Striking the Balance:
Optimizing Privacy, Utility, and Complexity in Private Machine Learning

by

Yue Niu

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Electrical and Computer Engineering)

August 2024

Dedicated to my beloved wife Zeheng Li,

my parents and my sister.

Acknowledgements

Starting from 2018 and ending in 2024, I am excited and grateful that I am reaching the end of the chapter of my PhD. This six-year research experience brought me to a new stage, where I learned to define research problems, to collaborate with people, and, importantly, to manage expectations and navigate the ups and downs of research and everyday life. I am grateful for the companionship and support of friends, mentors, and colleagues on this journey.

I first want to express my gratitude to my advisor, Professor Salman Avestimehr. I started working with Professor Avestimehr in my second year. As a researcher, Professor Avestimehr is sharp in defining research problems and looking for solutions. When I joined his group, I had few experiences in privacy-related research. Professor Avestimehr motivated and helped me look for interesting problems. The main topics and outcomes in this thesis stem from thorough and intensive discussions with him. Professor Avestimehr helped me quickly evaluate whether a problem/solution is good. On many occasions, he immediately identified flaws and issues in my proposals and then gave valuable suggestions. Moreover, I am grateful to have worked with him and learned the way he communicates with people. Professor Avestimehr can always grab essential components and explain them simply yet accurately. His communication style greatly motivates me to reflect on and sharpen my own communication skills.

I also want to thank our department academic advisor, Diane Demetras. When I decided to work with Professor Salman Avestimehr, I had a lot of conversations with her. During that hard time, I was struggling to look for a professor to work with. Diane provides as much information and help as I need. She gave me confidence when I needed to speak with professors and senior colleagues. When I was unsure about my choice, she offered valuable suggestions to help me make the right decision.

My collaborators, mentors, and friends, Ramy Ali and Saurav Prakash, also gave me valuable support and help in my research. Ramy was a PostDoc in our lab. We work hard to publish our first work on private machine learning. Ramy helped me a lot with proofreading our work and writing. I learned a lot from him on professional writing. Saurav was a senior PhD candidate in our lab. We had many conversations on research as well as everyday life. Importantly, Saurav gave me valuable advice on communicating with my advisor and other senior colleagues.

I also want to thank my committee members, Professor Murali Annavaram and Professor Meisam Razaviyayn, for their time and suggestions during my defense. Professor Murali Annavaram also mentored me in my research and closely participated in discussions and paper writing. I am also grateful for his advice and support during my job search.

I cherished the time spent with my lab mates, Amir Ziashahabi, Asal Mehradfar, Duygu Nur Yaldiz, Emir Ceyani, Erum Mushtaq, James Flemings, Lei Gao, Roushdy Elkordy, Yavuz Faruk Bakman, Rachit Rajat, Sara Babakniya, Tara Renduchintalam, Tingting Tang, Tuo Zhang, Zalan Fabian. They are always there when I want to share good and bad news.

At last, I am very fortunate to have finished this journey with my wife alongside me. In the fourth and fifth years of my PhD, I was struggling to make progress. I am fortunate to spend time with my wife, sharing our vision and feelings. The sharing gave me the energy to move forward. I treasure the time we spend traveling to different places, seeing different things, and rejuvenating ourselves. I also thank my sister and my parents. Since I came to the US in 2018, my sister has been duly playing my role in our family. I really thank her for her support during the journey.

Table of Contents

List of Tables

List of Figures

Abstract

Machine learning (ML) has become a backbone of current intelligent systems, such as computer vision and natural language processing. Owing to the surprising capabilities exhibited by contemporary models like transformers and convolutional neural networks (CNNs), machine learning systems have markedly narrowed the gap between humans and machines in many tasks previously thought to be exclusive to humans. However, a looming issue persists amidst the advancements in current machine learning: data privacy. Numerous studies have revealed that current models often leak private information in data. Adversaries can easily breach privacy by interacting with the model via attacks such as model inversion or membership inference. Current private ML solutions typically achieve privacy protection by making trade-offs with other design metrics, such as sacrificing model utility or increasing complexity. Consequently, the privacy-utility-complexity trilemma still has not been effectively resolved.

This thesis investigates privacy protection in machine learning and presents private ML frameworks that strike a balance between privacy, utility, and complexity. We design private training and inference algorithms by integrating the inherent structure of models and data into established privacy techniques (e.g., differential privacy, federated learning). In particular, this thesis studies the following scenarios.

This thesis first studies data protection with a typical computing paradigm involving a private and public execution environment. The private environment features strong privacy protection at the cost of low computation capacity. The public environment offers fast execution but lacks necessary privacy guarantees. The computing paradigm is seen in many real-world scenarios, such as client-server systems and trusted execution with public GPUs. This thesis presents a novel methodology for private training and inference that leverages both the private and public environments. The proposed methodology features a unique way of distributing data and computation in model training and inference. At a high level, the private environment secures the most sensitive information but only performs minimal computations involved in training and inference. On the other hand, the public environment undertakes the most computations but with minimal information leakage. The computation flow also ensures input information is captured and learned in different environments without theoretical information loss. Therefore, the privacy-utility-complexity trilemma is significantly mitigated.

