelNorm and KNConv layers are the main building blocks of KNConvNets. **KernelNorm** takes the kernel size (k_h, k_w), stride (s_h, s_w), padding (p_h, p_w), and dropout probability p as hyper-parameters. It pads the input with zeros if padding is specified. The normalization unit of KernelNorm (Figure 1g) is a tensor of shape (c, k_h, k_w), i.e. KernelNorm incorporates *all elements* in the channel dimension but a *subset of elements* specified by the kernel size from the height and width dimensions during normalization. The KernelNorm layer (1) applies random dropout (Srivastava et al., 2014) to the normalization unit to obtain the *dropped-out* unit, (2) computes mean and variance of the dropped-out unit, and (3) employs the calculated mean and variance to normalize the *original* normalization unit:

$$U' = D_p(U), \tag{1}$$

$$\begin{aligned} \mu_{u'} &= \frac{1}{c \cdot k_h \cdot k_w} \cdot \sum_{i_c=1}^{c} \sum_{i_h=1}^{k_h} \sum_{i_w=1}^{k_w} U'(i_c, i_h, i_w), \\ \sigma_{u'}^2 &= \frac{1}{c \cdot k_h \cdot k_w} \cdot \sum_{i_c=1}^{c} \sum_{i_h=1}^{k_h} \sum_{i_w=1}^{k_w} (U'(i_c, i_h, i_w) - \mu_{u'})^2, \end{aligned} \tag{2}$$

$$\hat{U} = \frac{U - \mu_{u'}}{\sqrt{\sigma_{u'}^2 + \epsilon}}, \tag{3}$$

where p is the dropout probability, D_p is the dropout operation, U is the normalization unit, U' is the dropped-out unit, $\mu_{u'}$ and $\sigma_{u'}^2$ are the mean and variance of the dropped-out unit, respectively, ϵ is a small number (e.g. 10^{-5}) for numerical stability, and $\hat{U}$ is the normalized unit.

Partially inspired by BatchNorm, KernelNorm introduces a *regularizing effect* during training by intentionally normalizing the elements of the original unit U using the statistics computed over the dropped-out unit U'. In BatchNorm, the normalization statistics are computed over the batch but not the whole dataset, where the mean and variance of the batch are randomized approximations of those from the whole dataset. The "stochasticity from the batch statistics" creates a regularizing effect in BatchNorm according to Ba et al. (2016). KernelNorm employs dropout to generate similar stochasticity in the mean and variance of the normalization unit. Notice that the naive option of injecting random noise directly into the mean and variance might generate too much randomness, and hinder model convergence. Using dropout in the aforementioned fashion, KernelNorm can control the regularization effect with more flexibility.

The first normalization unit of KernelNorm is bounded to a window specified by diagonal points (1, 1) and (k_h, k_w) in the height and width dimensions. The coordinates of the next normalization unit are $(1,\, 1+s_w)$ and $(k_h,\, k_w + s_w)$, which are obtained by sliding the window s_w elements along the width dimension. If there are not enough elements for kernel in the width dimension, the window is slid by s_h elements in the height dimension, and the above procedure is repeated. Notice that KernelNorm works on the padded input of shape $(n,\, c,\, h + 2 \cdot p_h,\, w + 2 \cdot p_w)$, where $(p_h,\, p_w)$ is the padding size. The output $\hat{X}$ of KernelNorm is the concatenation of the *normalized units* $\hat{U}$ from Equation 3 along the height and width dimensions. KernelNorm's output is of shape (n, c, h_{out}, w_{out}), and it has total of $n \cdot \frac{h_{out}}{k_h} \cdot \frac{w_{out}}{k_w}$ normalization units, where h_{out} and w_{out} are computed as follows:

$$h_{out} = k_h \cdot \lfloor \frac{h + 2 \cdot p_h - k_h}{s_h} + 1 \rfloor, w_{out} = k_w \cdot \lfloor \frac{w + 2 \cdot p_w - k_w}{s_w} + 1 \rfloor$$

In simple terms, KernelNorm behaves similarly to the pooling layers with two major differences: (1) KernelNorm normalizes the elements specified by the kernel size instead of computing the maximum/average over the elements, and (2) KernelNorm operates over all channels rather than a single channel. KernelNorm is a *batch-independent* and *local normalization* layer, but differs from the existing normalization layers in two aspects: (I) There is not necessarily a one-to-one correspondence between the original elements in the input and the normalized elements in the output of KernelNorm. Stride values less than kernel size lead to overlapping normalization units, where some input elements contribute more than once in the output (akin to the convolutional layer). If the stride value is greater than kernel size, some input elements are completely ignored during normalization. Therefore, the output shape of KernelNorm can be different from the input shape. (II) KernelNorm can *extensively* take into account the *spatial correlation* among the elements during normalization because of the overlapping normalization units.

KNConv is the combination of KernelNorm and the traditional convolutional layer (Figure 2). It takes the number of input channels ch_{in}, number of output channels (filters) ch_{out}, kernel size (k_h, k_w), stride (s_h, s_w), and padding (p_h, p_w), exactly the same as the convolutional layer, as well as the dropout probability p as hyper-parameters. KNConv first applies KernelNorm with kernel size (k_h, k_w), stride (s_h, s_w), padding (p_h, p_w), and dropout probability p to the input tensor. Next, it applies the convolutional layer with ch_{in} channels, ch_{out} filters, kernel size (k_h, k_w), stride (k_h, k_w), and padding of zero to the output of KernelNorm. That is, both kernel size and stride values of the convolutional layer are identical to kernel size of KernelNorm.

From another perspective, KNConv is the same as the convolutional layer except that it normalizes the input elements specified by the kernel window before computing the convolution. Assuming that U contains the input elements specified by the kernel window, $\hat{U}$ is the normalized version of U from KernelNorm (Equation 3), Z is the kernel weights of a given filter, $\star$ is the convolution (or dot product) operation, and b is the bias value, KNConv computes the output as follows:

$$\text{KNConv}(U, Z, b) = \hat{U} \star Z + b \tag{4}$$

KNConv (or in fact KernelNorm) leads to extremely high number of normalization units, and consequently, remarkable computational overhead. Thus, KNConv in its simple format outlined in Equation 4 (or as a combination of the KernelNorm and convolutional layers) is computationally inefficient. Compared to the convolutional layer, the additional computational overhead of KNConv originates from (I) calculating the mean and variance of the units using Equation 2, and (II) normalizing the elements by the mean and variance using Equation 3.

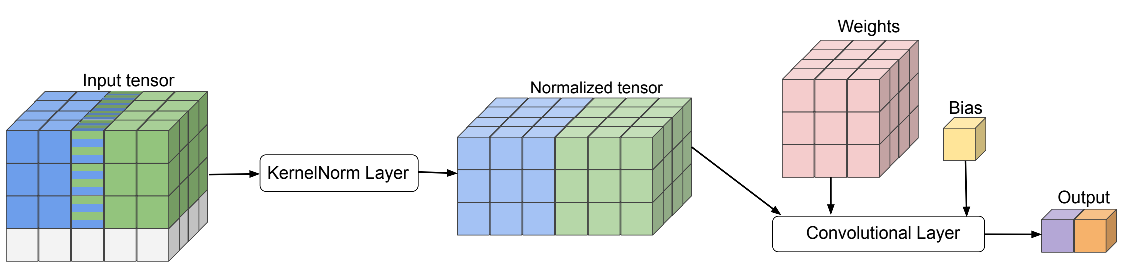


Figure 2: **KNConv** as the combination of the KernelNorm and convolutional layers. KNConv first applies KernelNorm with kernel size (3, 3) and stride (2,2) to the input tensor, and then gives KernelNorm's output to a convolutional layer with kernel size and stride (3, 3). That is, the kernel size and stride of the convolutional layer and the kernel size of KernelNorm are identical.

Computationally-efficient KNConv reformulates Equation 4 in a way that it completely eliminates the overhead of normalizing the elements:

$$
\begin{aligned}
\text{KNConv}(U, Z, b) &= \hat{U} \star Z + b = \sum_{i_c=1}^{c} \sum_{i_h=1}^{k_h} \sum_{i_w=1}^{k_w} \left(\frac{U(i_c, i_h, i_w) - \mu_{u'}}{\sqrt{\sigma_{u'}^2 + \epsilon}}\right) \cdot Z(i_c, i_h, i_w) + b \\
&= \left(\sum_{i_c=1}^{c} \sum_{i_h=1}^{k_h} \sum_{i_w=1}^{k_w} U(i_c, i_h, i_w) \cdot Z(i_c, i_h, i_w) - \mu_{u'} \cdot \sum_{i_c=1}^{c} \sum_{i_h=1}^{k_h} \sum_{i_w=1}^{k_w} Z(i_c, i_h, i_w)\right) \cdot \frac{1}{\sqrt{\sigma_{u'}^2 + \epsilon}} + b \\
&= \left(U \star Z - \mu_{u'} \cdot \sum_{i_c=1}^{c} \sum_{i_h=1}^{k_h} \sum_{i_w=1}^{k_w} Z(i_c, i_h, i_w)\right) \cdot \frac{1}{\sqrt{\sigma_{u'}^2 + \epsilon}} + b
\end{aligned} \tag{5}
$$

According to Equation 5 and Algorithm 1, KNConv applies the convolutional layer to the original unit, computes the mean and standard deviation of the dropped-out unit as well as the sum of the kernel weights, and finally adjusts the convolution output using the computed statistics. This way, it is not required to normalize the elements, improving the computation time of KNConv by orders of magnitude.

In terms of implementation, KernelNorm employs the *unfolding* operation in PyTorch (2023b) to implement the sliding window mechanism in the *kn_mean_var* function in Algorithm 1. Moreover, it uses the *var_mean* function in PyTorch (2023c) to compute the mean and variance over the unfolded tensor along the channel, width, and height dimensions.

The defining characteristic of KernelNorm and KNConv is that they take into consideration the *spatial correlation* among the elements during normalization on condition that the kernel size is greater than 1×1. Existing architectures (initially designed for global normalization), however, do not satisfy this condition. For instance, all ResNets use 1×1 convolution for downsampling and increasing the number of filters. ResNet-50/101/152, in particular, contains bottleneck blocks with a single 3×3 and two 1×1 convolutional layers. Consequently, the current architectures are unable to fully utilize the potential of kernel normalization.

KNConvNets are bespoke architectures for kernel normalization, consisting of computationally-efficient KNConv and KernelNorm as the main building blocks. KNConvNets are *batch-independent* (free of BatchNorm), which primarily employ kernel sizes of 2×2 or 3×3 to benefit from the spatial correlation of elements during normalization. In this study, we propose KNConvNets corresponding to ResNets, called *KNResNets*, for image classification and semantic segmentation.

Algorithm 1: Computationally-efficient KNConv layer

Input: input tensor X, number of input channels ch_{in}, number of output channels ch_{out}, kernel size (k_h, k_w), stride (s_h, s_w), padding (p_h, p_w), *bias* flag, dropout probability p, and epsilon ϵ

`// 2-dimensional convolutional layer`

conv_layer = Conv2d(in_channels=ch_{in}, out_channels=ch_{out}, kernel_size=(k_h, k_w), stride=(s_h, s_w), padding=(p_h, p_w), bias=$false$)

`// convolutional layer output`

conv_out = conv_layer(input=X)

`// mean and variance from KernelNorm`

μ, σ^2 = kn_mean_var(input=X, kernel_size=(k_h, k_w), stride=(s_h, s_w), padding=(p_h, p_w), dropout_p=p)

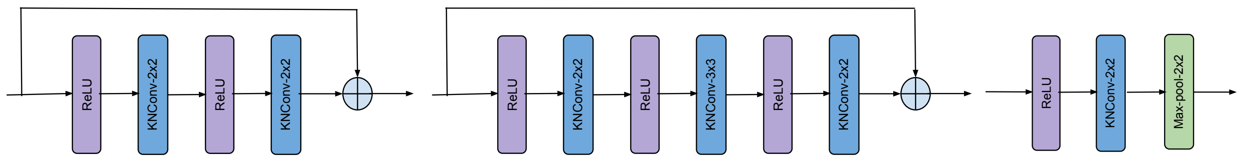
`// KNConv output`

kn_conv_out = (conv_out - $\mu \cdot \sum$ conv_layer.weights) / $\sqrt{\sigma^2 + \epsilon}$

`// apply bias`

if bias **then**
  kn_conv_out += conv_layer.bias

Output: kn_conv_out



(a) Basic block (b) Bottleneck block (c) Transitional block

Figure 3: **KNResNet blocks**: Basic blocks are employed in KNResNet-18/34, while KNResNet-50 is based on bottleneck blocks. Transitional blocks are used in all KNResNets for increasing the number of filters and downsampling. The architectures of KNResNet-18/34/50 are available in Figures 5-6 in Appendix A.

KNResNets comprise three types of blocks: residual *basic* block, residual *bottleneck* block, and *transitional* block (Figure 3). Basic blocks contain two KNConv layers with kernel size of 2×2, whereas bottleneck blocks consist of three KNConv layers with kernel sizes of 2×2, 3×3, and 2×2, respectively. The stride value in both basic and bottleneck blocks is 1×1. The padding values of the first and last KNConv layers, however, are 1×1 and zero so that the width and height of the output remain identical to the input's (necessary condition for residual blocks with identity shortcut). The middle KNConv layer in bottleneck blocks uses 1×1 padding. Transitional blocks include a KNConv layer with kernel size of 2×2 and stride of 1×1 to increase the number of filters, and a max-pooling layer with kernel size and stride of 2×2 to downsample the input.

We propose the KNResNet-18, KNResNet-34, and KNResNet-50 architectures based on the aforementioned block types (Figure 5 in Appendix A). KNResNet-18/34 uses basic and transitional blocks, while KNResNet-50 mainly employs bottleneck and transitional blocks. For semantic segmentation, we utilize KNResNet-18/34/50 as backbone (Figure 6 in Appendix A), but the kernel size of the KNConv and max-pooling layers in basic and transitional blocks is 3×3 instead of 2×2.

4 Evaluation

We compare the performance of KNResNets to the BatchNorm, GroupNorm, LayerNorm, and LocalContextNorm counterparts. For image classification, we do not include LocalContextNorm in our evaluation because its performance is similar to GroupNorm (Ortiz et al., 2020). The experimental evaluation is divided into four categories: (I) batch size-dependent performance analysis, (II) image classification on ImageNet, (III) semantic segmentation on Cityscapes, and (IV) differentially private image classification on ImageNet32×32.

We adopt the original implementation of ResNet-18/34/50 from PyTorch (Paszke et al., 2019), and the PreactResNet-18/34/50 (He et al., 2016b) implementation from Kuang (2021). The architectures are based on BatchNorm. For GroupNorm/LocalContextNorm related models, BatchNorm is replaced by GroupNorm/LocalContextNorm. Regarding LayerNorm based architectures, GroupNorm with number of groups of 1 (equivalent to LayerNorm) is substituted for BatchNorm. The number of groups of GroupNorm is 32 (Wu & He, 2018). The number of groups and window size for LocalContextNorm are 2 and 227×227, respectively (Ortiz et al., 2020).

For low-resolution datasets (CIFAR-100 and ImageNet32×32), we replace the first 7×7 convolutional layer with a 3×3 convolutional layer and remove the following max-pooling layer. Moreover, we insert a normalization layer followed by an activation function before the last average-pooling layer in the PreactResNet architectures akin to KNResNets (Figure 5 at Appendix A). The aforementioned modifications considerably enhance the accuracy of the competitors. For semantic segmentation, we employ the fully convolutional network architecture (Long et al., 2015a) with BatchNorm, GroupNorm, LayerNorm, and LocalContextNorm based ResNet-18/34/50 as backbone. For KNResNets, we use fully convolutional versions of KNResNet-18/34/50 (Figure 6 at Appendix A).

Table 1: Test accuracy versus batch size on **CIFAR-100**.

Model	Normalization	Parameters	B=2	B=32	B=256
ResNet-18-LN	LayerNorm	11.220 M	72.68±0.22	73.17±0.16	71.99±0.45
PreactResNet-18-LN	LayerNorm	11.220 M	73.51±0.10	73.36±0.15	72.91±0.07
ResNet-18-GN	GroupNorm	11.220 M	74.62±0.12	74.46±0.05	74.46±0.08
PreactResNet-18-GN	GroupNorm	11.220 M	74.82±0.24	74.74±0.44	74.62±0.36
ResNet-18-BN	BatchNorm	11.220 M	72.11±0.25	78.52±0.20	77.72±0.04
PreactResNet-18-BN	BatchNorm	11.220 M	72.57±0.19	78.32±0.09	77.83±0.16
KNResNet-18 (ours)	KernelNorm	11.216 M	**79.10**±0.10	**79.29**±0.02	**78.84**±0.10
ResNet-34-LN	LayerNorm	21.328 M	73.74±0.26	73.88±0.37	72.48±0.57
PreactResNet-34-LN	LayerNorm	21.328 M	74.79±0.13	74.34±0.42	73.10±0.42
ResNet-34-GN	GroupNorm	21.328 M	75.76±0.14	75.72±0.06	75.44±0.27
PreactResNet-34-GN	GroupNorm	21.328 M	75.82±0.05	75.85±0.28	75.76±0.25
ResNet-34-BN	BatchNorm	21.328 M	73.06±0.23	79.21±0.09	78.27±0.19
PreactResNet-34-BN	BatchNorm	21.328 M	72.20±0.19	79.09±0.03	78.59±0.24
KNResNet-34 (ours)	KernelNorm	21.323 M	**79.28**±0.09	**79.53**±0.15	**79.16**±0.21
ResNet-50-LN	LayerNorm	23.705 M	75.83±0.25	75.74±0.14	74.37±0.58
PreactResNet-50-LN	LayerNorm	23.705 M	74.28±0.31	74.57±0.32	73.41±0.15
ResNet-50-GN	GroupNorm	23.705 M	77.03±0.62	77.02±0.08	74.79±0.14
PreactResNet-50-GN	GroupNorm	23.705 M	75.67±0.27	76.08±0.18	75.52±0.13
ResNet-50-BN	BatchNorm	23.705 M	71.02±0.15	**80.39**±0.06	77.89±0.06
PreactResNet-50-BN	BatchNorm	23.705 M	70.83±0.41	80.28±0.15	78.88±0.21
KNResNet-50 (ours)	KernelNorm	23.682 M	**80.24**±0.18	80.18±0.10	**80.09**±0.26

4.1 Batch size-dependent performance analysis

Dataset. The CIFAR-100 dataset consists of 50000 train and 10000 test samples of shape 32×32 from 100 classes. We adopt the data preprocessing and augmentation scheme widely used for the dataset (Huang et al., 2017a; He et al., 2016b;a): Horizontally flipping and randomly cropping the samples after padding them. The cropping and padding sizes are 32×32 and 4×4, respectively. Additionally, the feature values are divided by 255 for KNResNets, whereas they are normalized using the mean and standard deviation (SD) of the dataset for the competitors.

Training. The models are trained for 150 epochs using the cosine annealing scheduler (Loshchilov & Hutter, 2017) with learning rate decay of 0.01. The optimizer is SGD with momentum of 0.9 and weight decay of 0.0005. For learning rate tuning, we run a given experiment with initial learning rate of 0.2, divide it by 2, and re-run the experiment. We continue this procedure until finding the best learning rate (Table 5 in Appendix B). Then, we repeat the experiment three times, and report the mean and SD over the runs.

Results. Table 1 lists the test accuracy values achieved by the models for different batch sizes. According to the table, (I) KNResNets dramatically outperform the BatchNorm counterparts for batch size of 2, (II) KNResNets deliver highly competitive accuracy values compared to BatchNorm-based models with batch sizes of 32 and 256, and (III) KNResNets achieve significantly higher accuracy than the batch-independent competitors (LayerNorm and GroupNorm) for all considered batch sizes.

4.2 Image classification on ImageNet

Dataset. The ImageNet dataset contains around 1.28 million training and 50000 validation images. Following the data preprocessing and augmentation scheme from TorchVision (2023a), the train images are horizontally flipped and randomly cropped to 224×224. The test images are first resized to 256×256, and then center cropped to 224×224. The feature values are normalized using the mean and SD of ImageNet.

Training. We follow the experimental setting from Wu & He (2018) and use the multi-GPU training script from TorchVision (2023a) to train KNResNets and the competitors. We train all models for 100 epochs with total batch size of 256 (8 GPUs with batch size of 32 per GPU) using learning rate of 0.1, which is divided by 10 at epochs 30, 60, and 90. The optimizer is SGD with momentum of 0.9 and weight decay of 0.0001.

Table 2: Image classification on **ImageNet**.

Model	Normalization	Parameters	Top-1 accuracy
ResNet-18-LN	LayerNorm	11.690 M	68.34
ResNet-18-GN	GroupNorm	11.690 M	68.93
ResNet-18-BN	BatchNorm	11.690 M	70.28
KNResNet-18 (ours)	KernelNorm	11.685 M	**71.17**
ResNet-34-LN	LayerNorm	21.798 M	71.64
ResNet-34-GN	GroupNorm	21.798 M	72.63
ResNet-34-BN	BatchNorm	21.798 M	73.99
KNResNet-34 (ours)	KernelNorm	21.793 M	**74.60**
ResNet-50-LN	LayerNorm	25.557 M	73.80
ResNet-50-GN	GroupNorm	25.557 M	75.92
ResNet-50-BN	BatchNorm	25.557 M	**76.41**
KNResNet-50 (ours)	KernelNorm	25.556 M	**76.54**

Results. Table 2 demonstrates the Top-1 accuracy values on ImageNet for different architectures. As shown in the table, (I) KNResNet-18 and KNResNet-34 outperform the BatchNorm counterparts by around 0.9% and 0.6%, respectively, (II) KNResNet-18/34/50 achieves higher accuracy (by about 0.6%-3.0%) than LayerNorm and GroupNorm based competitors, and (III) KNResNet-50 delivers almost the same accuracy as the batch normalized ResNet-50.

4.3 Semantic segmentation on CityScapes

Dataset. The CityScapes dataset contains 2975 train and 500 validation images from 30 classes, 19 of which are employed for evaluation. Following Sun et al. (2019); Ortiz et al. (2020), the train samples are randomly cropped from 2048×1024 to 1024×512, horizontally flipped, and randomly scaled in the range of [0.5, 2.0]. The models are tested on the validation images, which are of shape 2048×1024.

Training. Following Sun et al. (2019); Ortiz et al. (2020), we train the models with learning rate of 0.01, which is gradually decayed by power of 0.9. The models are trained for 500 epochs using 2 GPUs with batch size of 8 per GPU. The optimizer is SGD with momentum of 0.9 and weight decay of 0.0005. Notice that we use SyncBatchNorm instead of BatchNorm in the batch normalized models.

Table 3: Semantic segmentation on **CityScapes**.