The second part of this thesis investigates the privacy-utility-complexity trilemma in practical federated learning (FL) settings, where multiple private local resource-constrained clients collectively train a large ML model. To enable federated learning of large models in resource-constrained clients, the thesis presents a novel methodology that distributes full-model training across multiple clients. Each client is tasked with training a small sub-model. Different from the current literature, this thesis proposes a unique and effective model decomposition method for creating sub-models for participating clients. The model decomposition ensures each sub-model sufficiently approximates the original full model, and all sub-models together provide a near-full coverage of the full model. Hence, every part of the original full model will be trained on participating clients. Owing to the effective model decomposition and training methodology, the final model utility is preserved without incurring significant training complexity on resource-constrained clients. Moreover, the privacy assurance offered by federated learning is also upheld.

In a word, this thesis aims to address the privacy-utility-complexity trilemma in private machine learning. To that end, the thesis presents unique solutions that leverage the model and data inherent structure for better distributing information and computation, thereby balancing privacy, model utility, and training complexity.

Chapter 1

Introduction

Over the past years, many fields have observed the potential of machine learning (ML) in automating tasks and improving efficiency. Convolution neural networks (CNNs) [52, 84, 39], and vision transformers have significantly advanced the developments of computer vision such as image classification [39], objective detection [78, 38], and segmentation [105]. Transformers [91, 25, 14] are the driving force behind recent advances in natural language processing. The potential of ML also ripples beyond the basic ML applications and is being observed in our real life, such as autonomous driving [56], healthcare [95], and virtual assistants [17].

As ML becomes the backbone for many real-world systems, the attention to its unintended consequences is growing. In particular, data privacy in ML-driven systems is of great concern. Data is one of the most important components in ML pipelines. ML datasets encode a vast amount of information that is learned by and encoded in ML models. In many scenarios, an ML dataset contains highly sensitive information. For instance, hospitals may have datasets that store patients' records; social media platforms store all users' information and interaction history. As model training or inference usually requires sharing data with other parties, it opens a door for potential adversaries that can reveal private sensitive information.

Current literature has shown that in model training and deployments, data privacy can be broken in many ways. For instance, unsafe communication channels or untrusted execution environments can expose private data to other parties, who may eavesdrop on the communication channels or execution environments to steal private data for their own purposes [22, 99, 18]. Furthermore, attacks such as model inversion and membership inference can infer information of training data simply via a model's API [30, 104, 83].

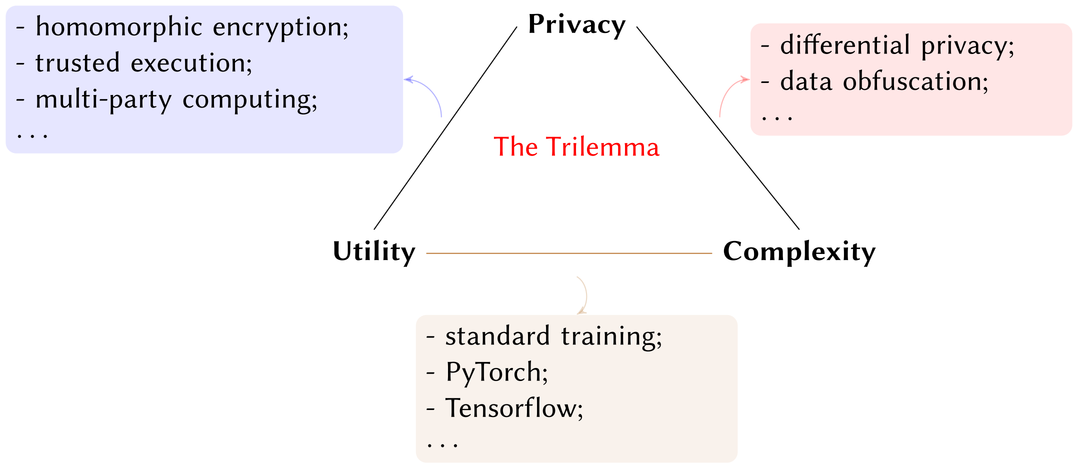


Figure 1.1: The privacy-utility-complexity trilemma in private machine learning.

However, data privacy is also difficult to achieve without introducing adverse effects. As shown in Figure 1.1, current solutions typically make tradeoffs between three design metrics: privacy, utility, and complexity. That is, the privacy-utility-complexity trilemma. For instance, the standard model training usually delivers models with the best model utility. However, it offers no privacy protection throughout the whole pipeline. On the other hand, while homomorphic encryption [101] and trusted execution [79] provide secure computing environments against untrusted parties, they introduce prohibitively high complexity. Model training with differential privacy [27, 1] or data obfuscation [103, 98] typically achieves strong privacy protection by sacrificing model utility. In a word, current ML pipelines still have not provided an effective solution that achieves a good balance between the aforementioned three design metrics: privacy, utility, and complexity. As a result, private machine learning still has not been widely adopted in real-world applications, even when data privacy is seriously compromised.

This thesis aims to mitigate the privacy-utility-complexity trilemma in private machine learning. In particular, this thesis focuses on two directions: efficient private machine learning in private and public environments, and efficient private machine learning in federated settings.

1.1 Efficient Private Learning in Private and Public Environments

This thesis first investigates a generic setting with a private and a public execution environment. The private environment provides strong data protection during training and inference. However, the private environment incurs additional complexity for protection or lacks sufficient computing capacity, leading to low computing performance. The public environment features high-end computing resources (e.g., GPUs), which offer much faster computing compared to the resource-constrained private environment. On the other hand, the public environment exposes data or models to the public without any protection.