Model	Normalization	Parameters	mIoU	Pixel accuracy	Mean accuracy
ResNet-18-LN	LayerNorm	13.547 M	59.10±0.46	92.42±0.17	69.43±0.58
ResNet-18-GN	GroupNorm	13.547 M	62.33±0.52	93.23±0.01	71.58±0.55
ResNet-18-LCN	LocalContextNorm	13.547 M	62.25±0.67	92.99±0.06	71.59±0.68
ResNet-18-BN	BatchNorm	13.547 M	63.90±0.06	**93.77**±0.02	73.15±0.14
KNResNet-18 (ours)	KernelNorm	13.525 M	**64.37**±0.14	**93.73**±0.01	**73.46**±0.12
ResNet-34-LN	LayerNorm	23.655 M	60.19±0.32	92.73±0.17	70.12±0.33
ResNet-34-GN	GroupNorm	23.655 M	64.21±0.58	93.59±0.07	74.32±0.49
ResNet-34-LCN	LocalContextNorm	23.655 M	64.75±0.38	93.31±0.09	74.25±0.37
ResNet-34-BN	BatchNorm	23.655 M	66.94±0.34	**94.27**±0.03	**76.50**±0.41
KNResNet-34 (ours)	KernelNorm	23.399 M	**67.61**±0.17	**94.13**±0.05	**76.58**±0.19
ResNet-50-LN	LayerNorm	32.955 M	57.88±0.84	92.31±0.21	68.25±0.75
ResNet-50-GN	GroupNorm	32.955 M	62.14±0.68	93.34±0.04	71.66±0.64
ResNet-50-LCN	LocalContextNorm	32.955 M	64.03±0.02	93.07±0.14	73.40±0.03
ResNet-50-BN	BatchNorm	32.955 M	65.19±0.50	93.98±0.03	74.65±0.62
KNResNet-50 (ours)	KernelNorm	32.874 M	**68.02**±0.13	**94.22**±0.04	**77.03**±0.05

Results. Table 3 lists the mean of class-wise intersection over union (mIoU), pixel accuracy, and mean of class-wise pixel accuracy for different architectures. According to the table, (I) KNResNet-18/34 and the BatchNorm-based counterparts achieve highly competitive mIoU, pixel accuracy, and mean accuracy, whereas KNResNet-50 delivers considerably higher mIoU and mean accuracy than batch normalized ResNet-50, (II) KNResNets significantly outperform the batch-independent competitors (the LayerNorm, GroupNorm, and LocalContextNorm based models) in terms of all considered performance metrics. Surprisingly, ResNet-50 based models perform worse than ResNet-34 counterparts for the competitors possibly because of the smaller kernel size they employ in ResNet-50 compared to ResNet-34 (1×1 instead of 3×3).

4.4 Differentially private image classification on ImageNet32×32

Dataset. ImageNet32×32 is the down-sampled version of ImageNet, where all images are resized to 32×32. For preprocessing, the feature values are divided by 255 for KNResNet-18, while they are normalized by the mean and SD of ImageNet for the layer and group normalized ResNet-18.

Training. We train KNResNet-18 as well as the GroupNorm and LayerNorm counterparts for 100 epochs using the SGD optimizer with zero-momentum and zero-weight decay, where the learning rate is decayed by factor of 2 at epochs 70, and 90. Note that BatchNorm is inapplicable to differential privacy. All models use the Mish activation (Misra, 2019). For parameter tuning, we consider learning rate values of {2.0, 3.0, 4.0}, clipping values of {1.0, 2.0}, and batch sizes of {2048, 4096, 8192}. We observe that learning rate of 4.0, clipping value of 2.0, and batch size of 8192 achieve the best performance for all models. Our differentially private training is based on DP-SGD (Abadi et al., 2016) from the Opacus library (Yousefpour et al., 2021) with ε=8.0 and δ=8×10^{-7}. The privacy accountant is RDP (Mironov, 2017)

Table 4: Differentially private image classification on **ImageNet32×32**.

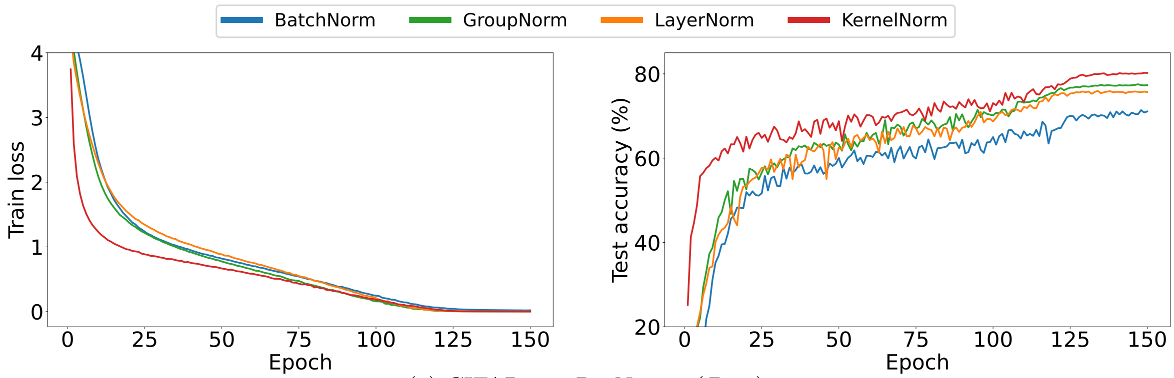
Model	Normalization	Parameters	Top-1 accuracy
ResNet-18-BN	BatchNorm	11.682 M	NA
ResNet-18-LN	LayerNorm	11.682 M	20.81
ResNet-18-GN	GroupNorm	11.682 M	20.99
KNResNet-18 (ours)	KernelNorm	11.678 M	**22.01**

Results. Table 4 lists the Top-1 accuracy values on ImageNet32×32 for different models trained in the aforementioned differentially private learning setting. As can be seen in the table, KNResNet-18 achieves significantly higher accuracy than the layer and group normalized ResNet-18.

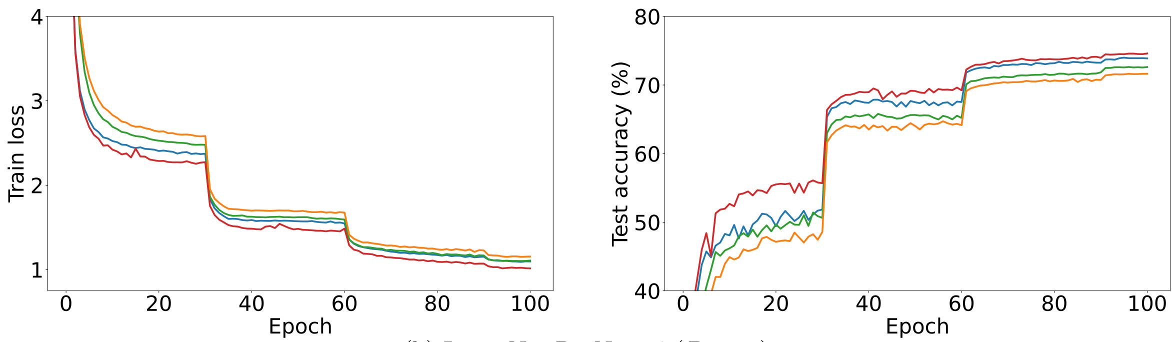
5 Discussion

KNResNets incorporate only batch-independent layers such as the proposed KernelNorm and KNConv layers into their architectures. Thus, they perform well with very small batch sizes (Table 1) and are applicable to differentially private learning (Table 4), which are not the case for the batch normalized models. Unlike the batch-independent competitors such as LayerNorm, GroupNorm, and LocalContextNorm based ResNets, KNResNets provide higher or very competitive performance compared to the batch normalized counterparts in image classification and semantic segmentation (Tables 1-3). Moreover, KNResNets converge faster than the batch, layer, and group normalized ResNets in non-private and differentially private image classification as shown in Figure 4. These results verify our key claim: the kernel normalized models combine the performance benefit of the batch normalized counterparts with the batch-independence advantage of the layer, group, and local-context normalized competitors.

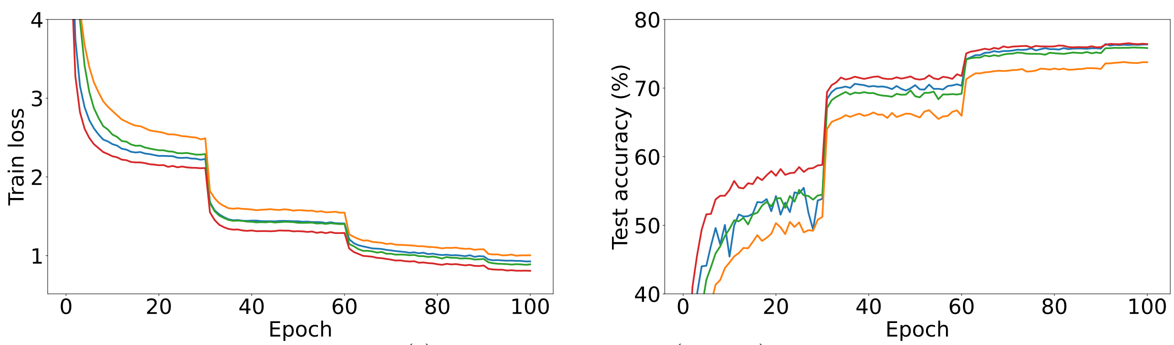
The key property of kernel normalization is the overlapping normalization units, which allows for kernel normalized models to *extensively* take advantage of the spatial correlation among the elements during normalization. Additionally, it enables KernelNorm to be combined with the convolutional layer effectively as a single KNConv layer (Equation 5 and Algorithm 1). The other normalization layers lack this property. BatchNorm, LayerNorm, and GroupNorm are global normalization layers, which completely ignore the spatial correlation of the elements. LocalContextNorm *partially* considers the spatial correlation dur-



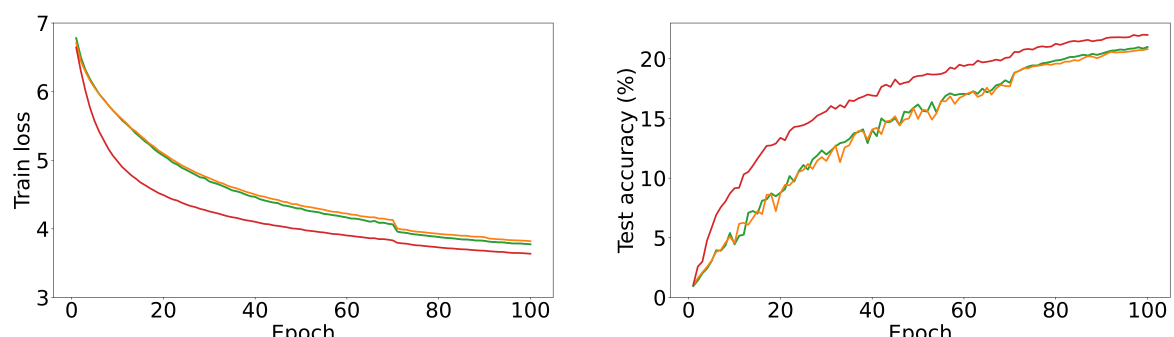
(a) CIFAR-100-ResNet-50 (B=2)



(b) ImageNet-ResNet-34 (B=256)



(c) ImageNet-ResNet-50 (B=256)



(d) ImageNet32×32-ResNet-18 (ε=8.0, δ=8×10^{-7})

Figure 4: **Convergence rate** of the models for different case studies: Kernel normalized models converge faster than the competitors. Notice that BatchNorm is inapplicable to differential privacy; B: batch size.

ing normalization because it has no overlapping normalization units, and must use very large window sizes to achieve practical computational efficiency. Our evaluations illustrate that this characteristic of kernel normalization lead to significant improvement in convergence rate and accuracy achieved by KNResNets.

Normalizing the feature values of the input images using the mean and SD of the whole dataset is a popular data preprocessing technique, which enhances the performance of the existing CNNs due to feeding the normalized values into the first convolutional layer. This is unnecessary for KNConvNets because all KNConv layers including the first one are self-normalizing (they normalize the input first, and then, compute the convolution). This makes the data preprocessing simpler during training of KNConvNets.

Compared to the corresponding non-normalized networks, the accuracy gain in KNResNets originates from normalization using KernelNorm and regularization effect of dropout. To investigate the contribution of each factor to the accuracy gain, we train KNResNet-50 on CIFAR-100 with batch size of 32 in three cases: (I) without KernelNorm, (II) with KernelNorm and without dropout, (III) with KernelNorm and dropout. The models achieve accuracy values of 71.48%, 78.32%, and 80.18% in (I), (II), and (III), respectively. Given that, normalization using KernelNorm provides accuracy gain of around 7.0% compared to the non-normalized model. Regularization effect of dropout delivers additional accuracy gain of about 2.0%.

Prior studies show that normalization layers can reduce the sharpness of the loss landscape, improving the generalization of the model (Lyu et al., 2022; Keskar et al., 2016). Given that, we train LayerNorm, GroupNorm, and BatchNorm based ResNet-18 as well as KNResNet-18 on CIFAR-10 to compare the generalization ability and loss landscape of different normalization methods (experimental details in Appendix C). The layer, group, batch, and kernel normalized models achieve test accuracy of 90.32%, 90.58%, 92.11%, 93.27%, respectively. Figure 7 (Appendix C) visualizes the loss landscape for different normalization layers. According to the figure, KNResNet-18 provides flatter loss landscape compared to batch normalized ResNet-18, which in turn, has smoother loss landscape than the group and layer normalized counterparts. These results indeed indicate that KNResNet-18 and BatchNorm-based ResNet-18 with flatter loss landscapes provide higher generalizability (test accuracy) than LayerNorm/GroupNorm based ResNet-18.

There is a prior work known as convolutional normalization (ConvNorm) (Liu et al., 2021), which takes into account the convolutional structure during normalization similar to this study. ConvNorm performs normalization on the kernel weights of the convolutional layer (weight normalization). Our proposed layers, on the other hand, normalize the input tensor (input normalization). In terms of performance on ImageNet, the accuracy of KNResNet-18 is higher than the accuracy of the ConvNorm+BatchNorm based ResNet-18 reported in Liu et al. (2021) (71.17% vs. 70.34%).

We explore the effectiveness of KernelNorm on the *ConvNext* architecture (Liu et al., 2022) in addition to ResNets. ConvNext is a convolutional architecture, but it is heavily inspired by vision transformers (Dosovitskiy et al., 2020), where it uses linear (fully-connected) layers extensively and employs LayerNorm as the normalization layer instead of BatchNorm. To draw the comparison, we train the original *ConvNextTiny* model from PyTorch and the corresponding kernel normalized version (both with around 28.5m parameters) on ImageNet using the training recipe and code from TorchVision (2023b) (more experimental details in Appendix B). The original model, which is based on LayerNorm, provides accuracy of 80.87%. The kernel normalized counterpart, on the other hand, achieves accuracy of 81.25%, which is 0.38% higher than the baseline. Given that, KernelNorm-based models are efficient not only with ResNets, but also with more recent architectures such as ConvNext, which incorporates several architectural elements from vision transformers into convolutional networks.

We also make a comparison between KNResNets and the BatchNorm-based counterparts from the computational efficiency and memory usage perspectives (Tables 6 and 7 in Appendix D). For the batch normalized models, we employ two different implementations of the BatchNorm layer: The CUDA implementation (PyTorch, 2023a) and the custom implementation (D2L, 2023) using primitives provided by PyTorch. Because the underlying layers of KNResNets (i.e. KernelNorm and KNConv) are implemented using primitives from PyTorch, we directly compare KNResNets with ResNets based on the latter implementation of BatchNorm to have a fair comparison. According to Table 6, KNResNet-50 (our largest model) is only slower than batch normalized ResNet-50 by factor of 1.66. This slowdown is acceptable given the fact that KernelNorm is a local normalization layer with much more normalization units than BatchNorm as a global normalization layer (Figure 1). The CUDA-based implementation of BatchNorm, moreover, is faster than that based on primitives from PyTorch by factor of 1.8. We can expect a similar speedup for KNResNets if the underlying layers are implemented in CUDA. Additionally, the memory usage of KNResNets is higher than the BatchNorm counterparts as expected, which relates to the current implementation of the KNConv layer (more details in Appendix D). Notice that the most efficient implementation of KNResNets is not the focus of this study, and is left as a future line of improvement. Our current implementation, however, provides enough efficiency that allows for training KNResNet-18/34/50 on large datasets such as ImageNet.

6 Conclusion and Future Work

BatchNorm considerably enhances the model convergence rate and accuracy, but it delivers poor performance with small batch sizes. Moreover, it is unsuitable for differentially private learning due to its dependence on the batch statistics. To address these challenges, we propose two novel batch-independent layers called KernelNorm and KNConv, and employ them as the main building blocks for KNConvNets, and the corresponding residual networks referred to as KNResNets. Through extensive experimentation, we show KNResNets deliver higher or very competitive accuracy compared to BatchNorm counterparts in image classification and semantic segmentation. Furthermore, they consistently outperform the batch-independent counterparts such as LayerNorm, GroupNorm, and LocalContextNorm in non-private and differentially private learning settings. To our knowledge, our work is the first to combine the batch-independence of LayerNorm/GroupNorm/LocalContextNorm with the performance advantage of BatchNorm in the context of convolutional networks.

The performance investigation of KNResNets for object detection, designing KNConvNets corresponding to other popular architectures such as DenseNets (Huang et al., 2017a), and optimized implementations of KernelNorm and KNResNets in CUDA are promising directions for future studies.

Acknowledgement

We would like to thank *Javad TorkzadehMahani* for assisting with the implementations and helpful discussions on the computationally-efficient version of the kernel normalized convolutional layer. We would also like to thank *Sameer Ambekar* for his helpful suggestion regarding fairer comparison among the normalization layers from the computational efficiency perspective.

This project was funded by the German Ministry of Education and Research as part of the PrivateAIM Project, by the Bavarian State Ministry for Science and the Arts, and by the Medical Informatics Initiative. The authors of this work take full responsibility for its content.

References

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

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. Can we gain more from orthogonality regularizations in training deep networks? *Advances in Neural Information Processing Systems*, 31, 2018.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

AB Bonds. Role of inhibition in the specification of orientation selectivity of cells in the cat striate cortex. *Visual neuroscience*, 2(1):41–55, 1989.

Andrew Brock, Soham De, and Samuel L Smith. Characterizing signal propagation to close the performance gap in unnormalized resnets. *arXiv preprint arXiv:2101.08692*, 2021a.

Andy Brock, Soham De, Samuel L Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. In *International Conference on Machine Learning*, pp. 1059–1071. PMLR, 2021b.

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3213–3223, 2016.

D2L. Batch normalization. `https://d2l.ai/chapter_convolutional-modern/batch-norm.html`, 2023.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9:211–407, 2014.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016b.

David J Heeger. Normalization of cell responses in cat striate cortex. *Visual neuroscience*, 9(2):181–197, 1992.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017a.

Lei Huang, Xianglong Liu, Yang Liu, Bo Lang, and Dacheng Tao. Centered weight normalization in accelerating training of deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2803–2811, 2017b.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, 2016.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Liu Kuang. Pytorch models for ciafr-10/100. `https://github.com/kuangliu/pytorch-cifar/`, 2021.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1 (4):541–551, 1989.

Boyi Li, Felix Wu, Kilian Q Weinberger, and Serge Belongie. Positional normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018a.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. `https://github.com/tomgoldstein/loss-landscape`, 2018b.

Sheng Liu, Xiao Li, Yuexiang Zhai, Chong You, Zhihui Zhu, Carlos Fernandez-Granda, and Qing Qu. Convolutional normalization: Improving deep convolutional network robustness and training. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11976–11986, 2022.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015a.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015b.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

Kaifeng Lyu, Zhiyuan Li, and Sanjeev Arora. Understanding the generalization benefit of normalization layers: Sharpness reduction. *Advances in Neural Information Processing Systems*, 35:34689–34708, 2022.

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

Diganta Misra. Mish: A self regularized non-monotonic activation function. *arXiv preprint arXiv:1908.08681*, 2019.

Anthony Ortiz, Caleb Robinson, Dan Morris, Olac Fuentes, Christopher Kiekintveld, Md Mahmudulla Hassan, and Nebojsa Jojic. Local context normalization: Revisiting local normalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11276–11285, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

PyTorch. Batch normalization. `https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html`, 2023a.

PyTorch. Unfold operation in pytorch. `https://pytorch.org/docs/stable/generated/torch.nn.Unfold.html`, 2023b.

PyTorch. var_mean function in pytorch. `https://pytorch.org/docs/stable/generated/torch.var_mean.html`, 2023c.

Haozhi Qi, Chong You, Xiaolong Wang, Yi Ma, and Jitendra Malik. Deep isometric learning for visual recognition. In *International conference on machine learning*, pp. 7824–7835. PMLR, 2020.

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

Mengye Ren, Renjie Liao, Raquel Urtasun, Fabian H. Sinz, and Richard S. Zemel. Normalizing the normalizers: Comparing and extending network normalization schemes. In *International Conference on Learning Representations*, 2017.

Tim Salimans and Diederik P Kingma. Weight normalization: a simple reparameterization to accelerate training of deep neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 901–909, 2016.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *Advances in neural information processing systems*, 31, 2018.

Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *2nd International Conference on Learning Representations, ICLR 2014*, 2014.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1): 1929–1958, 2014.

Ke Sun, Yang Zhao, Borui Jiang, Tianheng Cheng, Bin Xiao, Dong Liu, Yadong Mu, Xinggang Wang, Wenyu Liu, and Jingdong Wang. High-resolution representations for labeling pixels and regions. *arXiv preprint arXiv:1904.04514*, 2019.

TorchVision. Classification training script in pytorch. `https://github.com/pytorch/vision/tree/main/references/classification#resnet`, 2023a.

TorchVision. Classification training script in pytorch. `https://github.com/pytorch/vision/tree/main/references/classification#convnext`, 2023b.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

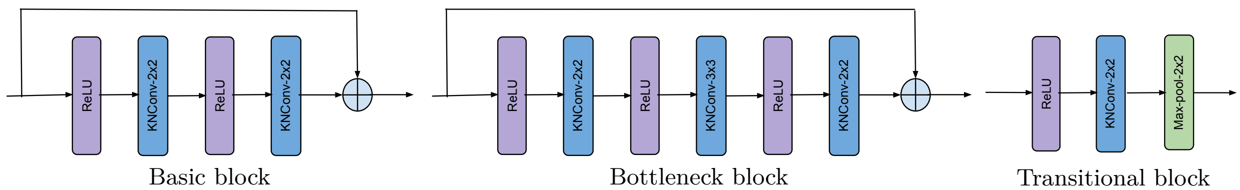
Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Jiayun Wang, Yubei Chen, Rudrasis Chakraborty, and Stella X Yu. Orthogonal convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11505–11515, 2020.

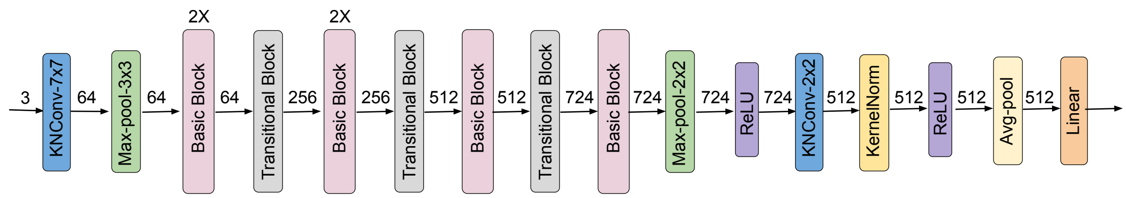
Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

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.

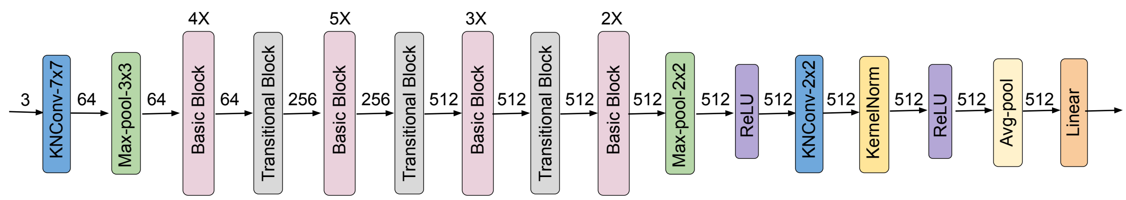
A KNResNet Architectures



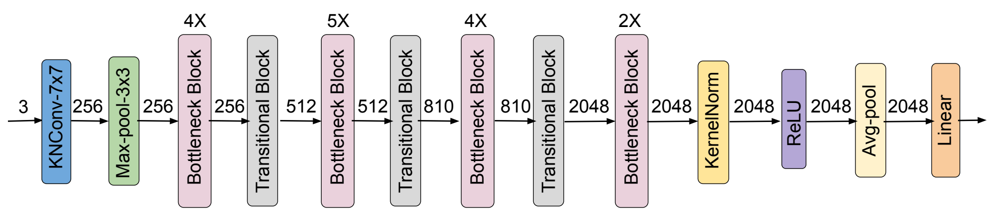
(a) KNResNet blocks (image classification)



(b) KNResNet-18 (image classification)

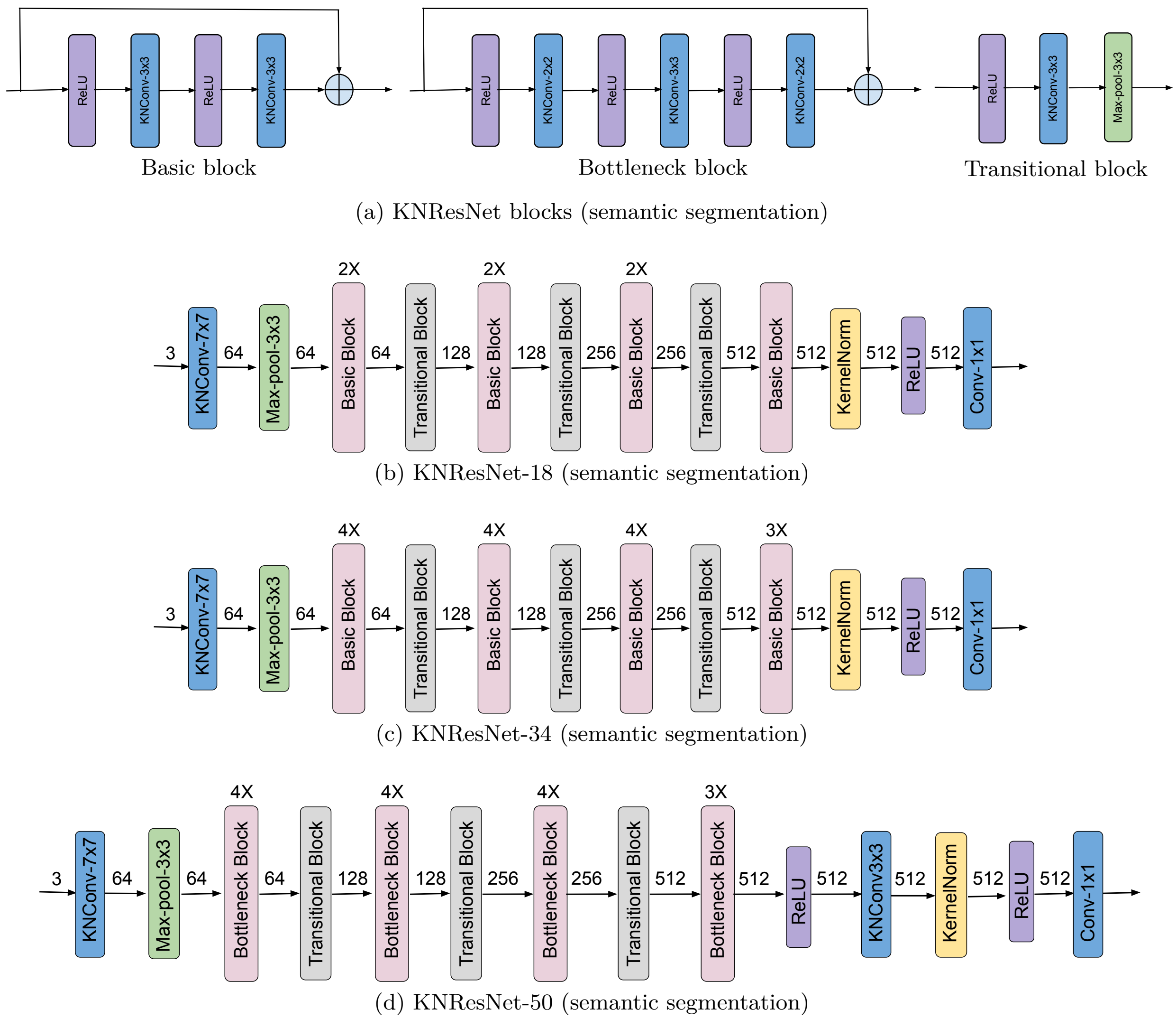


(c) KNResNet-34 (image classification)



(d) KNResNet-50 (image classification)

Figure 5: **KNResNets** for **image classification**: The dropout probability of the KNConv and KernelNorm layers are 0.05 and 0.25, respectively. For low-resolution images (e.g. CIFAR-100 with image shape of 32×32), the first KNConv layer is replaced by a KNConv layer with kernel size 3×3, stride 1×1, and padding 1×1, and the following max-pooling layer is removed. The kX (k=2/3/4/5) notation above the blocks means k blocks of that type. The numbers above arrows indicate the number of input/output channels of the first/last KNConv layer in the block. For KNResNet-18, the number of the output channels of the first KNConv layer (or the number of input channels of the second KNConv layer) is 256, 256, 512, and 724 for the first, second, third, and fourth set of basic blocks, respectively. For KNResNet-34, it is 256, 320, 640, and 843. For KNResNet-50, the number of output channels of the first and second KNConv layers are 64, 128, 201, and 512 in the first, second, third, and fourth set of bottleneck blocks, respectively. In KNResNet-50, the last transitional block and the last set of residual blocks use KNConv1×1 instead of KNConv2×2 to keep the number of parameters comparable to the original ResNet-50.



(a) KNResNet blocks (semantic segmentation)

(b) KNResNet-18 (semantic segmentation)

(c) KNResNet-34 (semantic segmentation)

(d) KNResNet-50 (semantic segmentation)

Figure 6: **KNResNets** for **semantic segmentation**: The dropout probability of the KNConv and KernelNorm layers are 0.1 and 0.5, respectively. For KNResNet-18, the number of the output channels of the first KNConv layer (or the number of input channels of the second KNConv layer) is 128, 256, 512, and 625 for the first, second, third, and fourth set of basic blocks. For KNResNet-34, they are 128, 256, 256, and 512, respectively. For KNResNet-50, the number of input/output channels of the middle KNConv layer are 128, 256, 458, and 512 for the first, second, third, and fourth set of bottleneck blocks. Unlike their counterparts for image classification, the KNConv and max-pooling layers in basic and transitional blocks employ kernel size of 3×3 instead of 2×2.

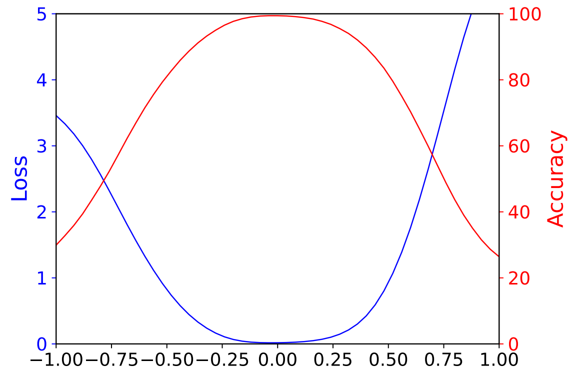
B Reproducibility

Table 5: Learning rate values achieving the highest accuracy on CIFAR-100.

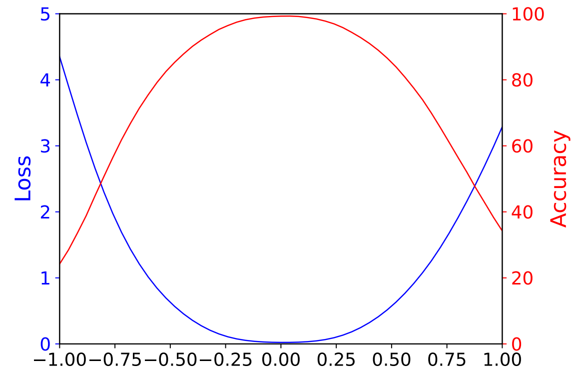
Model	Normalization	B=2	B=32	B=256
ResNet-18-LN	LayerNorm	0.0015625	0.0125	0.05
PreactResNet-18-LN	LayerNorm	0.0015625	0.0125	0.05
ResNet-18-GN	GroupNorm	0.0015625	0.025	0.1
PreactResNet-18-GN	GroupNorm	0.0015625	0.025	0.1
ResNet-18-BN	BatchNorm	0.00078125	0.025	0.2
PreactResNet-18-BN	BatchNorm	0.00078125	0.025	0.2
KNResNet-18	KernelNorm	0.0015625	0.05	0.2
ResNet-34-LN	LayerNorm	0.0015625	0.0125	0.05
PreactResNet-34-LN	LayerNorm	0.0015625	0.0125	0.05
ResNet-34-GN	GroupNorm	0.0015625	0.025	0.1
PreactResNet-34-GN	GroupNorm	0.0015625	0.025	0.1
ResNet-34-BN	BatchNorm	0.00078125	0.025	0.1
PreactResNet-34-BN	BatchNorm	0.000390625	0.025	0.2
KNResNet-34	KernelNorm	0.0015625	0.05	0.2
ResNet-50-LN	LayerNorm	0.00078125	0.0125	0.05
PreactResNet-50-LN	LayerNorm	0.0015625	0.0125	0.05
ResNet-50-GN	GroupNorm	0.00078125	0.0125	0.05
PreactResNet-50-GN	GroupNorm	0.0015625	0.025	0.1
ResNet-50-BN	BatchNorm	0.000390626	0.0125	0.1
PreactResNet-50-BN	BatchNorm	0.000195313	0.0125	0.2
KNResNet-50	KernelNorm	0.0015625	0.025	0.2

ConvNext on ImageNet: To train the LayerNorm and KernelNorm based ConvNextTiny models on ImageNet, we employ the code and recipe from TorchVision (2023b), where the models are trained with total batch size of 1024 using the AdamW optimizer, learning rate of 0.001, and cosine learning rate scheduler for 600 epochs. Note that we use 4 GPUs with batch size of 256 per GPU rather than 8 GPUs with batch size of 128 per GPU in the original recipe due to the resource limitation.

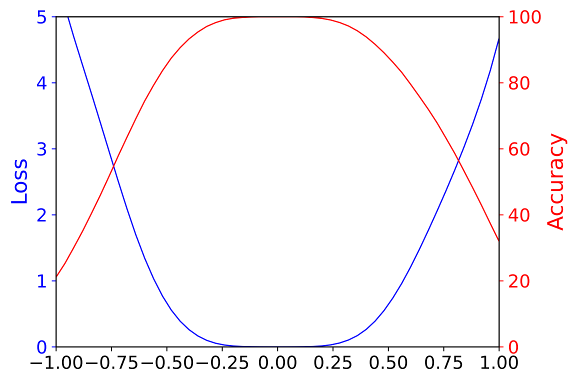
C Loss Landscape



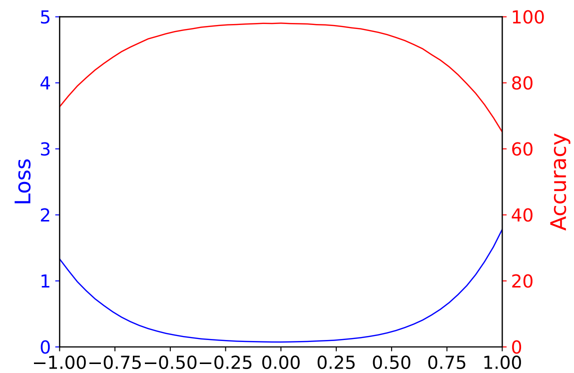
(a) LayerNorm



(b) GroupNorm



(c) BatchNorm



(d) KernelNorm

Figure 7: **Loss landscape** of different normalization layers: Kernel normalized ResNet-18 has flatter loss landscape compared to the batch, group, and layer normalized counterparts on CIFAR-10.

ResNet-18 on CIFAR-10: To compare the generalization ability and loss landscape of different normalization layers, we train BatchNorm, GroupNorm, LayerNorm, and KernelNorm based ResNet-18 on CIFAR-10. All models are trained for 70 epochs using batch size of 128 and tuned over learning rate values of $\{0.05, 0.1\}$. The weight decay is zero. The optimal learning rate is 0.05/0.05/0.1/0.1 for layer/group/batch/kernel normalized ResNet-18. The preprocessing and augmentation scheme and the other training settings are the same as the CIFAR-100 experiments in Section 4. We employ the source code from Li et al. (2018a;b) to visualize the loss landscape in Figure 7.

D Running Time and Memory Usage

Table 6: **Training and inference** time per epoch for ImageNet: The experiments are conducted with 8 NVIDIA A40 GPUs with batch size of 32 per GPU; m: minutes, s: seconds.

Model	Normalization	Implementation	Training time	Inference time
ResNet-50-BN	BatchNorm	CUDA	13m 23s	6s
ResNet-50-BN	BatchNorm	Primitives from PyTorch	23m 49s	10s
KNResNet-50 (ours)	KernelNorm	Primitives from PyTorch	39m 33s	19s
ResNet-34-BN	BatchNorm	CUDA	9m 12s	5s
ResNet-34-BN	BatchNorm	Primitives from PyTorch	12m 46s	5s
KNResNet-34 (ours)	KernelNorm	Primitives from PyTorch	27m 15s	12s
ResNet-18-BN	BatchNorm	CUDA	5m 28s	4s
ResNet-18-BN	BatchNorm	Primitives from PyTorch	7m 46s	4s
KNResNet-18 (ours)	KernelNorm	Primitives from PyTorch	13m 58s	7s

Table 7: **Memory usage** on ImageNet: The experiments are conducted with a single NVIDIA RTX A6000 GPU with batch size of 32; GB: Gigabytes.

Model	Normalization	Implementation	Memory usage (GB)
ResNet-50-BN	BatchNorm	CUDA	5.7
ResNet-50-BN	BatchNorm	Primitives from PyTorch	8.2
KNResNet-50 (ours)	KernelNorm	Primitives from PyTorch	13.6
ResNet-34-BN	BatchNorm	CUDA	3.6
ResNet-34-BN	BatchNorm	Primitives from PyTorch	4.4
KNResNet-34 (ours)	KernelNorm	Primitives from PyTorch	9.4
ResNet-18-BN	BatchNorm	CUDA	3.2
ResNet-18-BN	BatchNorm	Primitives from PyTorch	3.7
KNResNet-18 (ours)	KernelNorm	Primitives from PyTorch	7.2

The memory usage of KNResNets is higher than the BatchNorm counterparts. This observation is related to the current implementation of the KNConv layer, where the unfolding operation is performed in the kn_mean_var function (Algorithm 1) to compute the mean and variance of the units. We implemented KNConv in this fashion to avoid changing the CUDA implementation of the convolutional layer, which requires a huge engineering and implementation effort, and is outside the scope of our expertise.

In a hypothetical implementation of KNConv in CUDA, it would be possible to compute the mean/variance of the units directly inside the convolutional layer, and completely remove the kn_mean_var function, leading to substantially reducing the memory usage. This is because the units to compute convolution and mean/variance are the same, and those units are already available in the convolutional layer implementation.

Table 8: **Inference time | memory usage** for different stride, width (W) and height (H) values. The experiments are carried out with a single NVIDIA RTX A6000 GPU using batch size of 256 on the test set of CIFAR-100. The model contains four KNConv layers with kernel size of 3×3 and 256 channels; s: seconds, GB: Gigabytes.

	W/H=32×32	W/H=64×64	W/H=128×128
Stride=1×1	2.44s	2.80GB	8.43s	4.94GB	33.45s	13.44GB
Stride=2×2	0.64s	2.24GB	1.07s	2.72GB	2.91s	4.59GB
Stride=3×3	0.58s	2.16GB	0.79s	2.40GB	1.71s	3.28GB

Kernel Normalized Convolutional Networks for Privacy-Preserving Machine Learning

Reza Nasirigerdeh, Javad Torkzadehmahani, Daniel Rueckert & Georgios Kaissis

Conference: IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)

Synopsis: Normalization is an important but understudied challenge in privacy-related application domains such as federated learning (FL), differential privacy (DP), and differentially private federated learning (DP-FL). While the unsuitability of batch normalization for these domains has already been shown, the impact of other normalization methods on the performance of federated or privacy-preserving models is not well-known. To address this, we draw a performance comparison among layer normalization (*LayerNorm*), group normalization (*GroupNorm*), and our recently proposed kernel normalization (*KernelNorm*) in FL, DP, and DP-FL settings. Our results indicate LayerNorm and GroupNorm provide no performance gain compared to the baseline (i.e. no normalization) for shallow models in FL and DP. They, on the other hand, considerably enhance the performance of shallow models in DP-FL and deeper models in FL and DP. KernelNorm, moreover, significantly outperforms its competitors in terms of accuracy and convergence rate (or communication efficiency) for both shallow and deeper models in all considered learning environments. Given these key observations, we propose a kernel normalized ResNet architecture called KNResNet-13 for differentially private learning. Using the proposed architecture, we provide new state-of-the-art accuracy values on the CIFAR-10 and Imagenette datasets, when trained from scratch.

Contributions of thesis author: Leading role in conceiving and designing the study, gathering software resources, developing algorithms, implementing the software components, conducting the experiments, writing the manuscript, and significant contribution in scientific findings.

Publisher: the Institute of Electrical and Electronics Engineers (IEEE)

Date: February 2023

Copyright: the Institute of Electrical and Electronics Engineers (IEEE)

Reprint permission: According to IEEE, "the IEEE does not require individuals working on a thesis to obtain a formal reuse license." Note that the accepted version of the paper is included in the dissertation. Please see the following page for the complete description of the IEEE license.

Kernel Normalized Convolutional Networks for Privacy-Preserving Machine Learning

Conference Proceedings:
2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)

Author: Reza Nasirigerdeh

Publisher: IEEE

Date: February 2023

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Kernel Normalized Convolutional Networks for Privacy-Preserving Machine Learning

Reza Nasirigerdeh
Technical University of Munich
Klinikum rechts der Isar
Munich, Germany

Javad Torkzadehmahani
Azad University of Kerman
Kerman, Iran

Daniel Rueckert
Technical University of Munich
Klinikum rechts der Isar
Munich, Germany
Imperial College London
London, United Kingdom

Georgios Kaissis
Technical University of Munich
Helmholtz Zentrum Munich
Munich, Germany

***Abstract*—Normalization is an important but understudied challenge in privacy-related application domains such as federated learning (FL), differential privacy (DP), and differentially private federated learning (DP-FL). While the unsuitability of batch normalization for these domains has already been shown, the impact of other normalization methods on the performance of federated or differentially private models is not well-known. To address this, we draw a performance comparison among layer normalization (LayerNorm), group normalization (GroupNorm), and the recently proposed kernel normalization (KernelNorm) in FL, DP, and DP-FL settings. Our results indicate LayerNorm and GroupNorm provide no performance gain compared to the baseline (i.e. no normalization) for shallow models in FL and DP. They, on the other hand, considerably enhance the performance of shallow models in DP-FL and deeper models in FL and DP. KernelNorm, moreover, significantly outperforms its competitors in terms of accuracy and convergence rate (or communication efficiency) for both shallow and deeper models in all considered learning environments. Given these key observations, we propose a kernel normalized ResNet architecture called KNResNet-13 for differentially private learning. Using the proposed architecture, we provide new state-of-the-art accuracy values on the CIFAR-10 and Imagenette datasets, when trained from scratch.**

***Index Terms*—Differential Privacy, Federated Learning, Kernel Normalization, Group Normalization, Batch Normalization**

I. Introduction

Deep convolutional neural networks (*CNNs*) are popular in a diverse range of image vision tasks including image classification [1]. Deep CNNs rely on large-scale datasets to effectively train the model, which might be difficult to provide in a centralized manner [2]. This is because datasets are often distributed across different sites such as hospitals, and contain sensitive data which cannot be transferred to a centralized location due to privacy regulations [3]. Even if such datasets become available, training algorithms can pose privacy risks to the individuals participating in the dataset, leaking privacy-sensitive information through the trained model [4]–[6].

To appear in the IEEE Conference on Secure and Trustworthy Machine Learning (SaTML), February 2023.

Federated learning (*FL*) [7] addresses the large-scale data availability challenge by enabling clients to jointly train a global model under the coordination of a central server without sharing their private data. *Network communication*, on the other hand, emerges as a new challenge in federated environments, requiring a large number of communication rounds for model convergence, and exchanging a large amount of traffic in each round [8]. FL also causes utility (e.g. in terms of accuracy) reduction due to the *Non-IID* (not independent and identically distributed) nature of the data across the clients [9]. Finally, although FL eliminates the requirement of data sharing, it might still lead to privacy leakage, where the private data of the clients can be reconstructed from the model updates shared with the server [10]–[12].

Differential privacy (DP) [13] copes with the privacy challenge in both centralized and federated environments by injecting random noise into the model gradients to limit the information learnt about a particular sample in the dataset [14]. DP, however, adversely affects the model utility similar to FL because of the injected noise. In general, there is a trade-off between privacy and utility in DP, where stronger privacy leads to lower utility [15].