The setting can be generalized in real-world scenarios. For instance, distributed machine learning involves multiple computing nodes collaboratively training a model, where the local nodes act as a private environment, and the remote nodes act as a public one [58, 82]. Systems with trusted execution environments (TEEs) and GPUs use TEEs as the private environments and GPUs as the public environments [89, 68].

With the heterogeneous setting, this thesis proposes a methodology, `AsymML`, that effectively exploits their advantages: strong protection in the private environment, and fast execution in the public environment. `AsymML` first reveals the asymmetric structure in data and models' internal activations. In particular, the main information can be encoded into a low-dimensional representation. With this crucial observation, `AsymML` develops a unique data decomposition method that distributes data and computation into the private and public environment. Specifically, `AsymML` secures most information in the private environment with a low-dimensional representation, and offloads residual information into the public environment. `AsymML` further decomposes the computation flow (models' forward and backward passes) to minimize complexity in the private environments, and minimize information leakage in the public environments.

Along this direction, this thesis further proposes a fully asymmetric computation flow, `Delta`, to mitigate the bottleneck of inter-environment communication. `Delta` features two fully separated data flows, with one associated with the private environment and another one associated with the public environment. Compared to `AsymML`, `Delta` trains two separate models. A small model learns the low-dimensional representation in the private environments; a large model learns the high-dimensional residual representation in the public environment. Final predictions are obtained by aggregating the predictions from each model. Therefore, communication is minimized by eliminating layer-wise inter-environment data exchanges as in `AsymML`. The proposed model training and inference framework can be generalized to many real-world settings with private and public environments, such as systems with TEEs/GPUs, and client-server setups.

1.2 Efficient Private Learning in Federated Settings

The second part of this thesis aims to mitigate the privacy-utility-complexity trilemma in federated learning (FL) settings. This part considers a typical FL setting of training large models at the resource-constrained edge. The setting is commonly seen in the real world, where model training needs to be conducted in edge devices, a party that owns abundant data but cannot share with other parties [35, 33]. Prior works either aim to reduce complexity at the edge by training compressed models, which results in a degradation of the model utility at the edge, or break the privacy promises of the standard FL by sharing data with remote servers. As a result, the privacy-utility-complexity trilemma still persists.

To enable federated learning at the resource-constrained edge, this thesis proposes a new sub-model training methodology that reduces training complexity while still preserving models' utility as full-model training. This thesis first reveals the inherent low-rank structure in models widely used in real-world applications. In particular, for a linear layer such as a convolution layer and a fully connected layer, the weight matrix can be reduced to a low-rank representation without significantly affecting the model's performance. With the model's low-rank structure, the proposed method, `PriSM`, creates different low-rank approximations of the original model for different participating clients via performing a sampling process on the original model. The sampling process ensures each sub-model approximates the original model so that multiple clients still share similar initial model states in every training cycle. On the other hand, owing to the sampling process, different parts of the original model are distributed to multiple clients. Therefore, it ensures all sub-models together still provide nearly full coverage of the original models, leading to much-improved model utility. Compared to other compression methods, `PriSM`, though trains compressed models on clients, can still recover the full-model capacity on the server side.

The proposed method is extensively evaluated on multiple tasks, including vision and language tasks. In particular, `PriSM` performs much better than current FL algorithms when data on participating clients present a highly non-i.i.d distribution (independent and identically distributed). The performance gain stems from the unique training process in `PriSM`, where different clients are allowed to train a slightly different sub-model. Therefore, data with non-i.i.d distribution on clients can be learned by different low-rank versions of the original model.

Thesis structure. In the rest of this thesis, I will elaborate on the aforementioned works on mitigating the privacy-utility-complexity trilemma. In Chapter 2 and 3, I will dive into two works on efficient private learning using private and public environments. In Chapter 4, I will present efficient private learning in federated settings. At last, I will conclude the thesis in Chapter 5.

Chapter 2

3LegRace: Privacy-Preserving DNN Training over TEEs and GPUs

Leveraging parallel hardware (e.g. GPUs) for DNN training brings high computing performance. However, it raises data privacy concerns as GPUs lack a trusted environment to protect the data. Trusted execution environments (TEEs) have emerged as a promising solution to achieve privacy-preserving learning. Unfortunately, TEEs' limited computing power renders them not comparable to GPUs in performance. To improve the trade-off among privacy, computing performance, and model accuracy, we propose an *asymmetric* model decomposition framework, `AsymML`, to (1) accelerate training using parallel hardware; and (2) achieve a strong privacy guarantee using TEEs with much less accuracy compromised compared to DP-only methods. By exploiting the low-rank characteristics in training data and intermediate features, `AsymML` asymmetrically decomposes inputs and intermediate activations into low-rank and residual parts. With the decomposed data, the target DNN model is accordingly split into a *private* and an *public* part. The private part performs computations on low-rank data, with low compute and memory costs. The public part is fed with residuals perturbed by very small noise. Privacy, computing performance, and model accuracy are well managed by respectively delegating the private and the public parts to TEEs and GPUs. Furthermore, we present a rank bound analysis showing that the low-rank structure is preserved after each layer across the entire model. Our extensive evaluations on DNN models show that `AsymML` delivers $7.6\times$ speedup in training compared to the TEE-only executions while ensuring privacy.

2.1 Introduction

Deep neural networks (DNNs) are acting as an essential building block in various applications such as computer vision (CV) [84, 39] and natural language processing (NLP) [25]. Efficiently training a DNN model usually requires a large training dataset and sufficient computing resources. In many real applications, datasets are locally collected and not allowed to be publicly accessible, while training is computation-intensive and hence usually offloaded to parallel hardware (e.g., GPUs). Considering that data transfer can be hacked or a runtime memory with sensitive data can be accessed by third parties, such practice poses serious privacy concerns.

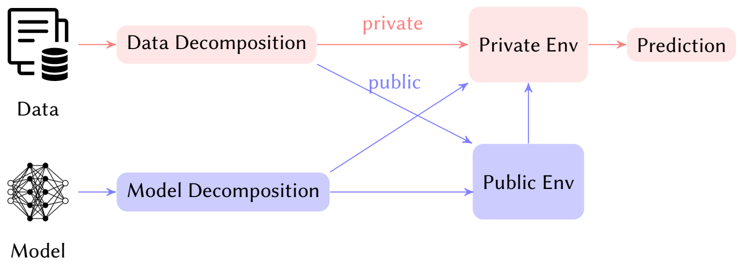


Figure 2.1: An Overview of `AsymML` is depicted. In `AsymML`, the models are asymmetrically decomposed into private and public parts. The private part performs computations on low-rank part X^{main} at small compute/memory cost; while X^{res} perturbed by small noise is offloaded onto the public part with little critical information involved.

The need for private data protection has motivated privacy-preserving machine learning methods such as machine learning with differential privacy (DP) [1, 74, 54] and machine learning using trusted execution environments (TEEs) [76, 34, 89].

DP-based methods usually defend against membership inference and model-inversion attacks [83, 30] that aim to infer or reconstruct the training data through the DNN models [1]. Specifically, by injecting noise to the gradients during training, DP-based methods reduce the correlation between the model parameters and the training data. Therefore, reconstructing training data through the model becomes more challenging. In addition to applying DP to the models, [103, 66] also apply DP directly to the input data during training. However, DP alone usually compromises too much accuracy to achieve strong privacy guarantees [9].

Unlike such DP methods, TEEs provide a direct hardware solution to protect data from any untrusted entities. TEE-based methods do not compromise accuracy as in DP. Trusted platforms such as Intel Software Guard Extensions (SGX) [21] and Arm TrustZone [55] create a sufficiently secure runtime environment, where sensitive data can be stored and processed. By performing all computations in TEEs, any untrusted access to internal memory is forbidden. However, the computing performance is severely affected when the TEEs are solely leveraged due to their limited computation capabilities as a result of not having GPU support. Such executions are known as the TEE-only executions. A natural solution for this problem is to leverage both TEEs and the untrusted GPUs while protecting the privacy of the data. This idea was leveraged in [89] to develop a privacy-preserving framework known as Slalom for DNN inference. However, Slalom does not support DNN training, which is a more challenging and computationally-intensive task.

Contributions – To efficiently perform private training in a heterogeneous system with *private* TEE-enabled CPUs and fast *public* accelerators, we propose a new training framework, `AsymML` [70]. By exploiting the potential low-rank structure in the inputs and intermediate features X, `AsymML` first *asymmetrically* decomposes the inputs and the features into a low-rank part X^{main} and a residual X^{res}. `AsymML` then accordingly decomposes the model into private and public parts, in which the private part is fed with X^{main}, and the public part is fed with X^{res} perturbed by a very small noise. When the inputs and intermediate features have a low-rank structure, the private part with X^{main} incurs small computation and memory costs, while the public part handles most computations with little privacy revealed. The DNN model training is performed by respectively delegating the private and public part to TEEs and GPUs, where TEEs protect the privacy, and GPUs guarantee the computing performance. We also present a theoretical analysis showing that the low-rank structure is preserved after each layer in the DNN model, which ensures efficient asymmetric decomposition. In summary, our contributions are as follows.

1. We propose a privacy-preserving training framework that decomposes the data, the intermediate features and the models into two parts which decouples the information from the computations. The decomposition is based on a lightweight singular value decomposition (SVD).
2. We provide essential theoretical analyses that show that the low-rank structure in intermediate features is well-preserved after each layer in a DNN model.
3. We implement `AsymML` in a heterogeneous system with TEE-enabled CPUs and fast GPUs that support various models. Our extensive experiments show that `AsymML` achieves up to $7.6\times$ training speedup in VGG and $5.8\times$ speedup in ResNet variants compared to the TEE-only executions.

2.2 AsymML Framework

In this section, we present `AsymML`. We start with the threat model considered in our work and then describe `AsymML` in detail, including data and model decomposition in DNNs through a lightweight SVD approximation. Finally, we provide the computation and the memory costs of `AsymML`.

Notations – We use lowercase letters for scalars and vectors, and uppercase letters for matrices and tensors. $\overline{X}$ denotes a 2D matrix flattened from a multidimensional tensor X, while $\left\|\overline{X}\right\|_F$ denotes the Frobenius norm of the matrix $\overline{X}$. $\overline{X}^*$ denotes the transpose of $\overline{X}$. X_i denotes i-th slice of a tensor $X \in \mathbb{R}^{N\times h\times w}$, $X_{i,:,:}$, while $X_{i,j}$ denotes (i,j)-th slice, $X_{i,j,:}$. We use $\circledast$ to denote convolution, and $\cdot$ for matrix multiplication. For a DNN model, given a loss function $\mathcal{L}$, $\nabla_W \mathcal{L}$ denotes gradients of the loss w.r.t parameters W. Finally, we use $\log$ to denote the logarithm to the base 2.