Batch normalization (BatchNorm) [16] is the de facto normalization layer in popular deep CNNs such as ResNets [17] and DenseNets [18], which remarkably improves the model convergence rate and accuracy in centralized training. BatchNorm, however, is not suitable for FL and DP settings. This is because BatchNorm relies on the IID distribution of feature values in the batch [16], which is not the case in federated settings. Moreover, per-sample gradients are required to be computed in DP that is impossible for batch-normalized CNNs [14]. *Batch-independent* layers such as *layer normalization* (LayerNorm) [19], *group normalization* (GroupNorm) [20], and the recently proposed *kernel normalization* (KernelNorm) [21] do not suffer from the BatchNorm's limitations, and therefore, are applicable to FL and DP.

Normalization challenge. Unsuitability of BatchNorm for federated and differentially private learning has presented a real challenge in the corresponding environments. Unlike the other challenges (i.e. utility, network communication, and privacy), the normalization issue has remained understudied in the context of FL and DP. Previous works [9], [22] illustrate that GroupNorm outperforms BatchNorm in terms of accuracy in federated settings. Likewise, GroupNorm also delivers higher accuracy than LayerNorm in differentially private learning [23]–[25]. Additionally, KernelNorm achieves significantly higher accuracy and faster convergence rate compared to LayerNorm and GroupNorm in both FL and DP settings according to the original study [21].

However, the prior studies have not made a comparison between different normalization layers and the NoNorm (no normalization layer) case in the first place. Moreover, the experimental evaluation regarding FL and DP environments is limited in the original KernelNorm study [21], focusing on a cross-silo federated setting (few clients with relatively large datasets) [26] and a shallow model in DP. Finally, the performance comparisons in the previous works do not consider differentially private federated learning (DP-FL) settings. Given that, two fundamental questions arise: (1) *Do LayerNorm, GroupNorm, and KernelNorm also deliver higher performance than NoNorm in FL, DP, and DP-FL environments?*, and (2) *Does KernelNorm still outperform other normalization layers in cross-device FL (many clients with small datasets), in DP-FL, and using deeper models in DP?*

Key findings. We conduct extensive experiments using the VGG-6 [27], ResNet-8 [21], PreactResNet-18 [28], and DenseNet20×16 [18] models trained on the CIFAR-10/100 [29] and Imagenette [30] datasets in FL, DP, and DP-FL settings to address those questions. The findings are as follows:

1) LayerNorm and GroupNorm do not necessarily outperform the NoNorm case for shallow models in FL and DP settings. For instance, LayerNorm and GroupNorm provide slightly lower accuracy and communication efficiency than NoNorm in the cross-silo federated setting, where the shallow VGG-6 model is trained on CIFAR-10. Similarly, LayerNorm and GroupNorm achieve lower accuracy than NoNorm using the shallow ResNet-8 model on CIFAR-10 in DP (Section III).
2) KernelNorm significantly outperforms NoNorm, LayerNorm, and GroupNorm in terms of communication efficiency (convergence rate) and accuracy in both cross-silo and cross-device FL, with both shallow and deeper models in DP, and using shallow models in DP-FL environments (Section III).

Solution. Based on our findings, we advocate employing KernelNorm as the effective normalization layer for FL, DP, and DP-FL settings. Given that, we propose a KernelNorm-based ResNet architecture called KNResNet-13, and show it delivers considerably higher accuracy than the state-of-the-art GroupNorm-based architectures on CIFAR-10 and Imagenette in differentially private learning environments (Section IV).

Contributions. We make the following contributions: (I) we show LayerNorm and GroupNorm do not deliver higher accuracy than NoNorm with shallow models in FL and DP settings, (II) we illustrate the recently proposed KernelNorm layer has a great potential to become the de facto normalization layer in privacy-enhancing/preserving machine learning, and (III) we propose the KNResNet-13 architecture, and provide new state-of-the-art (SOTA) accuracy values on CIFAR-10 and Imagenette using the proposed architecture in DP environments, when trained from scratch.

II. Preliminaries

Federated learning (FL). A federated environment consists of multiple clients as data holders and a central server as coordinator. FL is a privacy-enhancing technique, which enables the clients to train a global model without sharing their private data with a third party. In FL, or more precisely in the `FederatedAveraging` (*FedAvg*) algorithm [7], the server randomly chooses K clients, and sends them the global model parameters W_i^g in each communication round i. Next, each selected client j trains the global model on its local dataset using *mini-batch gradient descent*, and shares the local model parameters $W_{i,j}^l$ with the server. Finally, the server takes the weighted average over the local parameters from the clients to update the global model:

$$W_{i+1}^g = \frac{\sum_{j=1}^{K} N_j \cdot W_{i,j}^l}{\sum_{j=1}^{K} N_j},$$

where N_j is the number of samples in client j.

A *cross-device* federated setting contains a large number of clients such as mobile devices with small datasets [26]. The server selects a fraction of clients in each round. Moreover, the underlying assumption is that the communication between clients and server is unstable, and the clients might drop out during training. A *cross-silo* setting, on the other hand, consists of few clients such as hospitals or research institutions with relatively large datasets and stable network connection [26]. All clients participate in model training in all communication rounds. For more details on federated learning, the readers are referred to [7] and [26].

Differential privacy (DP). The differential privacy approach provides a theoretical framework and collection of techniques for privacy-preserving data processing and release [13]. Its guarantees are formulated in an information-theoretic fashion and describe the upper bound on the multiplicative information gain of an adversary observing the output of a computation over a sensitive database. This definition endows DP with a robust theoretical underpinning and ascertains that its guarantees hold in the presence of adversaries with unbounded prior knowledge and under infinite post-processing. Moreover, DP guarantees are *compositional*, meaning that they degrade predictably when a DP system is executed repeatedly on the same database. Formally, a randomised mechanism $\mathcal{M}$ is said to preserve (ε, δ)-DP if, for all databases D and D'

differing in the data of one individual and all measurable subsets S of the range of $\mathcal{M}$, the following inequality holds:

$$\mathbb{P}(\mathcal{M}(D) \in S) \leq e^{\varepsilon}\mathbb{P}(\mathcal{M}(D') \in S) + \delta,$$

where $\mathbb{P}$ is the probability of an event, $\varepsilon \geq 0$ and $0 \leq \delta < 1$. Of note, this inequality must hold also if D and D' are swapped. The guarantee is given over the randomness of $\mathcal{M}$. Intuitively, this characterisation implies that the output of the mechanism should not change *too much* when one individual's data is added or removed from a database, or equivalently, the influence of one individual's data on the result of the computation should be small.

The application of DP to the training of neural networks is usually (and in our work) based on the differentially private stochastic gradient descent (DP-SGD) algorithm [14]. Here, the role of the database is played by the individual (per-sample) gradients of the loss function with respect to the parameters. For the DP guarantee to be well-defined, the intermediate layer outputs (activations), leading to the computation of a per-sample gradient, are not allowed to be influenced by more than one sample. Hence, layers like BatchNorm, which normalize the activations of a layer by considering either other samples in the batch or the statistics of previously seen batches, cannot be employed in DP. We refer the readers to [13], [14], [31] for more information on differential privacy.

Differentially private federated learning (DP-FL). Although FL enhances data privacy by eliminating the requirement of data sharing, the model parameters shared with the server can still cause privacy leakage. To overcome this problem, the clients can rely on DP to train the global model on their local data, and share differentially private models with the server. This way, the clients can benefit from the guarantees of DP in federated environments.

Normalization. The normalization layers play a crucial role in deep CNNs. They can smoothen the optimization landscape [32] and effectively address the problem of vanishing gradients [33], leading to improved model performance. The normalization layers are different from each other in their normalization unit, which is a subset of elements from the original input that are normalized together with the mean and variance of the unit [21]. Assume that the input is a 4-dimensional tensor with batch, channel, height, and width as dimensions. BatchNorm [16] considers all elements in the batch, height, and width dimensions as its normalization unit. LayerNorm [19], on the other hand, performs normalization across all elements in the channel, height, and width dimensions but separately for each sample in the batch. The normalization unit of GroupNorm [20] contains all elements in the height and width dimensions similar to LayerNorm, but a subset of elements (specified by the group size) in the channel dimension.

BatchNorm, LayerNorm, and GroupNorm are referred to as *global normalization* layers because they consider all elements in the height and width dimensions during normalization [34]. There is also a one-to-one correspondence between the input and output elements in the aforementioned layers, implying that they do not modify the input shape [21]. These layers have *shift* and *scale* as learnable parameters too for ensuring that the distributions of the input and output elements remain similar [16]. In contrast to BatchNorm, LayerNorm and GroupNorm are *batch-independent* because they perform normalization separately for each sample in the batch.

KernelNorm [21] performs normalization along the channel, height, and width dimensions but independently of the batch dimension akin to LayerNorm and GroupNorm. The normalization unit of KernelNorm, however, is a tensor of shape (c, k_h, k_w), where c is the number of input channels, and (k_h, k_w) is the kernel size. Thus, KernelNorm considers *all elements* in the channel dimension but a *subset of elements* specified by the kernel size from the height and width dimensions during normalization. In simple words, KernelNorm is similar to the pooling layers, except that KernelNorm normalizes the elements instead of computing average or maximum, and carries out operation over all channels rather than on a single channel.

Formally, KernelNorm (1) applies dropout to the original normalization unit U to obtain the *dropped-out* unit U', (2) calculates the mean and variance of U', and (3) employs the computed mean and variance to normalize U:

$$U' = D_p(U), \tag{1}$$

$$\begin{aligned} \mu_{u'} &= \frac{1}{c \cdot k_h \cdot k_w} \cdot \sum_{i_c=1}^{c} \sum_{i_h=1}^{k_h} \sum_{i_w=1}^{k_w} U'(i_c, i_h, i_w), \\ \sigma_{u'}^2 &= \frac{1}{c \cdot k_h \cdot k_w} \cdot \sum_{i_c=1}^{c} \sum_{i_h=1}^{k_h} \sum_{i_w=1}^{k_w} (U'(i_c, i_h, i_w) - \mu_{u'})^2, \end{aligned} \tag{2}$$

$$\hat{U} = \frac{U - \mu_{u'}}{\sqrt{\sigma_{u'}^2 + \epsilon}}, \tag{3}$$

where p is the dropout [35] probability, $\mu_{u'}$ and $\sigma_{u'}^2$ are the mean and variance of U', respectively, and $\hat{U}$ is the normalized unit. Partially inspired by BatchNorm, KernelNorm introduces a regularizing effect during training through normalizing the elements of the original unit U via the statistics calculated over the dropped-out unit U'.

KernelNorm is a *local normalization* layer. Moreover, it has no learnable parameters, and its output might have very different shape than the input. Similar to LayerNorm and GroupNorm, KernelNorm is batch-independent because it performs normalization separately for each sample of the batch. The *kernel normalized convolutional* (*KNConv*) layer [21] is the combination of the KernelNorm and convolutional layer, where the output of the former is given as input to the latter.

The modern CNNs are batch-normalized, leveraging the BatchNorm and convolutional layers in their architectures. The corresponding layer/group-normalized networks are obtained by simply replacing BatchNorm with LayerNorm/GroupNorm. The kernel-normalized counterparts [21], on the other hand, employ the KernelNorm and KNConv layers as the main building blocks, while forgoing the BatchNorm layers. For more details on the normalization layers, the readers can see [16], [19]–[21].

III. Evaluation

We conduct extensive experiments to investigate the performance of different batch-independent normalization layers including LayerNorm, GroupNorm, and KernelNorm in the cross-silo and cross-device FL as well as DP and DP-FL environments. In the following, we first provide the description of the datasets, models, and case studies, and then discuss the results and findings.

A. Experimental Setup

Datasets. The CIFAR-10/100 dataset [29] contains 50000 train and 10000 test samples of shape 32×32 from 10/100 classes. The Imagenette dataset (160-pixel version) [30] is a subset of Imagenet [36], including 9469 train and 3925 validation images from 10 "easily classified" labels. The feature values are divided by 255 for KernelNorm based models, whereas they are normalized using the mean and standard deviation of CIFAR-10/100 or ImageNet for NoNorm, LayerNorm, and GroupNorm based counterparts. The samples of Imagenette are resized to 128×128.

Models. We adopt the VGG-6 architecture from [27], ResNet-8 model from [21], PreactResNet-18 implementation from [37], and DenseNet-20×16 (depth of 20 and growth rate of 16) implementation from [38]. In layer/group-normalized networks, BatchNorm is substituted by LayerNorm/GroupNorm. In the NoNorm case, the BatchNorm layers are either removed or replaced with the identity layer. The kernel-normalized counterparts are implemented by removing the BatchNorm layers, replacing the convolutional layers with KNConv, and inserting a KernelNorm layer before the final average-pooling layer in the ResNet, PreactResNet, and DenseNet models. In FL, the models employ the ReLU activation. In DP, on the other hand, the activation function is Mish [39], which was successfully used in [24] to achieve SOTA accuracy. We implement the models in the PyTorch library (version 1.11) [40].

Case Studies. We design nine different case studies (four in FL, three in DP, and two in DP-FL) to make the performance comparison among the normalization layers:

1) **CIFAR-10-VGG-6 (cross-silo FL):** This case study aims to train the *shallow* VGG-6 model on the *low-resolution* CIFAR-10 dataset in a cross-silo federated environment containing 10 clients, where each client has samples from only 2 classes. The sample sizes of the clients are almost the same.
2) **CIFAR-10-VGG-6 (cross-device FL):** Similar to the cross-silo counterpart, but in a cross-device federated setting including 100 clients, where 20 clients are randomly selected in each round.
3) **CIFAR-100-PreactResNet-18 (cross-silo FL):** The aim of this case study is to train the *deeper* PreactResNet-18 model on *more challenging*, low-resolution CIFAR-100 dataset in a cross-silo federated environment consisting of 10 clients with samples from 20 labels. The clients have highly similar sample sizes.
4) **CIFAR-100-PreactResNet-18 (cross-device FL):** Akin to the cross-silo counterpart, but in a cross-device federated setting consisting of 100 clients, where 20 clients are randomly chosen by the server in each round.
5) **CIFAR-10-ResNet-8 (DP):** The goal of this case study is to train the *shallow* ResNet-8 model on the *low-resolution* CIFAR-10 dataset in the DP environment.
6) **CIFAR-10-DenseNet-20×16 (DP):** This case study aims to train the *deeper* DenseNet-20×16 model on the *low-resolution* CIFAR-10 dataset in the DP setting.
7) **Imagenette-PreactResNet-18 (DP).** The purpose of this case study is to train the *deeper* PreactResNet-18 model on the *medium-resolution* Imagenette dataset in the differentially private environment.
8) **CIFAR-10-VGG-6 (DP-FL):** This case study aims to train the VGG-6 model on the CIFAR-10 dataset in a *differentially private federated setting* with 10 clients, where the clients have samples from 4 classes. The sample sizes of the clients are highly similar.
9) **CIFAR-10-ResNet-8 (DP-FL):** Similar to the previous case study, but with ResNet-8 as the model.

Federated training. We employ five different values for learning rate tuning in the federated case studies: η={0.005, 0.01, 0.025, 0.05, 0.1}. The KernelNorm based models are trained for 400 and 1000 communication rounds in the CIFAR-10 and CIFAR-100 case studies, respectively. The number of rounds for the NoNorm, LayerNorm, and GroupNorm based models is as twice as the kernel normalized counterparts due to their slower convergence rate. The group size is the default value of 32 for the GroupNorm layer [20]. The dropout probability for KNConv and KernelNorm layers are 0.1 and 0.5, respectively. The loss function is cross-entropy, optimizer is SGD with momentum of zero, and training algorithm is FedAvg with number of local epochs of 1.

Differentially private training. We set ε=6.0 and $\delta = 10^{-5}$ for all DP case studies. Regarding parameter tuning, we use learning rate values of η={1.0, 1.5, 2.0} and clipping values of C={1.0, 1.5, 2.0}. The ResNet-8, DenseNet-20×16, and PreactResNet-18 models are trained for 50, 70, and 70 epochs, respectively. The learning rate is divided by 2 at epochs (T-30) and (T-10), where T is the number of epochs (i.e. 50 or 70). The group size of GroupNorm is 16 for DenseNet-20×16, but 32 for the other models. Notice that we cannot set group size to 32 for DenseNet-20×16 because the number of channels must be divisible by the group size. The dropout probability is 0.1 for all KNConv layers in the kernel normalized models. For ResNet-8, the dropout probability of KernelNorm is 0.25, whereas it is 0.5 for DenseNet-20×16 and PreactResNet-18.

We employ cross-entropy as loss function, zero-momentum SGD as optimizer, and the Opacus library (version 1.1) [41] for model training. We observe that changing the kernel size of the shortcut connections in PreactResNet-18 from 1×1 to 2×2 slightly enhances the accuracy of the kernel normalized model, but provides no accuracy gain for the competitors. Thus, the aforementioned kernel size remains 1×1 for NoNorm, LayerNorm, and GroupNorm, whereas it is 2×2 for KernelNorm.

TABLE I: **Federated learning**: Test accuracy for different normalization layers; NoNorm (no normalization) slightly outperforms LayerNorm and GroupNorm in (a); KernelNorm delivers higher accuracy than the competitors; B: batch size.

(a) CIFAR-10-VGG-6 **(cross-silo FL)**

B	NoNorm	LayerNorm	GroupNorm	KernelNorm
16	80.19±0.29	78.93±0.43	78.63±0.56	**83.64**±0.41
64	79.23±0.31	78.97±0.36	79.4±0.38	**82.13**±0.25

(b) CIFAR-10-VGG-6 **(cross-device FL)**

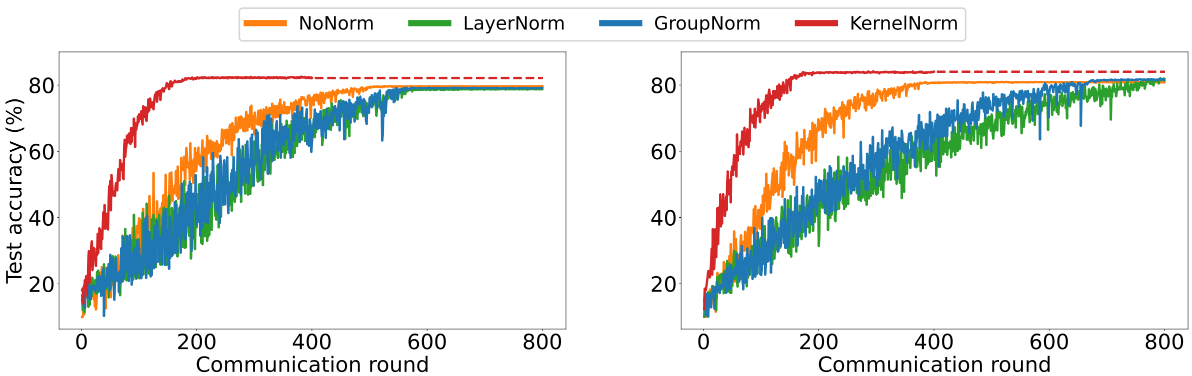
B	NoNorm	LayerNorm	GroupNorm	KernelNorm
16	80.95±0.27	81.89±0.32	81.39±0.47	**84.13**±0.26
64	80.72±0.06	81.43±0.19	81.44±0.18	**83.77**±0.11

(c) CIFAR-100-PreactResNet-18 **(cross-silo FL)**

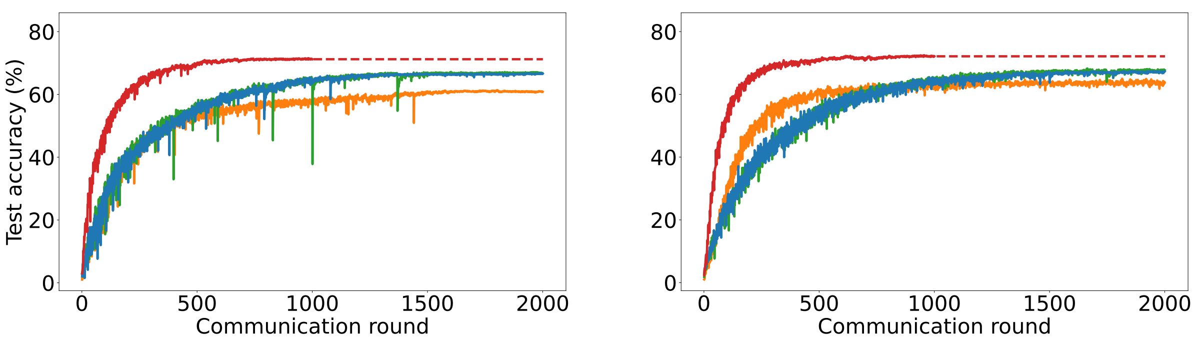
B	NoNorm	LayerNorm	GroupNorm	KernelNorm
16	61.89±0.13	68.16±0.44	67.86±0.1	**71.72**±0.19
64	60.8±0.33	66.9±0.41	66.45±0.18	**71.29**±0.21

(d) CIFAR-100-PreactResNet-18 **(cross-device FL)**

B	NoNorm	LayerNorm	GroupNorm	KernelNorm
16	63.54±0.22	68.05±0.92	68.23±0.13	**71.75**±0.24
64	63.33±0.36	67.84±0.43	67.47±0.24	**71.99**±0.09



(a) CIFAR-10-VGG-6 **(cross-silo FL)** (b) CIFAR-10-VGG-6 **(cross-device FL)**



(c) CIFAR-100-PreactResNet-18 **(cross-silo FL)** (d) CIFAR-100-PreactResNet-18 **(cross-device FL)**

Fig. 1: **Federated learning:** Communication efficiency for various normalization layers; KernelNorm provides significantly higher communication efficiency than the competitors. Surprisingly, NoNorm outperforms both LayerNorm and GroupNorm in terms of communication efficiency in most cases, i.e (a), (b), (d); batch size is 64.

Differentially private federated training. We set ε=8.0 and δ=10^{-5} for both DP-FL case studies. We leverage learning rate values of η={0.01, 0.025, 0.05} and clipping values of C={1.0, 1.5, 2.0} for parameter tuning. The group size of GroupNorm is 32, and the dropout probabilities of the KNConv and KernelNorm layers are 0.1 and 0.25, respectively. The models are trained for 100 communication rounds with a fixed learning rate. The loss function, optimizer, and training algorithm are cross-entropy, SGD with momentum of zero, and FedAvg with number of local epochs of 1, respectively.

B. Results

For all case studies, we first determine the optimal learning rate (and clipping value) based on the model accuracy on the test dataset (see Appendix). We repeat the experiment achieving the highest accuracy three times and report mean/median/mean and the standard deviation of the runs for the FL/DP/DP-FL case studies. We consider the average over the last 10 communication rounds, final accuracy, and the average over the last 3 rounds as the representative accuracy of the run in the FL, DP, and DP-FL settings, respectively.

TABLE II: **Differential privacy**: Test accuracy for various normalization layers; NoNorm (no normalization) delivers slightly higher accuracy than LayerNorm and GroupNorm in (a); KernelNorm considerably outperforms the competitors; ε=6.0, δ=10^{-5}.

(a) CIFAR-10-ResNet-8 (DP)

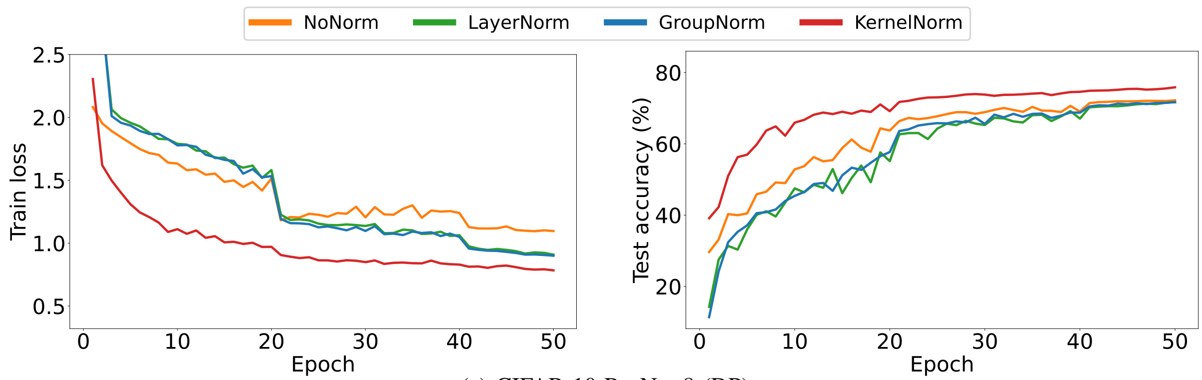
B	NoNorm	LayerNorm	GroupNorm	KernelNorm
512	65.11±0.29	70.01±0.19	70.27±0.08	**72.18**±0.15
1024	69.05±0.4	71.38±0.5	71.75±0.45	**74.31**±0.14
2048	72.7±0.25	71.67±0.42	71.73±0.31	**75.46**±0.34
3072	71.99±0.14	69.39±0.27	68.99±0.27	**75.48**±0.24

(b) CIFAR-10-DenseNet-20×16 (DP)

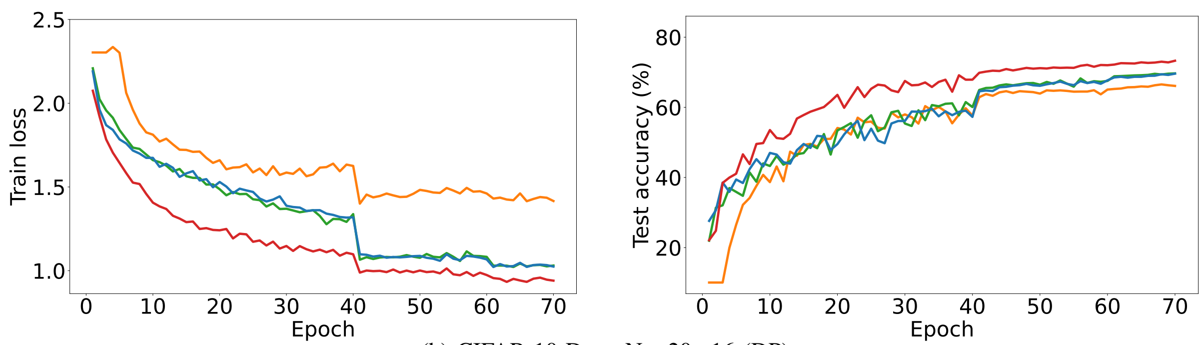
B	NoNorm	LayerNorm	GroupNorm	KernelNorm
256	57.03±0.48	65.62±0.7	66.16±0.56	**68.49**±0.24
512	64.15±0.74	69.24±0.68	68.72±0.65	**70.86**±0.44
1024	64.98±0.6	69.68±0.8	69.57±0.97	**72.74**±0.34
2048	65.29±0.53	66.66±0.78	67.31±0.26	**72.49**±0.39

(c) Imagenette-PreactResNet-18 (DP)

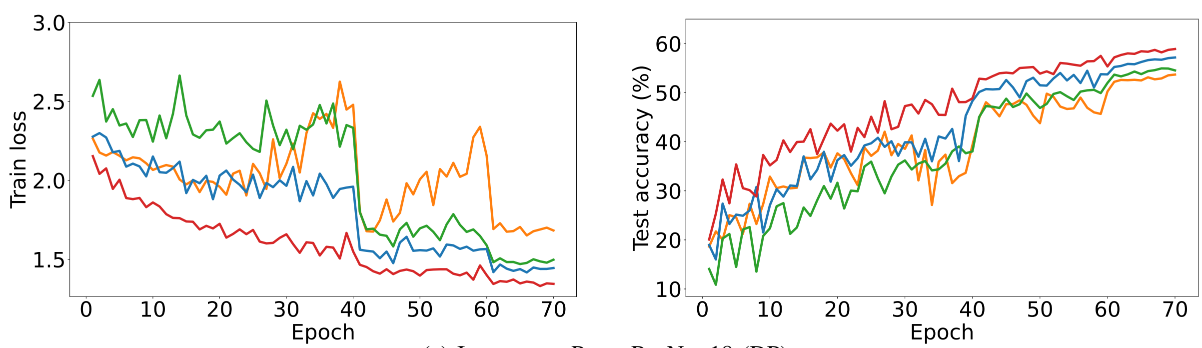
B	NoNorm	LayerNorm	GroupNorm	KernelNorm
512	25.27±3.95	54.83±0.65	56.7±0.19	**59.1**±0.33
1024	53.69±0.83	54.54±0.23	57.17±0.42	**58.9**±0.42
2048	53.53±0.99	53.3±0.32	54.59±0.27	**56.11**±0.26



(a) CIFAR-10-ResNet-8 (DP)



(b) CIFAR-10-DenseNet-20×16 (DP)



(c) Imagenette-PreactResNet-18 (DP)

Fig. 2: **Differential privacy**: Convergence rate for different normalization layers; kernel normalized models provides much faster convergence rate than the competitors; batch size is 2048, 1024, and 1024 for (a), (b), and (c), respectively.

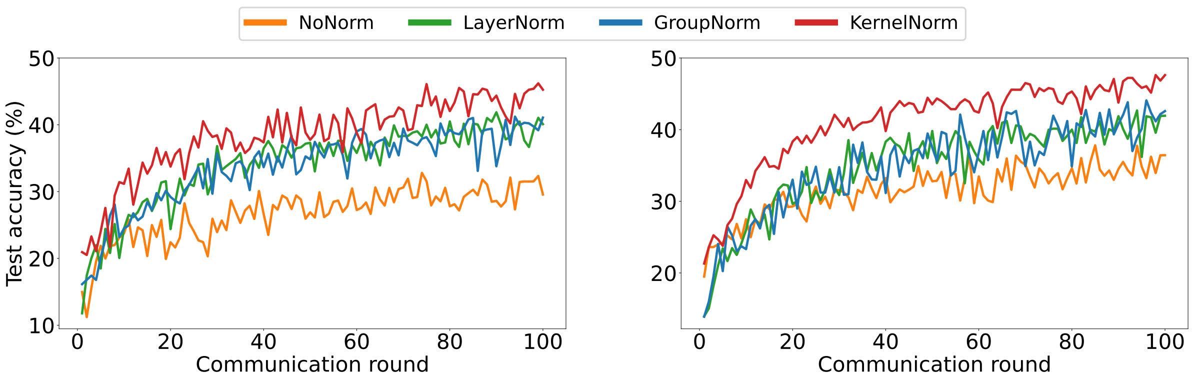
TABLE III: **Differentially private federated learning**: Test accuracy for different normalization layers; KernelNorm delivers considerably higher accuracy than the competitors; ε=8.0, δ=10^{-5}.

(a) CIFAR-10-VGG-6 (DP-FL)

B	NoNorm	LayerNorm	GroupNorm	KernelNorm
256	30.5±0.44	38.23±0.37	37.29±0.71	**46.79**±0.81
512	29.73±1.01	39.47±0.48	39.75±0.65	**45.37**±0.22
1024	33.43±1.33	39.19±0.64	38.85±0.97	**47.11**±0.37

(b) CIFAR-10-ResNet-8 (DP-FL)

B	NoNorm	LayerNorm	GroupNorm	KernelNorm
256	34.76±0.95	38.43±1.48	40.69±1.03	**45.18**±0.34
512	36.11±0.7	41.09±0.33	41.8±0.41	**46.75**±0.48
1024	38.19±0.19	41.41±1.08	41.39±0.82	**48.45**±1.09



(a) CIFAR-10-VGG-6 (DP-FL)

(b) CIFAR-10-ResNet-8 (DP-FL)

Fig. 3: **Differentially private federated learning**: Convergence rate for various normalization layers; kernel normalized models deliver higher convergence rate than the competitors; batch size is 512.

Federated learning. Table I lists the test accuracy values for the FL case studies. According to the table, (1) NoNorm slightly outperforms LayerNorm and GroupNorm in the CIFAR-10-VGG-6 (cross-silo FL) case study, whereas LayerNorm and GroupNorm deliver higher accuracy compared to NoNorm in the other case studies; (2) KernelNorm achieves considerably higher accuracy than the competitors. Fig. 1 illustrates the communication efficiency (i.e. accuracy versus communication round) for the FL case studies. As shown in the figure, (1) NoNorm, surprisingly, provides higher communication efficiency than LayerNorm and GroupNorm for most case studies; (2) KernelNorm achieves remarkably higher communication efficiency compared with NoNorm, LayerNorm, and GroupNorm.

Differential privacy. Table II and Fig. 2 demonstrate the test accuracy and convergence rate of different normalization layers for the DP case studies, respectively. According to the table and figure, (1) NoNorm slightly outperforms LayerNorm and GroupNorm in terms of accuracy in the CIFAR-10-ResNet-8 (DP) case study, but LayerNorm and GroupNorm achieve higher accuracy compared to NoNorm in the other case studies, (2) KernelNorm provides higher accuracy than the competitors in all DP case studies, and (3) KernelNorm based models converge much faster than those based on NoNorm, LayerNorm, and GroupNorm.

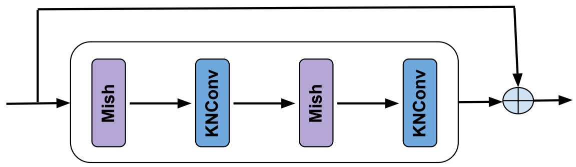
Differentially private federated learning. Table III lists the test accuracy values, and Fig. 3 illustrates the convergence rate of different normalization layers for the DP-FL case studies. As shown in the table and figure, (1) the NoNorm based models deliver much lower accuracy and slower convergence rate than LayerNorm, GroupNorm, and KernelNorm based ones, and (2) the kernel normalized models achieve considerably higher accuracy and faster convergence rate than the competitors.

C. *Findings*

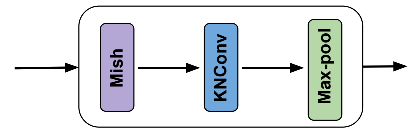
Based on our experimental evaluation, (I) LayerNorm and GroupNorm do not necessarily outperform NoNorm in shallow networks such as VGG-6/ResNet-8 under the FL/DP settings. However, they achieve significant accuracy gain compared to NoNorm for deeper models (e.g. DenseNet-20×16 and PreactResNet-18) in FL and DP as well as shallow models in DP-FL, and (II) KernelNorm delivers remarkably higher accuracy and convergence rate (communication efficiency) than NoNorm, LayerNorm, and GroupNorm with both shallow and deeper networks trained in FL (cross-silo and cross-device) and DP as well as shallow models in DP-FL. Therefore, KernelNorm is the most effective normalization method for FL, DP, and DP-FL settings.

IV. Kernel Normalized ResNet-13

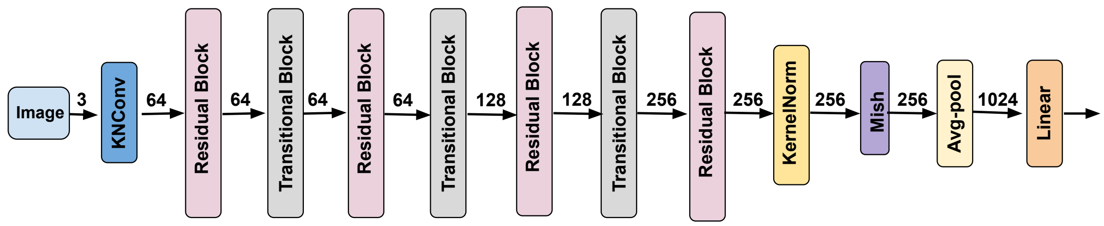
The experimental results from the previous section indicate KernelNorm outperforms the competitors in the DP setting using models that originally designed based on global normalization layers such as BatchNorm (e.g. PreactResNets or DenseNets). The existing architectures, however, are not necessarily optimal for KernelNorm. For instance, the kernel size of 1×1 in the shortcut connections of the ResNet architecture is not beneficial for KernelNorm, which requires kernel sizes greater than 1 to benefit from the spatial correlation of the elements during normalization.



(a) Residual block



(b) Transitional block



(c) KNResNet-13 architecture

Fig. 4: **KNResNet-13 architecture** consists of kernel normalized residual and transitional blocks. The kernel size, stride, and padding of the KNConv layers are 3×3, 1×1, and 1×1, respectively. The kernel size of max-pooling is 2×2. The dropout probability of KNConv and KernelNorm are 0.1 and 0.5, respectively. For medium-resolution images, the first KNConv layer is replaced by a KNConv layer with kernel size 7×7, stride 2×2, and padding 3×3, followed by a Mish activation and 2×2 max-pooling layer. The numbers indicate the input/output channels (filters) of KNConv or neurons of the linear layer.

Given that, we propose a bespoke ResNet architecture for KernelNorm (Fig. 4) to improve the SOTA accuracy values on the CIFAR-10 and Imagenette datasets in differentially private learning settings. We refer to the proposed architecture as *KNResNet-13*, which includes twelve kernel normalized convolutional layers and a final classification (linear) layer.

The convolutional blocks in KNResNet-13 are either residual (Fig. 4a) or transitional (Fig. 4b). The residual blocks contain two KNConv layers with the same number of input and output channels. The transitional blocks include a KNConv and max-pooling layer, aiming to downsample the input. All KNConv layers have kernel size 3×3, stride 1×1, padding 1×1, and dropout probability 0.1. The kernel size of the max-pooling layers is 2×2. The architecture employs Mish as the activation function. The last residual block is followed by a KernelNorm layer with dropout probability 0.5, Mish activation, $2{\times}2$ adaptive average-pooling, and linear layer with 1024 neurons. For medium-resolution images (e.g. 224×224), the first KNConv layer is replaced by a 7×7 KNConv layer followed by the Mish activation and 2×2 max-pooling layer.

In the following, we describe the data preprocessing and differentially private training procedure for the CIFAR-10 and Imagenette datasets. Then, we provide the accuracy values achieved by the KNResNet-13 model and compare them with those from the recent studies.

CIFAR-10. The only data preprocessing step is to divide the feature values by 255. KNResNet-13 is trained for T = 50, 70, 70, and 80 epochs with batch sizes of B=4096, 4096, 3072, and 3072 for ε=2.0, 4.0, 6.0, and 8.0, respectively. The learning rate is 2.0, clipping value is 1.5, and δ is 10^{-5}. The learning rate is divided by 2 at epochs (T - 30) and (T - 10). The optimizer is SGD with momentum of zero.

CIFAR-10 with augmentation multiplicity. The augmentation multiplicity is a recently proposed technique by *De et al.* [23], which computes the gradients for a given sample by taking average over the gradients computed for different augmentations of the same sample. For the CIFAR-10 dataset, this technique applies the sequence of random horizontal flipping and random cropping of size 32×32 and padding 4×4 to obtain an augmented version of a given sample. Here, we employ a slightly different way of augmentation multiplicity because the original technique provides negligible accuracy gain for our model. We first compute the gradients for the original sample, horizontally flipped (i.e. with probability of 1.0), and randomly cropped version of the sample, and then take the average over them to calculate the per-sample gradients. For ε=2.0, 4.0, 6.0, and 8.0, KNResNet-13 is trained for 80, 80, 100, and 100 epochs, respectively. The other training details are the same as CIFAR-10 with no augmentation multiplicity (previous paragraph).

Imagenette. We adopt the 320-pixel version of the dataset and resize the images to 224×224. We train KNResNet-13 with η=1.5, C=1.5, ε=7.0, δ=10^{-5}, and zero-momentum SGD for 100 epochs, where η is divided by 2 at epochs 70 and 90.

Results. Table IV lists the test accuracy values from KNResNet-13 and the recent studies on CIFAR-10, CIFAR-10 with augmentation multiplicity, and Imagenette. KNResNet-13 delivers significantly higher accuracy than the models based on GroupNorm or NoNorm for all considered ε values on CIFAR-10 without augmentation multiplicity. Compared to kernel normalized ResNet-8 [21], KNResNet-13 provides up to 2% accuracy gain depending on the ε value.

On CIFAR-10 with augmentation multiplicity, KNResNet-13 outperforms both wide ResNet-16-4 and ResNet-40-4 [43]

TABLE IV: **Differential privacy**: Comparison of the test accuracy values from the proposed KNResNet-13 architecture with those from the recent studies; δ=10^{-5}.

(a) CIFAR-10

Study	Model	Normalization	ε	Test accuracy
Klause et al. (2022) [24]	ResNet-9	GroupNorm	9.88	73.0
Nasirigerdeh et al. (2022) [21]	ResNet-8	KernelNorm	8.0	76.66
Ours	KNResNet-13	KernelNorm	8.0	**78.51**$\pm$0.35
Dörmann et al. (2021) [42]	VGG-8	NoNorm	7.42	70.1
Klause et al. (2022) [24]	ResNet-9	GroupNorm	7.42	71.8
Remerscheid et al. (2022) [25]	DenseNet-14	GroupNorm	7.0	73.5
Nasirigerdeh et al. (2022)	ResNet-8	KernelNorm	6.0	75.46
Ours	KNResNet-13	KernelNorm	6.0	**77.09**$\pm$0.31
Dörmann et al. (2021) [42]	VGG-8	NoNorm	4.21	66.2
Nasirigerdeh et al. (2022)	ResNet-8	KernelNorm	4.0	73.32
Ours	KNResNet-13	KernelNorm	4.0	**74.51**$\pm$0.19
Klause et al. (2022) [24]	ResNet-9	GroupNorm	2.89	65.6
Nasirigerdeh et al. (2022)	ResNet-8	KernelNorm	2.0	**68.08**
Ours	KNResNet-13	KernelNorm	2.0	**68.05**$\pm$0.07

(b) CIFAR-10 with augmentation multiplicity (K)

Study	Model	Normalization	K	ε	Test accuracy
De et al. (2022) [23]	Wide ResNet-16-4	GroupNorm	16	8.0	79.5
De et al. (2022) [23]	Wide ResNet-40-4	GroupNorm	32	8.0	**81.4**
Ours	KNResNet-13	KernelNorm	3	8.0	80.8 $\pm$0.22
De et al. (2022) [23]	Wide ResNet-16-4	GroupNorm	16	6.0	77.0
De et al. (2022) [23]	Wide ResNet-40-4	GroupNorm	32	6.0	78.8
Ours	KNResNet-13	KernelNorm	3	6.0	**79.09**$\pm$0.07
De et al. (2022) [23]	Wide ResNet-16-4	GroupNorm	16	4.0	71.9
De et al. (2022) [23]	Wide ResNet-40-4	GroupNorm	32	4.0	73.5
Ours	KNResNet-13	KernelNorm	3	4.0	**76.19**$\pm$0.04
De et al. (2022) [23]	Wide ResNet-16-4	GroupNorm	16	2.0	64.9
De et al. (2022) [23]	Wide ResNet-40-4	GroupNorm	32	2.0	65.9
Ours	KNResNet-13	KernelNorm	3	2.0	**70.57**$\pm$0.24

(c) Imagenette

Study	Model	Normalization	ε	Test accuracy
Klause et al. (2022) [24]	ResNet-9	GroupNorm	7.42	64.8
Klause et al. (2022) [24]	ResNet-9	GroupNorm	9.88	67.1
Remerscheid et al. (2022) [25]	DenseNet-14	GroupNorm	7.0	69.7
Ours	KNResNet-13	KernelNorm	7.0	**72.24**$\pm$0.48

with much lower augmentation multiplicity (3 vs. 16 vs. 32) for ε values of 2.0, 4.0, and 6.0. On Imagenette, KNResNet-13 achieves around 3% and 7% higher accuracy than GroupNorm based DenseNet-14 [25] and ResNet-9 [24], respectively.

Given the results from Table IV, we provide new SOTA accuracy values on the CIFAR-10 and Imagenette datasets, when trained from scratch:

- On CIFAR-10 *without* augmentation multiplicity, the accuracy values of 74.51%, 77.09%, and 78.51% for ε=4.0, 6.0, and 8.0, respectively.
- On CIFAR-10 *with* augmentation multiplicity, the accuracy values of 70.57%, 76.19%, and 79.09% for ε=2.0, 4.0, and 6.0, respectively.
- On Imagenette, the accuracy value of 72.24% for ε=7.0.

V. Discussion

Our experimental evaluation shows KernelNorm delivers higher performance than LayerNorm and GroupNorm in FL, DP, and DP-FL. This can be because KernelNorm is a local normalization method, taking into account the spatial correlation of the elements in the height and width dimensions during normalization. This leads to faster convergence rate compared to global batch-independent layers including LayerNorm and GroupNorm, likely due to the smoother optimization landscape [24]. It implies KernelNorm requires less amount of total injected noise to achieve a target accuracy value for a given privacy budget in DP, and a fewer number of communication rounds, and thus, higher communication efficiency in FL.

Moreover, LayerNorm and GroupNorm have scale and shift as learnable parameters. In FL these parameters are aggregated, while they are perturbed with noise in DP. The performance of the layer and group normalized models can negatively be impacted in both cases. KernelNorm, however, is free of these learnable parameters, which can be another factor in superior performance of KernelNorm compared to LayerNorm and GroupNorm.

Finally, the feature values are not required to be normalized with the per-channel mean and standard deviation of the dataset in KernelNorm based models due to self-normalizing nature of KNConv, which normalizes the input before computing convolution. This is beneficial, especially in federated environments, because it is not required for clients to share the mean and standard deviation of their local datasets with server to compute the corresponding global values.

Given the aforementioned properties and its superior performance, KernelNorm has a great potential to become the standard normalization layer for federated learning, differential privacy, and differentially private federated learning.