2.2.1 Threat model

Based on the capabilities of common TEE platforms such as Intel SGX [21], we consider the following threat model. 1) An adversary may compromise the OS where the TEE is running. However, it cannot breach the TEE environment, 2) the adversary may access hardware disk, runtime stack, memory outside TEEs and communication between trusted and untrusted environments, 3) the adversary may obtain the model parameters and the gradients during training and then analyze underlying relations with the training data, 4) the adversary may obtain data in untrusted environments and infer information in training data, and 5) the adversary may gain some knowledge of the training dataset (e.g., labels), and use public/online resources to improve its attack performance.

However, we do not consider side-channel attacks that compromise TEEs by probing physical signals such as power consumption and electromagnetic leaks.

2.2.2 Asymmetric data and model decomposition using SVD

`AsymML` decomposes compute-intensive and memory-intensive modules, especially convolutional layers in modern DNNs [84, 39], and then assigns each part to a suitable platform. At a high level, `AsymML` starts with decomposing a convolutional layer such that the computation involving privacy-sensitive information is performed in TEEs while the residual part is offloaded to GPUs. Specifically, the input of a convolutional layer denoted by X is decomposed into a low-rank part X^{main} and a residual part X^{res} as shown in Fig. 2.2. When the convolutions in GPUs and TEEs are completed, outputs from GPUs denoted by Y^{res} are merged into TEEs denoted by Y^{main}, and then followed by a non-linear layer (a pooling layer might be also needed). Outputs after a non-linear layer will be then re-decomposed before proceeding to the next convolution layer. During the whole forward/backward pass, the low-rank part in TEEs is never exposed, which effectively prevents crucial parts of data from being leaked.

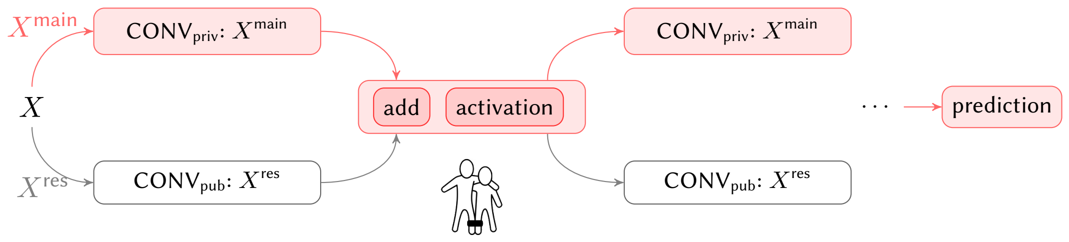

X^{main}
$\text{CONV}_{\text{priv}}$: X^{main}
X
X^{res}
CONV_{pub}: X^{res}
add
activation
$\text{CONV}_{\text{priv}}$: X^{main}
CONV_{pub}: X^{res}
...
prediction

Figure 2.2: An illustration of the model decomposition in `AsymML` is depicted. First, the input of a convolutional layer X is decomposed into a low-rank part X^{main} and a residual part X^{res}. The convolution of the private and the public parts are then performed in TEEs and GPUs, respectively. Finally, the results are combined in TEEs followed by a non-linear layer and then the decomposition is performed again and so on. Therefore, training using `AsymML` behaves like a "three-legged race".

We now explain the decomposition in detail. For a convolutional layer with input $X \in \mathbb{R}^{N\times h\times w}$ and kernels $W \in \mathbb{R}^{M\times N\times k\times k}$, where N and M are the number of input and output channels, and h, w and k are the size of inputs and kernels respectively, the i-th output channel is computed as follows[*]

$$Y_i = \text{CONV}(X, W_i) = \sum_{j=1}^{N} X_j \circledast W_{i,j}. \tag{2.1}$$

By splitting the input X into X^{main} and X^{res}, `AsymML` decomposes the convolution into a private convolution $\text{CONV}_{\text{priv}}$ and an public convolution CONV_{pub} as follows

$$Y_i = \text{CONV}_{\text{priv}}(X^{\text{main}}, W_i) + \text{CONV}_{\text{pub}}(X^{\text{res}}, W_i). \tag{2.2}$$

The *asymmetric* decomposition is inspired by an observation that channels in inputs and intermediate activations are usually highly correlated. By exploiting such channel correlations, X can be decomposed in a way that a low-rank tensor X^{main} keeps most information, while X^{res} stores the residuals. Therefore, the complexity of $\text{CONV}_{\text{priv}}$ is significantly reduced with little privacy compromised.

[*]The batch size and the bias are omitted here for simplicity.

To extract a low-rank tensor X^{main}, we first apply singular value decomposition (SVD) to $\overline{X} \in \mathbb{R}^{N \times hw}$ flattened from X as

$$\overline{X} = U \cdot \text{diag}(s) \cdot V^*, \tag{2.3}$$

where the *principal* channels are stored in V while the corresponding singular values in s denote the importance of each principal channel. The j-th channel in X^{main} is obtained from the first $\mathcal{R}$ most principal channels as follows

$$X_j^{\text{main}} = \sum_{p=1}^{\mathcal{R}} s_p \cdot U(j,p) \cdot X_p^{'} \equiv \sum_{p=1}^{\mathcal{R}} a_{j,p} \cdot X_p^{'}, \tag{2.4}$$

where $X_p^{'} \in \mathbb{R}^{h \times w}$ is a 2D matrix reshaped from the p-th column of V and $a_{j,p} = s_p \cdot U(j,p)$. On the other hand, the residual part X_j^{res} is given by

$$X_j^{\text{res}} = X_j - X_j^{\text{main}} = \sum_{p=\mathcal{R}+1}^{N} a_{j,p} \cdot X_p^{'}. \tag{2.5}$$

With the low-rank input X^{main}, the forward and backward passes of CONV$_{\text{priv}}$ can be reformulated in a way such that the complexity depends on the number of principal channels $\mathcal{R}$ as follows.

- **Forward.** During a forward pass, the outputs in TEEs are calculated as follows

$$\begin{aligned} Y_i^{\text{main}} &= \sum_{j=1}^{N} X_j^{\text{main}} \circledast W_{i,j} = \sum_{j=1}^{N} \sum_{p=1}^{\mathcal{R}} a_{j,p} X_p^{'} \circledast W_{i,j} \\ &= \sum_{p=1}^{\mathcal{R}} X_p^{'} \circledast \sum_{j=1}^{N} a_{j,p} W_{i,j} \equiv \sum_{p=1}^{\mathcal{R}} X_p^{'} \circledast W_{i,p}^{'}, \end{aligned} \tag{2.6}$$

  where $W_{i,p}^{'} = \sum_{j=1}^{N} a_{j,p} W_{i,j}$. According to Eq. (2.6), CONV$_{\text{priv}}$ essentially is a convolution operation with $\mathcal{R}$ input channels, and the kernels $W^{'}$ that are obtained by regrouping W.

- **Backward.** During a backward pass, given gradient $\nabla_Y \mathcal{L}$, $\nabla^{(\mathrm{T})}_{W_{i,j}} \mathcal{L}$ in TEEs is computed as

$$\nabla^{(\mathrm{T})}_{W_{i,j}} \mathcal{L} = X_j^{\mathrm{main}} \circledast \nabla_{Y_i} \mathcal{L} = \sum_{p=1}^{\mathcal{R}} a_{j,p} \cdot X_p^{'} \circledast \nabla_{Y_i} \mathcal{L}, \tag{2.7}$$

where $X_p^{'} \circledast \nabla_{Y_i} \mathcal{L}$ for $p = 1, 2, \cdots, \mathcal{R}$ are first computed. $\nabla^{(\mathrm{T})}_{W_{i,j}} \mathcal{L}$ is obtained by a simple linear transformation with $a_{j,p}$. Hence, the computation complexity of backward passes also depends on $\mathcal{R}$.

On the other hand, the untrusted forward output Y_i^{res} and the backward gradient $\nabla^{(\mathrm{U})}_{W_{i,j}} \mathcal{L}$ in GPUs are the same as in the classical convolutional layers but with a different input $\mathcal{M}(X^{\mathrm{res}})$. The final result Y_i and $\nabla_{W_{i,j}} \mathcal{L}$ are obtained by simply adding the results of the TEEs and the GPUs. In addition, as computing $\nabla_X \mathcal{L}$ does not involve the input X, it is offloaded onto GPUs.

2.2.3 Lightweight SVD approximation

Performing exact SVD on $X \in \mathbb{R}^{N \times h \times w}$ incurs a significant complexity of $O(Nh^2w^2 + N^2hw)$, which goes against our objective of reducing the complexity in TEEs. To reduce the complexity, we propose a lightweight SVD approximation that reduces complexity to only $O(\mathcal{R}Nhw)$.

SVD essentially is an algorithm that finds two vectors $\boldsymbol{u}^{(i)}$ and $\boldsymbol{ii}$ to minimize $\left\| \overline{X}^{(i-1)} - \boldsymbol{u}^{(i)} \cdot \boldsymbol{ii}^* \right\|_F$, where $\overline{X}^{(i-1)}$ is the remaining $\overline{X}$ with $i-1$ most principal components extracted. $\{\boldsymbol{u}^{(i)}\}_{i=1}^{hw}$ and $\{\boldsymbol{ii}\}_{i=1}^{N}$ are both orthogonal set of vectors. `AsymML`, however, does not require such orthogonality. Therefore, SVD can be then relaxed as

$$\begin{aligned} \overline{X}^{(\mathrm{T})} &= \underset{\overline{X}^{(\mathrm{T})}}{\arg\min} \quad \left\| \overline{X} - \overline{X}^{(\mathrm{T})} \right\|_F^2, \\ \text{s.t. } &\operatorname{rank}\left(\overline{X}^{(\mathrm{T})}\right) \leq \mathcal{R}, \quad \overline{X}^{(\mathrm{U})} = \overline{X} - \overline{X}^{(\mathrm{T})}. \end{aligned} \tag{2.8}$$

Using alternating optimization [13], each component i in $\overline{X}^{(\mathrm{T})}$ can be obtained as described in Algorithm 1[†]. Due to the fast convergence of alternating optimization, given suitable initial values (e.g. output channels from the previous ReLU/Pooling layer), we have experimentally observed that the maximum number of iterations $\mathrm{max\_iter}$ to reach a near-optimal solution $\{\boldsymbol{u}^{(i)}\}_{i=1}^{\mathcal{R}}$ and $\{\boldsymbol{ii}\}_{i=1}^{\mathcal{R}}$ is typically $1 \sim 2$. Finally, it is worth noting that the computation complexity of this algorithm is much less than exact SVD as it only increases linearly with $\mathcal{R}$.