VI. Related Work

There are few studies that compare the performance of various normalization layers in federated settings. *Hsieh et al.* [9] experimentally show GroupNorm delivers higher accuracy than BatchNorm in supervised FL. *Zhang et al.* [22] demonstrate this also holds for semi-supervised FL. However, these studies have not compared GroupNorm with NoNorm as the baseline. Our experiments illustrate GroupNorm does not necessarily provide accuracy gain compared to NoNorm for shallow models in supervised federated settings.

Several studies investigate the performance of different batch-independent normalization layers for differentially private learning. *Klause et al.* [24] and *Remerscheid et al.* [25] show GroupNorm outperforms LayerNorm in terms of accuracy in DP settings. *Nasirigerdeh et al.* [21] illustrate KernelNorm delivers considerable accuracy gain compared to both LayerNorm and GroupNorm in DP. These prior works, however, do not consider NoNorm as the baseline for comparison. Our evaluation indicates NoNorm slightly outperforms both LayerNorm and GroupNorm for the shallow ResNet-8 model on CIFAR-10, whereas KernelNorm still provides significant accuracy improvement compared to NoNorm for the aforementioned setting. The experimental evaluation of *Nasirigerdeh et al.* [21], moreover, is limited to a single case study. We conduct more extensive experiments with deeper models on both low-resolution and medium-resolution datasets to draw the performance comparisons among NoNorm, LayerNorm, GroupNorm, and KernelNorm.

Some studies propose novel architectures or data augmentation techniques to enhance the accuracy of differentially private models. *Klause et al.* [24] present a 9-layer ResNet architecture in which an additional normalization is performed after the addition operation of the residual block, and show their architecture improves the accuracy compared to the original ResNet architecture. *Remerscheid et al.* [25] introduce a novel DenseNet-based architecture called SmoothNet, which employs 3×3 convolutional layers with a high number of filters in the DenseNet blocks, and demonstrate it outperforms the previous ones in terms of accuracy. Both architectures employ GroupNorm as their normalization layer. We propose the KNResNet-13 architecture based on KernelNorm, and show it delivers considerably higher accuracy than the aforementioned architectures on CIFAR-10 and Imagenette.

De et al. [23] present the augmentation multiplicity technique, which computes the per-sample gradients by taking average over the gradients from different augmentations of the sample. We adopt this technique to train the proposed KNResNet-13 architecture on CIFAR-10. The accuracy from KNResNet-13 is higher than the wide ResNet-16-4 and ResNet-40-4 used in [23] for ε values of 2.0, 4.0, and 6.0.

VII. Conclusion and Future Work

We address the normalization challenge in the context of federated and differentially private learning. Through extensive experiments, we demonstrate: (1) in FL and DP, using no normalization layer in the architecture of shallow networks such as VGG-6 and ResNet-8 delivers slightly higher accuracy than LayerNorm and GroupNorm, (2) on deeper models such as DenseNet-20×16 and PreactResNet-18 in FL and DP as well as the shallow models in DP-FL, however, LayerNorm and GroupNorm considerably outperform NoNorm, and (3) the recently proposed KernelNorm method achieves significantly higher accuracy and convergence rate compared to NoNorm, LayerNorm, and GroupNorm in FL, DP, and DP-FL.

Given the superior performance of KernelNorm, we propose a kernel normalized ResNet architecture called KNResNet-13 for differentially private learning. Using the proposed architecture, we provide new SOTA accuracy values on CIFAR-10 with and without augmentation multiplicity as well as Imagenette for different ε values, when trained from scratch.

We employ a low augmentation multiplicity value (i.e. 3) in our study due to the remarkable computational overhead of the technique. KNResNet-13 might deliver even higher accuracy with larger augmentation multiplicity values (e.g. 16 or 32), which can be an investigated in future studies. Additionally, the performance evaluation of kernel normalized architectures on the large Imagenet-32×32 dataset [36] is an interesting direction for future works.

REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[2] Eric Horvitz and Deirdre Mulligan. Data, privacy, and the greater good. *Science*, 349(6245):253–255, 2015.

[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] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE symposium on security and privacy (SP)*, pages 739–753. IEEE, 2019.

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

[6] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st computer security foundations symposium (CSF)*, pages 268–282. IEEE, 2018.

[7] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[8] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[9] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip Gibbons. The non-iid data quagmire of decentralized machine learning. In *International Conference on Machine Learning*, pages 4387–4398. PMLR, 2020.

[10] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 691–706. IEEE, 2019.

[11] Ligeng Zhu and Song Han. Deep leakage from gradients. In *Federated Learning*, pages 17–31. Springer, 2020.

[12] Dmitrii Usynin, Daniel Rueckert, Jonathan Passerat-Palmbach, and Georgios Kaissis. Zen and the art of model adaptation: Low-utility-cost attack mitigations in collaborative machine learning. *Proceedings on Privacy Enhancing Technologies*, 2022(1):274–290, 2022.

[13] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9:211–407, 2014.

[14] 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.

[15] Mário S Alvim, Miguel E Andrés, Konstantinos Chatzikokolakis, Pierpaolo Degano, and Catuscia Palamidessi. Differential privacy: on the trade-off between utility and information leakage. In *International Workshop on Formal Aspects in Security and Trust*, pages 39–54. Springer, 2011.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

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

[18] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[19] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[20] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[21] Reza Nasirigerdeh, Reihaneh Torkzadehmahani, Daniel Rueckert, and Georgios Kaissis. Kernel normalized convolutional networks. *arXiv preprint arXiv:2205.10089*, 2022.

[22] Zhengming Zhang, Zhewei Yao, Yaoqing Yang, Yujun Yan, Joseph E Gonzalez, and Michael W Mahoney. Benchmarking semi-supervised federated learning. *arXiv preprint arXiv:2008.11364*, 17:3, 2020.

[23] 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.

[24] Helena Klause, Alexander Ziller, Daniel Rueckert, Kerstin Hammernik, and Georgios Kaissis. Differentially private training of residual networks with scale normalisation. *arXiv preprint arXiv:2203.00324*, 2022.

[25] Nicolas W Remerscheid, Alexander Ziller, Daniel Rueckert, and Georgios Kaissis. Smoothnets: Optimizing cnn architecture design for differentially private deep learning. *arXiv preprint arXiv:2205.04095*, 2022.

[26] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[27] Dae Young Park, Moon-Hyun Cha, Daesin Kim, Bohyung Han, et al. Learning student-friendly teacher networks for knowledge distillation. *Advances in Neural Information Processing Systems*, 34, 2021.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[29] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[30] Jeremy Howard. https://github.com/fastai/imagenette/.

[31] Cynthia Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.

[32] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *Advances in neural information processing systems*, 31, 2018.

[33] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[34] Anthony Ortiz, Caleb Robinson, Dan Morris, Olac Fuentes, Christopher Kiekintveld, Md Mahmudulla Hassan, and Nebojsa Jojic. Local context normalization: Revisiting local normalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11276–11285, 2020.

[35] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[36] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[37] Liu Kuang. https://github.com/kuangliu/pytorch-cifar/.

[38] Andreas Veit. https://github.com/andreasveit/densenet-pytorch.

[39] Diganta Misra. Mish: A self regularized non-monotonic activation function. *arXiv preprint arXiv:1908.08681*, 2019.

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[41] 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.

[42] Friedrich Dörmann, Osvald Frisk, Lars Nørvang Andersen, and Christian Fischer Pedersen. Not all noise is accounted equally: How differentially private learning benefits from large sampling rates. In *2021 IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2021.

[43] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference 2016*. British Machine Vision Association, 2016.

Appendix

TABLE V: **Federated learning**: Learning rate values giving the highest accuracy for each normalization layer; B: batch size.

(a) CIFAR-10-VGG-6 **(cross-silo FL)**

B	NoNorm	LayerNorm	GroupNorm	KernelNorm
16	0.025	0.025	0.01	0.025
64	0.025	0.025	0.05	0.025

(b) CIFAR-10-VGG-6 **(cross-device FL)**

B	NoNorm	LayerNorm	GroupNorm	KernelNorm
16	0.025	0.025	0.05	0.025
64	0.05	0.025	0.05	0.05

(c) CIFAR-100-PreactResNet-18 **(cross-silo FL)**

B	NoNorm	LayerNorm	GroupNorm	KernelNorm
16	0.01	0.01	0.005	0.025
64	0.01	0.01	0.01	0.05

(d) CIFAR-100-PreactResNet-18 **(cross-device FL)**

B	NoNorm	LayerNorm	GroupNorm	KernelNorm
16	0.01	0.01	0.005	0.025
64	0.05	0.01	0.01	0.1

TABLE VI: **Differential privacy**: Learning rate values giving the highest accuracy for each normalization layer; B: batch size.

(a) CIFAR-10-ResNet-8 (DP)

B	NoNorm	LayerNorm	GroupNorm	KernelNorm
512	1.0	1.0	1.0	1.0
1024	2.0	2.0	1.5	1.5
2048	2.0	2.0	2.0	2.0
3072	2.0	2.0	2.0	2.0

(b) CIFAR-10-DenseNet-20×16 (DP)

B	NoNorm	LayerNorm	GroupNorm	KernelNorm
256	1.0	1.5	2.0	1.5
512	1.0	2.0	2.0	1.5
1024	1.5	2.0	1.5	1.5
2048	2.0	2.0	2.0	1.5

(c) Imagenette-PreactResNet-18 (DP)

B	NoNorm	LayerNorm	GroupNorm	KernelNorm
512	1.0	1.0	1.0	1.5
1024	1.0	1.0	1.0	2.0
2048	1.5	1.0	1.0	2.0

TABLE VII: **Differential privacy**: Clipping values giving the highest accuracy for each normalization layer; B: batch size.

(a) CIFAR-10-ResNet-8 (DP)

B	NoNorm	LayerNorm	GroupNorm	KernelNorm
512	1.0	1.0	1.0	1.0
1024	1.0	1.5	2.0	1.5
2048	2.0	2.0	2.0	2.0
3072	2.0	2.0	2.0	2.0

(b) CIFAR-10-DenseNet-20×16 (DP)

B	NoNorm	LayerNorm	GroupNorm	KernelNorm
256	1.0	1.5	2.0	1.5
512	1.0	1.5	1.5	1.5
1024	2.0	2.0	2.0	1.5
2048	2.0	1.5	2.0	1.0

(c) Imagenette-PreactResNet-18 (DP)

B	NoNorm	LayerNorm	GroupNorm	KernelNorm
512	1.0	1.0	1.0	1.5
1024	1.0	1.5	1.0	1.0
2048	1.0	1.0	1.0	1.0

TABLE VIII: **Differentially private federated learning**: Learning rates giving the highest accuracy for each norm layer.

(a) CIFAR-10-VGG-6 (DP-FL)

B	NoNorm	LayerNorm	GroupNorm	KernelNorm
256	0.01	0.01	0.01	0.01
512	0.025	0.01	0.01	0.025
1024	0.025	0.01	0.025	0.025

(b) CIFAR-10-ResNet-8 (DP-FL)

B	NoNorm	LayerNorm	GroupNorm	KernelNorm
256	0.01	0.01	0.01	0.01
512	0.025	0.01	0.01	0.01
1024	0.025	0.01	0.01	0.05

TABLE IX: **Differentially private federated learning**: Clipping values giving the highest accuracy for each norm layer.

(a) CIFAR-10-VGG-6 (DP-FL)

B	NoNorm	LayerNorm	GroupNorm	KernelNorm
256	1.0	1.0	1.5	1.0
512	1.5	1.0	1.0	1.0
1024	2.0	1.5	2.0	2.0

(b) CIFAR-10-ResNet-8 (DP-FL)

B	NoNorm	LayerNorm	GroupNorm	KernelNorm
256	1.0	1.5	1.0	1.0
512	1.0	1.0	1.0	1.0
1024	1.0	1.0	2.0	2.0

PART III

CONCLUDING REMARKS

Conclusion

This dissertation focuses on enhancing the efficiency of machine learning models including the regression and neural network models in terms of utility, network communication (convergence rate), and/or privacy in federated, differentially private, and differentially private federated learning environments given centralized training as baseline. We present the core contributions of the dissertation in chapters 4-7 in the form of four sole first-authored publications: sPLINK [30], UPFL [31], KernelNorm [32], and KernelNorm for privacy-related domains [33].

In the first study, we introduce a software called sPLINK for GWAS, which implements the hybrid federated versions of the chi-square, linear regression, and logistic regression models. We analytically and experimentally demonstrate that sPLINK provides ideal utility, which is identical to the utility from PLINK [34] on the centralized (aggregated) data, independent of the data distribution across the clients. sPLINK operates in a federated environment, where the private data of the clients is not shared with a third party (privacy-enhancing). sPLINK is also efficient from the communication perspective, requiring a few rounds to calculate the statistics.

In the second study, we theoretically prove and experimentally validate that the DNN models can achieve ideal utility in federated settings akin to the regression models provided that the (1) model and loss function are batch-independent and deterministic, (2) optimizer uses a linear momentum function, and (3) training algorithm selects all clients, the clients perform a single local update per round, and the server employs weighted averaging as aggregation function. We refer to a federated environment satisfying the above-mentioned conditions as UPFL, which preserves utility compared to the corresponding centralized setting. UPFL, however, incurs remarkable communication overhead. In other words, it sacrifices communication efficiency for ideal utility. UPFL is also privacy-enhancing, but not privacy-preserving.

sPLINK and UPFL do not make the training procedure more efficient in terms of communication, utility, and privacy at the same time. That is, they do not break the CUP trade-off similar to many studies in the literature. To address this challenge, we propose a novel normalization layer called KernelNorm in our third study.

KernelNorm is a batch-independent and local normalization layer, which extensively considers the spatial correlation among the elements in the width and height dimensions during normalization. We also introduce KNConv as the combination of the KernelNorm and convolutional layers. We incorporate the proposed KernelNorm and KNConv layers as the main building blocks of KNResNets while forgoing BatchNorm. Through extensive experiments, we show that KNResNets provide higher or very competitive accuracy compared to BatchNorm-based counterparts, and significantly outperform the batch-independent competitors including layer and group normalized ResNets for image classification and semantic segmentation in centralized settings. We also demonstrate that KNResNet-18 achieves higher accuracy than LayerNorm and GroupNorm based ResNet-18 in differentially private learning.

In the last study, we draw an extensive comparison among KernelNorm, LayerNorm, GroupNorm, and NoNorm (no normalization) using the VGG, ResNet, and DenseNet models in FL, DP, and DP-FL environments. Our results indicate that the KernelNorm based models considerably outperform the competitors in all three environments, and as a result, KernelNorm is the most efficient normalization layer for privacy-related domains. We also propose the KNResNet-13 architecture for differentially private training, and provide the state-of-the-art accuracy values on the CIFAR-10 and Imagenette datasets, when trained from scratch.

Finally, we conduct an elegant experiment in a DP-FL environment to illustrate how to break the CUP trade-off using kernel normalized models. Through our experiment, we show that kernel normalized ResNet-9 can deliver higher accuracy with lower privacy budget in fewer communication rounds compared to group normalized ResNet-9, implying that it enhances utility, communication efficiency, and privacy simultaneously, and breaks the CUP trade-off.

Outlook

The sPLINK tool provides ideal utility and high communication efficiency for three popular models in GWAS, i.e. chi-square and linear/logistic regression, in a privacy-enhancing environment, where the private data of clients is not shared with third parties. The current version of sPLINK, however, does not support population stratification using PCA, which is essential in practical GWAS. The *Fever-PCA* tool proposed by *Hartebrodt et al.* [67] addresses this limitation. Given that, combining sPLINK with Fever-PCA to perform practical GWAS is a logical direction for future research. Two main challenges, however, should be taken into account in this regard: (1) Unlike sPLINK, Fever-PCA is not efficient from the network communication perspective, requiring a couple of hundreds of rounds for model convergence, and (2) both sPLINK and Fever-PCA are not privacy-preserving because they do not employ differential privacy during training. The latter challenge is especially of great importance for the GWAS community, which deals with private data of patients.

The proposed KernelNorm and KNConv layers are incorporated into KNConvNets, which are efficient not only in CL but also in FL, DP, and DP-FL settings. The current implementation of the proposed layers, however, is not optimal from the computation aspect. This is because they are implemented using PyTorch [82] primitives but not in CUDA. The implementation of the proposed layers in CUDA is a crucial step towards the widespread adoptability of KNConvNets by the deep learning community.

We illustrate the effectiveness of KNConvNets for the image classification and semantic segmentation tasks. The performance evaluation of KNConvNets for a variety of application domains such as object detection [83], generative adversarial networks (GANs) [84], diffusion generative models [85], and image denoising [86] is an interesting direction for future studies.

The training algorithm, model, loss function, and optimizer are considered as the main DNN training components. In this dissertation, we focus on model, or more precisely, the normalization layer in model to break the CUP trade-off. Future studies can follow orthogonal directions by focusing on the other training components to improve utility, communication, and privacy simultaneously in DP-FL environments.

Bibliography

[1] H. Brink, J. Richards, and M. Fetherolf. *Real-world machine learning*. Simon and Schuster, 2016.

[2] I. H. Sarker. "Machine learning: Algorithms, real-world applications and research directions." In: *SN computer science* 2.3 (2021), p. 160.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems* 25 (2012).

[4] M. Hassaballah and A. I. Awad. *Deep learning in computer vision: principles and applications*. CRC Press, 2020.

[5] A. Graves, A.-r. Mohamed, and G. Hinton. "Speech recognition with deep recurrent neural networks." In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 6645–6649.

[6] D. W. Otter, J. R. Medina, and J. K. Kalita. "A survey of the usages of deep learning for natural language processing." In: *IEEE Transactions on Neural Networks and Learning Systems* 32.2 (2020), pp. 604–624.

[7] OpenAI. *GPT-4 Technical Report*. 2023.

[8] D. Shen, G. Wu, and H.-I. Suk. "Deep learning in medical image analysis." In: *Annual review of biomedical engineering* 19 (2017), pp. 221–248.

[9] K. Suzuki. "Overview of deep learning in medical imaging." In: *Radiological physics and technology* 10.3 (2017), pp. 257–273.

[10] M. Fareed and M. Afzal. "Single nucleotide polymorphism in genome-wide association of human population: A tool for broad spectrum service." In: *Egyptian Journal of Medical Human Genetics* 14.2 (2013), pp. 123–134.

[11] P. M. Visscher, N. R. Wray, Q. Zhang, P. Sklar, M. I. McCarthy, M. A. Brown, and J. Yang. "10 years of GWAS discovery: biology, function, and translation." In: *The American Journal of Human Genetics* 101.1 (2017), pp. 5–22.

[12] R. De, W. Bush, and J. Moore. “Bioinformatics challenges in genome-wide association studies (GWAS).” In: *Methods in Molecular Biology (Clifton, NJ)* 1168 (2014), pp. 63–81.

[13] E. Horvitz and D. Mulligan. “Data, privacy, and the greater good.” In: *Science* 349.6245 (2015), pp. 253–255.

[14] P. Regulation. “General data protection regulation.” In: *Intouch* 25 (2018), pp. 1–5.

[15] G. D. P. Regulation. “General data protection regulation (GDPR).” In: *Intersoft Consulting* 24.1 (2018).

[16] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. “Communication-efficient learning of deep networks from decentralized data.” In: *Artificial intelligence and statistics*. PMLR. 2017, pp. 1273–1282.

[17] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al. “Advances and open problems in federated learning.” In: *arXiv preprint arXiv:1912.04977* (2019).

[18] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith. “Federated learning: Challenges, methods, and future directions.” In: *IEEE Signal Processing Magazine* 37.3 (2020), pp. 50–60.

[19] K. Hsieh, A. Phanishayee, O. Mutlu, and P. Gibbons. “The non-iid data quagmire of decentralized machine learning.” In: *International Conference on Machine Learning*. PMLR. 2020, pp. 4387–4398.

[20] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha. “Privacy risk in machine learning: Analyzing the connection to overfitting.” In: *2018 IEEE 31st computer security foundations symposium (CSF)*. IEEE. 2018, pp. 268–282.

[21] M. Nasr, R. Shokri, and A. Houmansadr. “Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning.” In: *2019 IEEE symposium on security and privacy (SP)*. IEEE. 2019, pp. 739–753.

[22] N. Homer, S. Szelinger, M. Redman, D. Duggan, W. Tembe, J. Muehling, J. V. Pearson, D. A. Stephan, S. F. Nelson, and D. W. Craig. “Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays.” In: *PLoS genetics* 4.8 (2008), e1000167.

[23] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. “Membership inference attacks against machine learning models.” In: *2017 IEEE symposium on security and privacy (SP)*. IEEE. 2017, pp. 3–18.

[24] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov. “Exploiting unintended feature leakage in collaborative learning.” In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2019, pp. 691–706.

[25] L. Zhu and S. Han. “Deep leakage from gradients.” In: *Federated Learning*. Springer, 2020, pp. 17–31.

[26] D. Usynin, D. Rueckert, J. Passerat-Palmbach, and G. Kaissis. “Zen and the art of model adaptation: Low-utility-cost attack mitigations in collaborative machine learning.” In: *Proceedings on Privacy Enhancing Technologies* 2022.1 (2022), pp. 274–290.

[27] C. Dwork and A. Roth. “The Algorithmic Foundations of Differential Privacy.” In: *Found. Trends Theor. Comput. Sci.* 9 (2014), pp. 211–407.

[28] 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*. 2016, pp. 308–318.

[29] M. S. Alvim, M. E. Andrés, K. Chatzikokolakis, P. Degano, and C. Palamidessi. “Differential privacy: on the trade-off between utility and information leakage.” In: *International Workshop on Formal Aspects in Security and Trust*. Springer. 2011, pp. 39–54.

[30] R. Nasirigerdeh, R. Torkzadehmahani, J. Matschinske, T. Frisch, M. List, J. Späth, S. Weiss, U. Völker, E. Pitkänen, D. Heider, et al. “sPLINK: a hybrid federated tool as a robust alternative to meta-analysis in genome-wide association studies.” In: *Genome Biology* 23 (2022), pp. 1–24.

[31] R. Nasirigerdeh, D. Rueckert, and G. Kaissis. “Utility-preserving Federated Learning.” In: *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*. AISec ’23. Copenhagen, Denmark: Association for Computing Machinery, 2023, pp. 55–65.

[32] R. Nasirigerdeh, R. Torkzadehmahani, D. Rueckert, and G. Kaissis. “Kernel Normalized Convolutional Networks.” In: *Transactions on Machine Learning Research* (2024), pp. 1–21.

[33] R. Nasirigerdeh, J. Torkzadehmahani, D. Rueckert, and G. Kaissis. "Kernel Normalized Convolutional Networks for Privacy-Preserving Machine Learning." In: *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*. IEEE. 2023, pp. 107–118.

[34] S. Purcell, B. Neale, K. Todd-Brown, L. Thomas, M. A. Ferreira, D. Bender, J. Maller, P. Sklar, P. I. De Bakker, M. J. Daly, et al. "PLINK: a tool set for whole-genome association and population-based linkage analyses." In: *The American journal of human genetics* 81.3 (2007), pp. 559–575.