Algorithm 1: Lightweight SVD Approx.

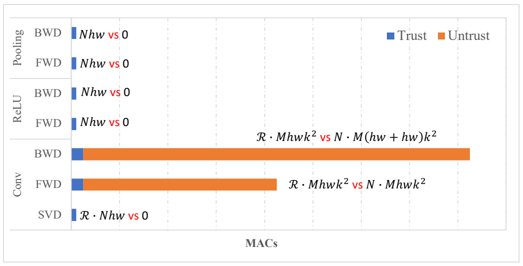
```
Data: $\mathcal{R}, \overline{X}, \left\{\boldsymbol{u}_0^{(i)}, \boldsymbol{ii}_0 \quad | \quad i = 1, \cdots, \mathcal{R}\right\}$, max_iter
Result: $\overline{X}^{(\mathrm{T})}, \overline{X}^{(\mathrm{U})}$
Initialize $\overline{X}^{(\mathrm{T})}$ as 0;
for $i$ in $1, \cdots, \mathcal{R}$ do
    for $j$ in $1, \cdots$, max_iter do
        /* Alternating optimization                                   */
        $\boldsymbol{u}_j^{(i)} = \frac{\overline{X} \cdot \boldsymbol{ii}_{j-1}}{\|\boldsymbol{ii}_{j-1}\|_F^2}$;
        $\boldsymbol{ii}_j = \frac{\overline{X}^* \cdot \boldsymbol{u}_j^{(i)}}{\left\|\boldsymbol{u}_j^{(i)}\right\|_F^2}$;
    end
    $\overline{X}^{(\mathrm{T})} = \overline{X}^{(\mathrm{T})} + \boldsymbol{u}_j^{(i)} \cdot \boldsymbol{ii}_j^*$;
    $\overline{X} = \overline{X} - \boldsymbol{u}_j^{(i)} \cdot \boldsymbol{ii}_j^*$;
end
$\overline{X}^{(\mathrm{U})} = \overline{X}$;
```

2.2.4 Overhead analysis

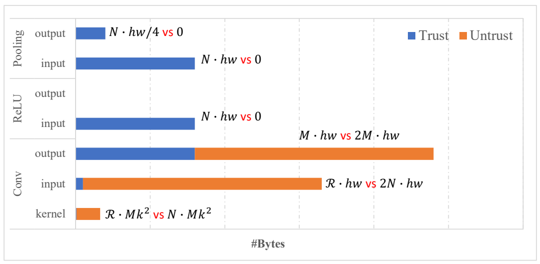
The computation and memory costs in TEEs are of great concern as they decide `AsymML`'s performance in a heterogeneous system. In this section, we break down these costs and compare the costs in TEEs and GPUs.

Figure 2.3 shows computation and memory costs in TEEs and GPUs for the case where $\mathcal{R}/N = 1/16$. As described in Section 2.2.2, the computation complexity of $\mathrm{CONV_{priv}}$ in TEEs increases with the number

[†]In the actual implementation, $\overline{X}^{(\mathrm{T})}$ is stored as a list of vectors $\left\{\boldsymbol{u}^{(i)}, \boldsymbol{ii} \quad | \quad i = 1, \cdots, \mathcal{R}\right\}$, rather than as a matrix.



(a) Computation in TEEs and GPUs



(b) Memory in TEEs and GPUs

Figure 2.3: The computation and the memory costs in TEEs (Trusted) and GPUs (Untrusted) ($\frac{\mathcal{R}}{N} = \frac{1}{16}$, pooling kernel size is 2)

of principal channels $\mathcal{R}$, while the complexity of CONV_{pub} in GPUs is the same as that of the original convolutional layer. In addition to $\text{CONV}_{\text{priv}}$, all element-wise operations (ReLU, Pooling, etc) and the lightweight SVD are performed in TEEs. As for the memory cost, the inputs and outputs of all element-wise operations are stored in TEEs, together with the inputs X^{main} and the outputs of the convolution $\text{CONV}_{\text{priv}}$, Y^{main}. On the other hand, inputs $\mathcal{M}(X^{\text{res}})$, outputs Y^{res} of CONV_{pub} and the corresponding gradients $\nabla_X \mathcal{L}$ and $\nabla_Y \mathcal{L}$ are stored in GPUs.

2.3 Theoretical Guarantees: Low-Rank Structure in NNs

We first define a metric termed as *SVD-channel entropy* to formally quantify the low-rank structure in the intermediate features. Then, we show how the SVD-channel entropy changes in a DNN model after each layer. Inspired by SVD entropy [3] and Rényi entropy [45], we define SVD-channel entropy based on singular values obtained in (2.3). Given the singular values $\{s_p(X) |\, p = 1, \cdots, N\}$ of an input matrix X with N channels, the SVD-channel entropy of X, denoted by μ_X, is defined as follows.