[35] R. Cramer, I. B. Damgård, and J. B. Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge: Cambridge University Press, 2015.

[36] S. Ioffe and C. Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[37] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization." In: *3rd International Conference on Learning Representations, ICLR*. 2015.

[38] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith. "Federated Optimization in Heterogeneous Networks." In: *Proceedings of Machine Learning and Systems*. Vol. 2. 2020, pp. 429–450.

[39] S. J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan. "Adaptive Federated Optimization." In: *International Conference on Learning Representations*. 2021.

[40] J. L. Ba, J. R. Kiros, and G. E. Hinton. "Layer normalization." In: *arXiv preprint arXiv:1607.06450* (2016).

[41] Y. Wu and K. He. "Group normalization." In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 3–19.

[42] 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*. Ieee. 2009, pp. 248–255.

[43] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: *International Conference on Learning Representations*. 2015.

[44] 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*. 2017, pp. 4700–4708.

[45] 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*. 2016, pp. 770–778.

[46] K. He, X. Zhang, S. Ren, and J. Sun. "Identity mappings in deep residual networks." In: *European conference on computer vision*. Springer. 2016, pp. 630–645.

[47] A. Krizhevsky, G. Hinton, et al. "Learning multiple layers of features from tiny images." In: (2009).

[48] J. Howard. *imagenette*. 2019.

[49] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation applied to handwritten zip code recognition." In: *Neural computation* 1.4 (1989), pp. 541–551.

[50] J. Long, E. Shelhamer, and T. Darrell. "Fully convolutional networks for semantic segmentation." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[51] A. L. Maas, A. Y. Hannun, A. Y. Ng, et al. "Rectifier nonlinearities improve neural network acoustic models." In: *Proc. icml*. Vol. 30. 1. Atlanta, GA. 2013, p. 3.

[52] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. "Self-normalizing neural networks." In: *Advances in neural information processing systems* 30 (2017).

[53] D. Ulyanov, A. Vedaldi, and V. Lempitsky. "Instance normalization: The missing ingredient for fast stylization." In: *arXiv preprint arXiv:1607.08022* (2016).

[54] M. Tan and Q. Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." In: *International conference on machine learning*. PMLR. 2019, pp. 6105–6114.

[55] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning*. Cambridge: Springer series in statistics, 2009.

[56] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[57] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor. "Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization." In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 7611–7623.

[58] R. Nasirigerdeh, M. Bakhtiari, R. Torkzadehmahani, A. Bayat, M. List, D. B. Blumenthal, and J. Baumbach. “Federated Multi-Mini-Batch: An Efficient Training Approach to Federated Learning in Non-IID Environments.” In: *arXiv preprint arXiv:2011.07006* (2020).

[59] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani. “Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization.” In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 2021–2031.

[60] Y. Mao, Z. Zhao, G. Yan, Y. Liu, T. Lan, L. Song, and W. Ding. “Communication-efficient federated learning with adaptive quantization.” In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 13.4 (2022), pp. 1–26.

[61] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek. “Robust and communication-efficient federated learning from non-iid data.” In: *IEEE transactions on neural networks and learning systems* 31.9 (2019), pp. 3400–3413.

[62] H. Sun, S. Li, F. R. Yu, Q. Qi, J. Wang, and J. Liao. “Toward communication-efficient federated learning in the Internet of Things with edge computing.” In: *IEEE Internet of Things Journal* 7.11 (2020), pp. 11053–11067.

[63] B. Wang, J. Fang, H. Li, and B. Zeng. “Communication-Efficient Federated Learning: A Variance-Reduced Stochastic Approach With Adaptive Sparsification.” In: *IEEE Transactions on Signal Processing* (2023).

[64] R. Torkzadehmahani, R. Nasirigerdeh, D. B. Blumenthal, T. Kacprowski, M. List, J. Matschinske, J. Spaeth, N. K. Wenke, and J. Baumbach. “Privacy-preserving artificial intelligence techniques in biomedicine.” In: *Methods of Information in Medicine* 61 (2022), e12–e27.

[65] R. Nasirigerdeh, R. Torkzadehmahani, J. Matschinske, J. Baumbach, D. Rueckert, and G. Kaissis. “HyFed: A Hybrid Federated Framework for Privacy-preserving Machine Learning.” In: *arXiv preprint arXiv:2105.10545* (2021).

[66] O. Zolotareva*, R. Nasirigerdeh*, J. Matschinske, R. Torkzadehmahani, M. Bakhtiari, T. Frisch, J. Späth, D. B. Blumenthal, A. Abbasinejad, P. Tieri, et al. “Flimma: a federated and privacy-aware tool for differential gene expression analysis.” In: *Genome biology* 22 (2021), pp. 1–26.

[67] A. Hartebrodt, R. Nasirigerdeh, D. B. Blumenthal, and R. Röttger. “Federated principal component analysis for genome-wide association studies.” In: *2021 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2021, pp. 1090–1095.

[68] OpenMined. *PySyft*. 2019.

[69] A. Ziller, A. Trask, A. Lopardo, B. Szymkow, B. Wagner, E. Bluemke, J.-M. Nounahon, J. Passerat-Palmbach, K. Prakash, N. Rose, et al. “Pysyft: A library for easy federated learning.” In: *Federated Learning Systems: Towards Next-Generation AI* (2021), pp. 111–139.

[70] J. Matschinske, J. Späth, M. Bakhtiari, N. Probul, M. M. Kazemi Majdabadi, R. Nasirigerdeh, R. Torkzadehmahani, A. Hartebrodt, B.-A. Orban, S.-J. Fejér, et al. “The FeatureCloud Platform for Federated Learning in Biomedicine: Unified Approach.” In: *Journal of Medical Internet Research* 25 (2023), e42621.

[71] A.-C. Hauschild, M. Lemanczyk, J. Matschinske, T. Frisch, O. Zolotareva, A. Holzinger, J. Baumbach, and D. Heider. “Federated Random Forests can improve local performance of predictive models for various healthcare applications.” In: *Bioinformatics* 38.8 (2022), pp. 2278–2286.

[72] J. Späth, J. Matschinske, F. K. Kamanu, S. A. Murphy, O. Zolotareva, M. Bakhtiari, E. M. Antman, J. Loscalzo, A. Brauneck, L. Schmalhorst, et al. “Privacy-aware multi-institutional time-to-event studies.” In: *PLOS Digital Health* 1.9 (2022), e0000101.

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

[74] N. W. Remerscheid, A. Ziller, D. Rueckert, and G. Kaissis. “SmoothNets: Optimizing CNN architecture design for differentially private deep learning.” In: *arXiv preprint arXiv:2205.04095* (2022).

[75] A. Cheng, J. Wang, X. S. Zhang, Q. Chen, P. Wang, and J. Cheng. “Dpnas: Neural architecture search for deep learning with differential privacy.” In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 6. 2022, pp. 6358–6366.

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

[77] S. Fort, A. Brock, R. Pascanu, S. De, and S. L. Smith. "Drawing multiple augmentation samples per image during training efficiently decreases test error." In: *arXiv preprint arXiv:2105.13343* (2021).

[78] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. Quek, and H. V. Poor. "Federated learning with differential privacy: Algorithms and performance analysis." In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 3454–3469.

[79] G. Kaissis, A. Ziller, J. Passerat-Palmbach, T. Ryffel, D. Usynin, A. Trask, I. Lima Jr, J. Mancuso, F. Jungmann, M.-M. Steinborn, et al. "End-to-end privacy preserving deep learning on multi-institutional medical imaging." In: *Nature Machine Intelligence* 3.6 (2021), pp. 473–484.

[80] M. Noble, A. Bellet, and A. Dieuleveut. "Differentially private federated learning on heterogeneous data." In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2022, pp. 10110–10145.

[81] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh. "Scaffold: Stochastic controlled averaging for federated learning." In: *International conference on machine learning*. PMLR. 2020, pp. 5132–5143.

[82] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[83] C. Szegedy, A. Toshev, and D. Erhan. "Deep neural networks for object detection." In: *Advances in neural information processing systems* 26 (2013).

[84] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative adversarial networks." In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[85] P. Dhariwal and A. Nichol. "Diffusion models beat gans on image synthesis." In: *Advances in neural information processing systems* 34 (2021), pp. 8780–8794.

[86] V. Jain and S. Seung. "Natural image denoising with convolutional networks." In: *Advances in neural information processing systems* 21 (2008).

[87] D. Misra. "Mish: A self regularized non-monotonic activation function." In: *arXiv preprint arXiv:1908.08681* (2019).

PART IV

APPENDICES

Supplementary Material: sPLINK: a Hybrid Federated Tool as a Robust Alternative to Meta-analysis in Genome-wide Association Studies

Experimental details

We used *PLINK* V1.9 to generate the splits and perform the aggregated analysis; SNPs with minor allele frequency below 0.05 were filtered out. The common SNPs among the splits have been considered in all analyses. Tables S1-S3 list the sample size (case | control | total) and the number of SNPs for each split in the aggregated analysis with PLINK, meta-analysis using PLINK, METAL, and GWAMA, and the federated analysis using sPLINK.

Table S1 The SHIP case study

<table>
<tr><th>Association test</th><th colspan="2">Split1</th><th colspan="2">Split2</th><th colspan="2">Split3</th><th colspan="2">Split4</th><th colspan="2">Aggregated</th></tr>
<tr><td></td><td>Sample size</td><td># of SNPs</td><td>Sample size</td><td># of SNPs</td><td>Sample size</td><td># of SNPs</td><td>Sample size</td><td># of common SNPs</td><td>Sample size</td><td># of common SNPs</td></tr>
<tr><td>Chi-square</td><td>229 | 712 | 941</td><td>5070067</td><td>276 | 768 | 1044</td><td>5062964</td><td>245 | 761 | 1006</td><td>5070192</td><td>184 | 524 | 708</td><td>5077381</td><td>934 | 2765 | 3699</td><td>4878280</td></tr>
<tr><td>Logistic regression</td><td>229 | 712 | 941</td><td>5070067</td><td>276 | 768 | 1044</td><td>5062964</td><td>245 | 761 | 1006</td><td>5070192</td><td>184 | 524 | 708</td><td>5077381</td><td>934 | 2765 | 3699</td><td>4878280</td></tr>
<tr><td>Linear regression</td><td>941</td><td>5070067</td><td>1044</td><td>5062964</td><td>1006</td><td>5070192</td><td>708</td><td>5077381</td><td>3699</td><td>4878280</td></tr>
</table>

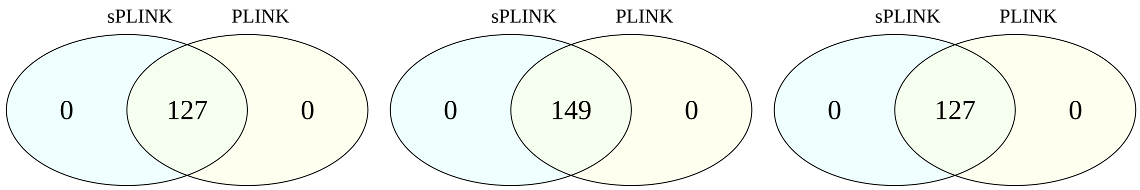
Table S2 The COPDGene case study

<table>
<tr><th>Scenario</th><th colspan="2">Split1</th><th colspan="2">Split2</th><th colspan="2">Split3</th><th colspan="2">Aggregated</th></tr>
<tr><td></td><td>Sample size</td><td># of SNPs</td><td>Sample size</td><td># of SNPs</td><td>Sample size</td><td># of SNPs</td><td>Sample size</td><td># of common SNPs</td></tr>
<tr><td>I</td><td>937 | 844 | 1781</td><td>584910</td><td>937 | 844 | 1781</td><td>584816</td><td>937 | 844 | 1781</td><td>585071</td><td>2811 | 2532 | 5343</td><td>580719</td></tr>
<tr><td>II</td><td>737 | 1044 | 1781</td><td>584928</td><td>937 | 844 | 1781</td><td>585108</td><td>1137 | 644 | 1781</td><td>584816</td><td>2811 | 2532 | 5343</td><td>580743</td></tr>
<tr><td>III</td><td>537 | 1244 | 1781</td><td>584978</td><td>937 | 844 | 1781</td><td>584983</td><td>1337 | 444 | 1781</td><td>584860</td><td>2811 | 2532 | 5343</td><td>580783</td></tr>
<tr><td>IV</td><td>337 | 1444 | 1781</td><td>585105</td><td>937 | 844 | 1781</td><td>584960</td><td>1537 | 244 | 1781</td><td>584655</td><td>2811 | 2532 | 5343</td><td>580709</td></tr>
<tr><td>V</td><td>237 | 1544 | 1781</td><td>585260</td><td>937 | 844 | 1781</td><td>585020</td><td>1637 | 144 | 1781</td><td>584658</td><td>2811 | 2532 | 5343</td><td>580789</td></tr>
<tr><td>VI</td><td>936 | 845 | 1781</td><td>585042</td><td>936 | 845 | 1781</td><td>585073</td><td>937 | 844 | 1781</td><td>584839</td><td>2811 | 2532 | 5343</td><td>580719</td></tr>
</table>

Table S3 The FinnGen case study

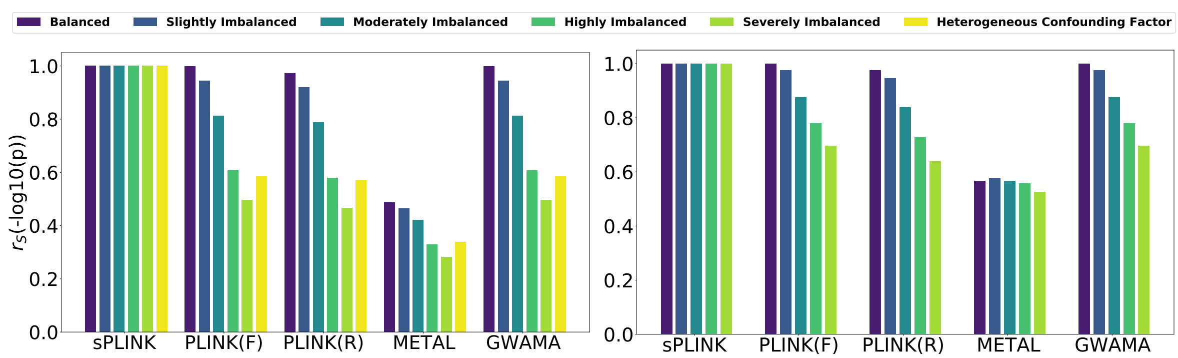
<table>
<tr><th>Scenario</th><th colspan="2">Split1</th><th colspan="2">Split2</th><th colspan="2">Split3</th><th colspan="2">Aggregated</th></tr>
<tr><td></td><td>Sample size</td><td># of SNPs</td><td>Sample size</td><td># of SNPs</td><td>Sample size</td><td># of SNPs</td><td>Sample size</td><td># of common SNPs</td></tr>
<tr><td>I</td><td>22838</td><td>997660</td><td>22838</td><td>997744</td><td>22838</td><td>997696</td><td>68514</td><td>994881</td></tr>
<tr><td>II</td><td>22838</td><td>997751</td><td>28547</td><td>997962</td><td>19983</td><td>997604</td><td>71368</td><td>995016</td></tr>
<tr><td>III</td><td>22838</td><td>997786</td><td>45676</td><td>998442</td><td>17129</td><td>997233</td><td>85643</td><td>995090</td></tr>
<tr><td>IV</td><td>22838</td><td>997722</td><td>68514</td><td>998843</td><td>14274</td><td>996997</td><td>105626</td><td>994999</td></tr>
<tr><td>V</td><td>22838</td><td>997803</td><td>99345</td><td>999114</td><td>12561</td><td>996775</td><td>134744</td><td>994918</td></tr>
</table>

Supplementary results



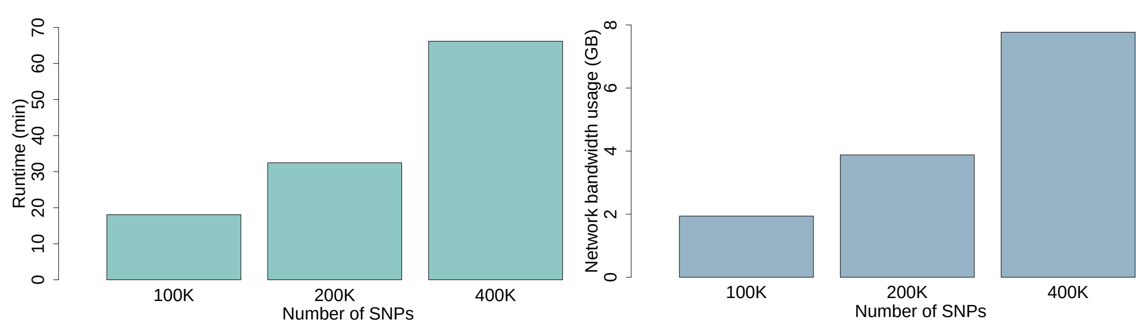
(a) Logistic regression **(b)** Linear regression **(c)** Chi-square test

Fig. S1 The significant SNPs overlapped between *sPLINK* and *PLINK* for the SHIP case study considering **Bonferroni** significance threshold, which is $\approx 1 \times 10^{-8}$ in our case. *sPLINK* and *PLINK* identify the same set of SNPs as significant.



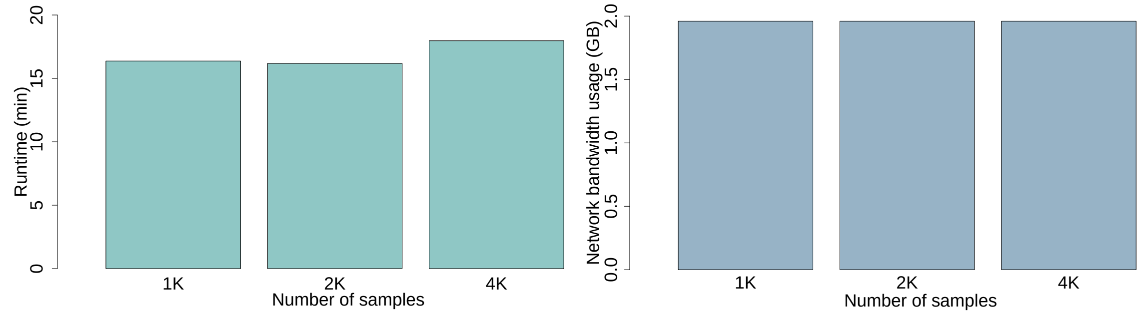
(a) The COPDGene case study **(b)** The FinnGen case study

Fig. S2 The **Spearman** rank correlation coefficient between the p-values from each tool and the aggregated analysis for the COPDGene and FinnGen case studies. *F* and *R* stand for fixed-effect and random-effect, respectively.



(a) Runtime **(b)** Network bandwidth usage

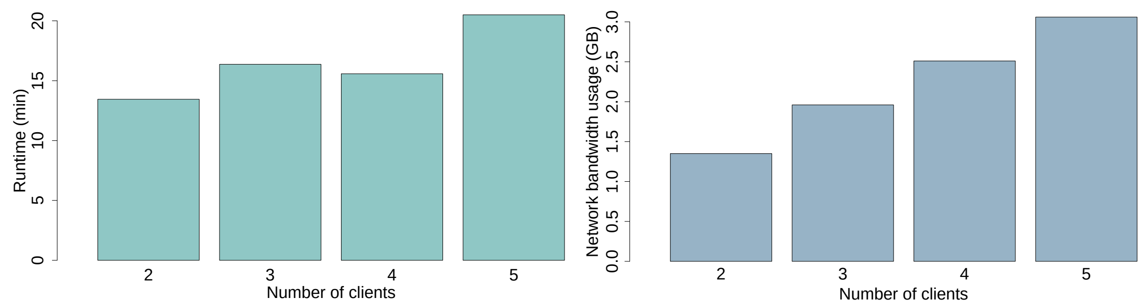
Fig. S3 Runtime and network bandwidth usage of *sPLINK* with **varying number of SNPs**



(a) Runtime

(b) Network bandwidth usage

Fig. S4 Runtime and network bandwidth usage of *sPLINK* with **varying number of samples**



(a) Runtime

(b) Network bandwidth usage

Fig. S5 Runtime and network bandwidth usage of *sPLINK* with **varying number of clients**

Experimental setup

Table S4 The system specification of the physical machines and laptops used to measure the runtime and network bandwidth usage of sPLINK; Download/upload speeds are approximate values measured using *speedtest-cli* (https://github.com/sivel/speedtest-cli); GB: Gigabyte; Mbps: Megabit per second

System name	# of cores used	Memory size (GB)	Upload (Mbps)	Download (Mbps)	Location	Experiment sets used
Server	8	12	411	527	Freising	All
Compensator	4	12	810	830	Odense	All
Laptop1	4	16	35	76	Munich	All
Laptop2	4	16	10	58	Freising	All
Laptop3	4	8	24	21	Freising	1
Laptop4	4	8	95	93	Freising	4
Desktop-PC	4	64	11	93	Freising	2,3,4

Table S5 The experimental setup used for measuring the runtime and network bandwidth usage of sPLINK; COPDGene is employed as the dataset in all experiment sets; logistic regression is used in experiment sets 2-4; In the first experiment of the experiment set 2 (i.e. sample size 1781 and SNP count 100K), 12 cores of the Desktop-PC system is used instead of 4; K: 1000

Experiment set #	Description	# of clients	Sample size per client	# of SNPs	Chunk size	Beta iterations
1	chi-square | linear | logistic	3	1781	$\sim 580K$	200K	- | - | 20
2	varying # of SNPs	3	1781	100K, 200K, 400K	100K	20
3	varying # of samples	3	1K, 2K, 4K	100K	100K	5
4	varying # of clients	2,3,4,5	1K	100K	100K	5

Supplementary Material: The Setup for the CUP Trade-off Experiment

The experiment associated with the CUP trade-off discussed in Chapter 1 is conducted in a DP-FL environment consisting of 10 clients, where each client has 5000 samples from 4 labels (out of 10 labels). That is, the sample size distribution is completely balanced, but the label distribution is NonIID across the clients. The dataset is CIFAR-10 [47], which includes total of 50000 train images and 10000 test images of shape 32×32. The ResNet-9-GN architecture is adopted from [73], which uses Mish [87] as the activation function. The number of groups of GroupNorm is 32. The dropout probabilities for the KNConv and KernelNorm layers in ResNet-9-KN is 0.05, and 0.25 respectively. The training algorithm is FedAvg [16] with local epochs of 1, optimizer is SGD with zero-momentum, and loss function is cross-entropy. Both ResNet-9-GN and ResNet-9-KN are trained for 100 communication rounds.

The clients employ DP-SGD (Algorithm 1) to train the models on their local data in a differentially private manner. The ResNet-9-GN and ResNet-9-KN models are trained using the privacy parameter values of (ε=8.0, δ=10^{-5}) and (ε=7.0, δ=10^{-5}), respectively. We perform parameter tuning using initial learning rate values of η={0.1, 0.05, 0.025, 0.0125}, clipping values of C={1, 1.5, 2.0}, and batch sizes of B={1024, 2048, 3072}. The earning rate is decayed by factor of 0.99 in each communication round. The optimal parameter values obtained for ResNet-9-GN are η=0.025, C=1.5, and B=2048. For ResNet-9-KN, they are η=0.1, C=1.5, and B=3072.

We repeat the experiment with the optimal parameter values three times and report the mean accuracy of the runs as final accuracy. We consider the average of the accuracy values in the last five communication rounds as the representative accuracy of the run. Similarly, we employ moving average with window size of five to smoothen the accuracy curves in Figure 1.1.

ACM Publication Rights and Licensing Policy

Publication Rights & Licensing Policy

Updated on January 1, 2023

Introduction

ACM embraces a not-for-profit business model that aims to assure sustainable revenue for the continued operation and enhancement of the ACM Publications Program and the ACM Digital Library, while making ACM Publications available to the widest possible global audience of computing professionals and students.

For over half a century, ACM has requested that authors transfer copyright of their articles, so that ACM could act as a steward of their published work, manage the publication process, respond to requests related to third-party rights and permissions, and defend their published Works against misconduct such as plagiarism or copyright infringement. Since that time, ACM's copyright and permissions policies have been widely used as a model by other scholarly publishers in adapting their own policies to the ever-changing realities of electronic dissemination and open access publication.

Over the years, ACM has made regular updates to its copyright policy, which is now in its 10th iteration, to ensure we are acting in the best interest of our authors and the global computing community, such as we did in 2013 when ACM introduced Exclusive, Non-exclusive, and Creative Commons licensing options as part of our ACM eRights process for authors. Many of these changes were done to support our authors with options enabling them to comply with government open science mandates around the world and to retain the underlying intellectual property of their Work.

Today, every ACM author of a scholarly Work accepted by an ACM Publication has the option of retaining the copyright of their Work and granting ACM a license to publish that Work in the ACM Digital Library. For Corresponding Authors affiliated with ACM Open participating institutions or Corresponding Authors not affiliated with an ACM Open institution, but who are willing to pay a reasonably priced Article Processing Charge (APC), there is an additional option to select an appropriate **Creative Commons** license to facilitate sharing and reuse of their Works, so the community may build on their Work without the need to obtain additional permissions from ACM or the Author, provided proper attribution is given.

As ACM continues to transition its entire scholarly Publication program to an Open Access model, the use of **Creative Commons** licensing is becoming more prevalent. In fact, many of the large government Open Science mandates around the world require the use of a Creative Commons or equivalent license when research grant recipients publish Work funded by those governments. Many private research funders are following suit (i.e. - Gates Foundation, Welcome Trust, etc.).

With its stated goal of sustainably transitioning to a fully Open Access Publisher around the end of 2025 and in response to calls for greater copyright retention and intellectual property ownership by ACM's authorship, ACM is now taking the most significant step forward since the creation of its Copyright Policy in 1994 by effectively sensetting the existing Copyright Policy and replacing it with this new Publication Rights and Licensing Policy. ACM will continue to register and hold copyright and other intellectual property rights of ACM Journals, Magazines, Conference Proceedings, Newsletters, Books, and other ACM Publications, but after January 1, 2023 ACM will no longer hold copyright in any of the newly published articles in ACM Publications.

What is Changing?

ACM will continue to require authors to assign publication rights to ACM as a condition of publishing the work. This is necessary to protect both ACM's authors and ACM against infringement and misconduct by third parties.

During the June 2022 meeting of the ACM Publications Board, the Board took perhaps the most significant "copyright-related" step taken in its history by voting to end the "Copyright Transfer" option **starting January 1, 2023**. After January 1, 2023, when authors' Works are accepted into any of ACM's Publications and enter the ACM Rights System via the link in their Acceptance Email, the "Corresponding Author" will no longer be given the option (currently listed as the 3rd of 3 options) of transfering copyright to ACM. For published Works prior to that date where copyright has been transferred by the Author to ACM, ACM will continue to be the copyright holder for such Works.

After January 1, 2023, there will be two remaining options, as follows:

- **Institutional Paid Open Access / Permissions Release** - This is the Open Access option. Wording may vary slightly depending on whether the Corresponding Author is affiliated with an ACM Open participating institution or not. If not, they will be given the option to pay an Article Proceeding Charge (APC). This option is the default when the Corresponding Author is affiliated with an ACM Open participating institution. Authors selecting this option will retain all rights to their Work and agree to grant ACM a non-exclusive permission to publish their Work in the ACM Digital Library and have the additional option of displaying a Creative Commons license on the published version of their Work in the ACM Digital Library.
- **Closed Access / Exclusive License to Publish** - This is the Closed Access option. Authors selecting this option will retain all rights to their Work and grant ACM an exclusive license to publish their Work in the ACM Digital Library.

Creative Commons Licensing Options

If the Corresponding Author of a Work accepted into an ACM Publication is either affiliated with an **ACM Open participating institution** or has decided to pay the **Open Access Article Processing Charge** (APC), the Corresponding Author will be given the additional option of applying a Creative Common license to govern how their Work may be shared and reused. Most US and European funding agencies prefer the use of the CC-BY 4.0 License, although authors should check with their specific funder to learn if their funder has any firm requirements on the version of Creative Commons license they must use as part of the publishing process.

The current ACM Policy is to allow authors the option of selecting their preferred version. ACM currently offers 6 Creative Commons license options, including:

- **CC-BY 4.0 License** - This license allows reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator. The license allows for commercial use.
- **CC-BY 4.0-SA -** This license allows reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator. The license allows for commercial use. If you remix, adapt, or build upon the material, you must license the modified material under identical terms.
- **CC-BY 4.0-NC -** This license allows reusers to distribute, remix, adapt, and build upon the material in any medium or format for noncommercial purposes only, and only so long as attribution is given to the creator.
- **CC-BY 4.0-NC-SA -** This license allows reusers to distribute, remix, adapt, and build upon the material in any medium or format for noncommercial purposes only, and only so long as attribution is given to the creator. If you remix, adapt, or build upon the material, you must license the modified material under identical terms.
- **CC-BY 4.0-ND -** This license allows reusers to copy and distribute the material in any medium or format in unadapted form only, and only so long as attribution is given to the creator. The license allows for commercial use.
- **CC-BY 4.0-NC-ND -** This license allows reusers to copy and distribute the material in any medium or format in unadapted form only, for noncommercial purposes only, and only so long as attribution is given to the creator.

Creative Common Zero (CC-0) License

There is one additional CC License that ACM Authors may apply to their research artifacts (i.e. - data, code, etc.) called **CC-0**. CC-O allows creators to give up their copyright and put their Works in the worldwide public domain. CC-0 is no longer offered in the ACM Rights system for ACM Publications, because it places the Work in the public domain and is irreversible, which could create problems for the author and ACM as the Publisher in the future. However, when ACM Authors are depositing their research artifacts either in the ACM DL or a third-party site such as GITHUB, some authors may wish to assign a CC-0 license to those research artifacts. ACM cautions the use of CC-0 unless the author has given significant consideration to this and would like to give away their copyright and allow unrestricted use of their research artifacts to the public. When ACM Authors choose to apply a CC-0 license to their research artifacts, they should indicate this alongside the artifact(s) wherever that artifact is hosted inside or outside the ACM Digital Library.

Defending Authors Against Misconduct

One of the major changes with the removal of the copyright transfer option is that regardless of which option the Author selects, ACM commits to defending their published Work in the ACM Digital Library against infringement and misconduct without the requirement to hold copyright on the published Work. In practice, ACM has been doing this for years, but is formalizing this commitment in this new Policy. When an ACM Author agrees to have ACM serve as the Publisher of Record for their accepted Work, protecting that Work against various forms of infringement and misconduct by third parties is one of the services ACM commits to provide to the Author. In return, ACM Authors agree to abide by all of **ACMs Publications Policies** and cooperate with ACM staff, volunteers, and advisers in their investigations and process to adjudicate allegations of infringement and misconduct.

Requirement to Grant ACM Exclusive or Non-Exclusive Publication Rights (applies to Journal, Conference, and Magazine articles)

ACM requires that authors have the authority to grant publication rights to ACM or that they obtain the necessary authorization to execute the grant of publication rights and that they complete ACM's Rights Management Process as a pre-condition for publishing their Work with ACM. Such grant applies to any medium used by ACM for publication (i.e.- print, online, etc.). If Authors are uncertain about their having the authority to grant these rights as a result of their employer's intellectual property rights requirements or working for a government employer with specific requirements, they should always check with their employer before completing ACM's Rights Assignment process. Authors should also take note of the following:

- Authors should incorporate the appropriate Copyright or License notice and ACM citation of the publication into copies they personally maintain on non-ACM servers.
- The author's grant of publication rights applies only to the Work as a whole, and not to any embedded objects owned by third parties. An author who embeds an object, such as an art image that is copyrighted by a third party, must obtain that party's permission to include the object, with the understanding that the entire work may be distributed as a unit in any medium.
- The requirement to obtain third-party permission does not apply if the author embeds only a link to the copyright holder's object. Other requirements for third-party permissions can be found below under the section called 3rd Party Permissions.
- Authors who wish to embed a component of another ACM-copyrighted or licensed work, e.g., an excerpt, a table, or a figure, must obtain an explicit permission (there is no fee) from ACM.

Self-Archiving and Posting Rights

All ACM published authors of magazine articles, journal articles, and conference papers retain the right to post the pre-submitted (also known as "pre-prints"), submitted, accepted, and peer-reviewed versions of their work in any and all of the following sites:

- Author's Homepage
- Author's Institutional Repository
- Any Repository legally mandated by the agency or funder funding the research on which the work is based
- Any Non-Commercial Repository or Aggregation that does not duplicate ACM tables of contents. Non-Commercial Repositories are defined as Repositories owned by non-profit organizations that do not charge a fee to access deposited articles and that do not sell advertising or otherwise profit from serving scholarly articles.

Authors should include an appropriate citation and attribution statement on all Submitted or Accepted versions of the Work similar to the following:

- **"© {Owner/Author | ACM} {Year}. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in {SourcePublication}, http://dx.doi.org/10.1145/{number}."**

For the avoidance of doubt, an example of a site ACM authors may post all versions of their work to, with the exception of the final published "Version of Record", is arXiv. ACM does request authors, who post to ArXiv or other permitted sites, to also post the published version's Digital Object Identifier (DOI) alongside the pre-published version on these sites, so that easy access may be facilitated to the published "Version of Record" upon publication in the ACM Digital Library.

Examples of sites ACM authors may not post their work to are ResearchGate, Academia.edu, Mendeley, or Sci-Hub, as these sites are all either commercial or in some instances utilize predatory practices that violate copyright, which negatively impacts both ACM and ACM authors.

Current ACM Publications Policy is that ACM sponsored and ICPS conferences may not impose embargoes on authors posting pre-prints of submissions on arXiv or disqualify such submissions that have already been posted on arXiv at the time of submission or during the peer review process. This policy was most recently reaffirmed by the ACM Publications Board in 2019. This Policy is currently under reconsideration by the ACM Publications Board and it is expected that this policy will either be reaffirmed or updated by December 31, 2022.

Requirements for ACM Books Authors

Unlike other types of ACM Publications listed above, ACM Books authors shall continue to be given the option of signing either a Copyright Transfer & Publishing Agreement or Exclusive License to Publish Agreement. The reason for this is that there are fundamental differences in how books are published, marketed, sold, and distributed via the ACM Digital Library, 3rd party channels, and in print that relate primarily to commercial considerations, financial remuneration for ACM Books authors, and posting or self-archiving policies for ACM Books, which differs from ACM's general posting and self-archiving policy for journal, conference, and magazine authors. For more information, please see the **Publishing Policies related to ACM Books authors**.

Definitive Versions of Record, Official Publication Dates, and Corrections to the Version of Record

Preserving the scholarly record "as published" is a critical component of maintaining the community and public's trust in scientific publications in general and trust in ACM specifically. As a result, ACM is committed to the publication and long term digital preservation of published works in the ACM Digital Library and via several third-party digital preservations initiatives, including **CLOCKS** and **Portico**. ACM will create and maintain a definitive Version of Record (VoR) of all ACM published works and share these with our digital preservation providers. There are instances where VoRs are hidden in the ACM Digital Library for legal or public safety reasons, to comply with other ACM Publications Policies, such as in connection with the implementation of ACMs **Name Change Policy**, when **Retractions** are made, or when Corrected Versions of Record (CVoR) are added to ACM Digital Library citation pages when errata or corrigenda are created in connection with a published work. ACM will provide the reason for the Correction on the article's Digital Library citation page. ACM does not alter works once published. There are times, however, when it is appropriate to publish a revised or corrected version of a work; doing so requires the approval of the responsible editor. Please see ACM's **Publications Policy on the Withdrawal, Correction, Retraction, and Removal of Works from ACM Publications and ACM DL**

Persistent Unique Identifiers for Every ACM Article

The **DOI (Digital Object Identifier)** is the scholarly publishing standard (ISO 26324) identifier for articles published by ACM in the ACM Digital Library. Every article in the ACM Digital Library shall have one and only one DOI.

The official publication date of an ACM published article will be considered the date on which the article’s official Version of Record (VoR) is published online in the ACM Digital Library, and the official VoR of an ACM article shall be the final peer reviewed, accepted, edited, tagged, and identified (using a DOI or other standardized identifier) definitive version that appears in ACM Publications (i.e. - journals, magazines, conference proceedings, newsletters, books, etc.) inside the ACM Digital Library.

For the avoidance of doubt, only the official VoR or in CVoR shall be considered the “Published” version of the Work for purposes of attribution, rights & permissions, prior art, investigations into potential ethics & plagiarism violations or other forms of infringement, and relevant open access embargo periods. If a new Work is substantially developed, i.e., it contains at least 25% new substantive material, it is considered a new Derivative Work or Major Revision. It is important to note that word counts are not an absolute measure, but rather a useful guid, and in general the author must use their discretion when determining if a new article is to be considered a new Derivative Work, a Minor Revision, or a Major Revision. The owner/author controls all rights in the new Work and may do as they wish with it. That said, it is commonly accepted practice that for new Derivative or Major Revision Works, the author should incorporate a citation to the previous work.

For example:

"This work is based on an earlier work: TITLE, in PUBLICATION, {VOL#, ISS#, (DATE)} © Author, {YEAR}. http://dx.doi.org/10.1145/{number}"

If the work is a *Minor Revision, the copyright or exclusive publishing license remains with ACM and the Owner should use best efforts to display the ACM citation,

"© {Owner/Author {YEAR}. This is a minor revision of the work published in PUBLICATION, {VOL#, ISS#, (DATE)} http://dx.doi.org/10.1145/{number}"

The appropriate notice should appear both within the document and in the metadata associated with the document. Instructions for how to do this will be found in the instructions for authors in ACM's various publications.

Solicited Works

From time to time, ACM solicits works for publication. Examples are columns, invited works, award lectures, and keynote speeches. ACM asks authors of such works not to distribute copies or post these works on their Home Pages until ACM has published them. Authors who wish to circulate before publication should get permission from ACM. ACM considers lectures and speeches to be published at the time they are given.

PERMISSIONS

ACM grants gratis permission for individual digital or hard copies made without fee for use in academic classrooms and for use by individuals in personal research and study. Further reproduction or distribution requires explicit permission and possibly a fee.

ACM is now a signatory of the **STM Permission Guidelines Initiative**, which supports an approach to research based on common decency, respect, fairness and mutual trust. These Guidelines are built to allow Signatory STM Publishers to use limited amounts of material in other original published works without charge, and with a minimum of effort needed for permissions clearance. ACM joined the initiative in 2022 to lower the burden on authors to obtain third party permissions when authoring works for ACM and third party publishers.

All copies should carry the original citation, the appropriate copyright and notice of permission on the first page or initial screen of the document. (See **§2.2 Copyright Notice**.)

Most permission requests should go through ACM's automated rights system available in the ACM Digital Library and pointed to by **permissions@acm.org**. Requests that cannot be handled through the online system will take longer to resolve: requestors may expect a response to their inquiry within seven business days.

Fair Use for Educational Purposes

Definition of classroom use: Copying and distributing single works by a university/college instructor, where no fee is charged to the students, and the distribution is limited to students enrolled in a university/college course and their instructors.

- **Course Material** - Permission granted without fee if the course material is produced without charge to the student. (See Commercially produced Course Packs below.)

- **Electronic Reserves** - Permission granted without fee provided the library or institution has an authentication mechanism for controlled access to the server and a license to the ACM-published work. A college, university or other accredited institution may place a copy of a definitive Version of Record of the work in its library's electronic reserves for the duration of its educational needs for that work, provided that access is limited to its enrolled students (including those in its distance learning programs), faculty, and staff. Those institutions without a current license to the work should contact **permissions@acm.org**.

- **Distance Learning** - Permission granted without fee for distance learning students enrolled at the institution. They have the same access rights to those ACM copyrighted materials licensed by their institution as any other student. Since institutional access is authenticated by IP address, it is up to the institution to provide a proxy server for its remote users, and to register the IP address of that proxy with ACM.

- **Interlibrary Loan (ILL)** - Permission granted without fee for an institution with an ACM Digital Library license to download and print works for Interlibrary Loan. The Digital Library may be used as the source for the printed copy. The loan of the work is limited to printed copies, as part of normal library functions.

- **Walk-Ins** - Permission granted without fee for access to all ACM publications, print or electronic, by all members of the community which a subscribing library is charted to serve.

- **Open Access / Creative Commons Material** - Permission is granted without fee, provided proper attribution is given to the Author(s) and Publisher at the time of use.

Commercial Republication

Definition of commercial republication: Any use that is not personal or non-profit educational use. Includes reprinting by trade and scholarly publishers, and use in corporate settings and their web sites, both internal and external. No direct profit need be realized from the publication or sale of ACM material.

Commercial use normally requires a license and payment of release fees. All reproductions other than those listed in this document require specific permission and a fee payable to ACM. This includes republishing in textbooks, commercially-produced course packs sold to students, anthologies, and other edited publications, and posting or other electronic distributions, unless use is done in connection with the **STM Permission Guidelines Initiative**.

- **Commercially Produced Course Packs** - Use of copyrighted or licensed material in course packs sold to students requires an appropriate license. Send requests to **permissions@acm.org** or go to **http://www.copyright.com**.

- **Print permission** - A grant of permission involves consultation with the lead author of the work, the publisher's agreement to pay the required fees, and prominent display of the proper credit acknowledgment.

- **Electronic permission** - Rules for commercial distribution will apply unless the request falls under educational use as defined above. Fees for internal and external commercial posting of ACM published material are tied to the term of the license. All postings must include pointers to the correct Citation Page in the ACM Digital Library.

- **Multiple copies** - Producing multiple copies of ACM copyrighted or licensed works for distribution to more than ten peers, co-workers, clients, etc. requires a transactional license from the CCC and payment of the required per copy fee Send requests to **permissions@acm.org** or go to**http://www.copyright.com**.

- **Software** - Owners/Authors of software grant ACM a non-exclusive permission to publish and manage all rights and permissions themselves.

3rd Party Permissions

Lastly, another major change relating to how ACM handles rights and permissions is that ACM has adopted **STM Permissions Guidelines**, which simplifies the process for third parties (including researchers) to reuse ACM published content in new works under development. This is a broad-based publisher initiative that includes the vast majority of publishers in computing literature. Other signatories of these guidelines are listed **here**. It is our goal to simplify the process of publishing with ACM, and we welcome your feedback after the above steps have been implemented.

ACM publications staff will monitor requests for permission not handled by ACM's automated permissions system which is accessed via the ACM Digital Library. Persons granted permission to copy an ACM published work should display the appropriate Publication Notice followed by: "Included here by permission."

Edited Collections

Edited collections such as conference proceedings and newsletters are copyrighted as a whole by ACM. Going forward after January 1, 2023, authors will retain the copyright of individual components of those Works, such as articles, letters-to-the-editor, abbreviated works, etc. For these individual components, ACM will obtain either an exclusive or non-exclusive permission to publish (conveyed tacitly or by the ACM Permission Form) that permits publication in both print and online forms, and also grants ACM the right to transform the work into any formats as necessary for use within the ACM Digital Library or other media.

No ACM-copyrighted or exclusively licensed collection may be posted for open distribution without prior permission from ACM and before it has been included in the ACM Digital Library. Approved distributions must include a notice of this permission along with the copyright notice for the Work.

Links

ACM treats links as citations (references to objects) rather than as incorporations (embedding of objects). Permission is not needed to create links to citations in The ACM Digital Library or Online Guide to Computing Literature. ACM encourages the widespread distribution of links to the definitive Version of Records of its copyrighted works in the ACM Digital Library and does not require that authors obtain prior permission to include such links in their new works.

However, someone who creates a work or a service whose pattern of links substantially duplicates an ACM-copyrighted volume or issue should get prior permission from ACM. One example: the creator of "A Table of Contents for the Current Issue of TODS" -- consisting of citations and active links to author-versions of the works in the latest issue of TODS -- needs ACM permission because that creator is reproducing an ACM-copyrighted work. If all the links in the "Table of Contents" pointed to the ACM-held definitive Version of Records, ACM would normally give permission because then the new work advertises an ACM work. To avoid misunderstandings, consult with ACM before duplicating an ACM work via links.

If an author wishes to embed a copyrighted object---rather than a link---in a new work, that author needs to obtain the copyright holder's permission.

Distributions From non-ACM Servers

Service providers do not need to obtain prior permission from ACM to locate and dispense links to the ACM-held definitive Version of Records of works, but they do need permission if they are making, collecting, or distributing copies of ACM-copyrighted or licensed works.

Other Related Policies

Conference Publication Policy

Please see the **Conference Publication Policy** for additional expectations related spcifically to ACM Conference Publications.

Inappropriate Content Policy

Please see **ACM's Inappropriate Content Policy**

Submitting and Investigating Potential Violations of this Policy

See **Policy on Submitting and Investigating Claims**

Confidentiality Policy

See **Confidentiality Policy**

Communicating Results of Investigations

See **Policy on Communicating Results of Investigations**

Appealing Violation Decisions

See **Appealing Policy Violation Decisions**

Contact ACM

The ACM Director of Publications should be contacted for any:

- Questions about the interpretation of this policy
- Questions about appeals of decisions
- Requests for deviations from, or extensions to, this policy
- Reporting of egregious behavior related to this policy, including purposeful evasion of the policy or false reporting

Mailing address:
ACM Director of Publications
Association for Computing Machinery
1601 Broadway, 10th Floor
New York, NY 10019-7434
Phone: +1-212-626-0659
Or via email:
scott.delman@hq.acm.org