Definition 1. *(SVD-Channel Entropy). The SVD-channel entropy of a matrix X is given by*

$$\mu_X = -\log\left(\sum_{j=1}^{N} \bar{s}_j^2(X)\right), \tag{2.9}$$

where $\bar{s}_j(X) = s_j(X)/\sum\limits_{p=1}^{N} s_p(X)$ *is the* j*-th normalized singular value.*

Next, we show in Lemma 1 and Theorem 1 that μ_X defined above indicates the number of *principal* channels actually needed to approximately reconstruct the original data X.

Lemma 1. *The SVD-channel entropy of an input* X *with* N *input channels is bounded as* $0 \leq \mu_X \leq \log N$.

Proof. Let $Q = \sum_{i=1}^{N} s_i$ and $P = \sum_{i=1}^{N} s_i^2$, in which $s_1 \geq s_2 \geq \cdots \geq s_N$. We note that Q satisfies

$$\sqrt{P} \leq Q \leq \sqrt{NP}, \tag{2.10}$$

where the left equality holds when all singular values are zeros except s_1 and the right equality holds when all singular values are equal. According to Definition 1, we have

$$\begin{aligned}\mu_X &= -\log\left(\sum_{i=1}^{N} \bar{s}_i^2(X)\right) = -\log\left(\sum_{i=1}^{N} \frac{s_i^2}{Q^2}\right) \\ &= -\log(\sum_{i=1}^{N} s_i^2) + \log Q^2 = 2\log Q - \log P.\end{aligned}$$

Finally, from above equation, we have $0 \leq \mu_X \leq \log N$. □

If we use the first $\lceil 2^{\mu_X} \rceil$ most principal channels to reconstruct X, then Theorem 1 shows that such a reconstruction is sufficient to approximate X.

Theorem 1. *Given a matrix* X *with SVD-channel entropy* μ_X*, and assuming the* j*-th singular value is given as* $s_j(X) = a \cdot b^{j-1}$*, for some constants* $a > 0$, $0 < b < 1$*, if we use the* $\mathcal{R} = \lceil 2^{\mu_X} \rceil$ *most principal channels to reconstruct* X*, then we have* $\frac{\sum_{j=1}^{\mathcal{R}} s_j^2(X)}{\sum_{j=1}^{N} s_j^2(X)} \geq 0.97$.

Proof. Let $Q = \sum_{i=1}^{N} s_i$ and $P = \sum_{i=1}^{N} s_i^2$, and assume that $s_1 > s_2 > \cdots > s_N$. According to the assumption, the i-th singular value is given as $s_i = a \cdot b^{i-1}$, where $a > 0, 0 < b < 1$. Hence, we have

$$
\begin{aligned}
Q &= \sum_{i=1}^{N} s_i = a \cdot \sum_{i=1}^{N} b^{i-1} = \frac{1-b^N}{1-b}, \\
P &= \sum_{i=1}^{N} s_i^2 = a^2 \cdot \sum_{i=1}^{N} b^{2(i-1)} = \frac{1-b^{2N}}{1-b^2}, \\
2^{\mu_X} &= \frac{Q^2}{P} = \frac{(1+b)(1-b^N)}{(1-b)(1+b^N)}.
\end{aligned}
$$

Let $\eta = \frac{\sum_{i=1}^{\mathcal{R}} s_i^2}{\sum_{j=1}^{N} s_j^2}$, which can be lower-bounded as follows

$$
\eta(b, N) = \frac{1-b^{2\mathcal{R}}}{1-b^{2N}} \geq \frac{1-b^{2\cdot 2^{\mu_X}}}{1-b^{2N}} = \frac{1-b^{\frac{2Q^2}{P}}}{1-b^{2N}} = \frac{1-b^{\frac{2(1+b)(1-b^N)}{(1-b)(1+b^N)}}}{1-b^{2N}}.
$$

Finally by minimizing the function $\eta(b, N)$, we get a minimum of 0.97. Therefore, with $\mathcal{R} = \lceil 2^{\mu_X} \rceil$ principal channels, the total energy in the reconstructed data is at least 97% of the total energy in the original data X. $\square$

Remark 1. The assumption that $s_j(X) = a \cdot b^{j-1}$ usually holds in natural images, where the data have highly correlated channels. In such cases, the singular values usually decay exponentially [31].

2.3.1 SVD-channel entropy in CNNs

In DNNs, given input data with low-rank structure, it is crucial to measure how such structure (measured using SVD-channel entropy) changes after every layer so that we can systematically set the number of principal channels in TEEs. In CNNs, convolutional, batch normalization, pooling and non-linear layers such as ReLU are the basic layers. In this section, we mainly analyze how these basic operators change the structure of the data. All proofs are provided in Appendix A.

We now start with the convolutional layers.

Convolutional layer – For a convolutional layer with input $X \in R^{N\times h\times w}$, we first bound the SVD-channel entropy of the outputs with 1×1 kernels in Theorem 2 and then we extend this to the general case of $k \times k$ kernels in Theorem 3.

Theorem 2. *For a convolution layer, given input $X \in R^{N\times h\times w}$ with SVD-channel entropy μ_X, kernel $W \in R^{M\times N\times 1\times 1}$, then the SVD-channel entropy of the output $Y \in R^{M\times h^{'}\times w^{'}}$ is upper-bounded as follows*

$$\mu_Y \leq \log \lceil 2^{\mu_X} \rceil.$$

Proof. Let $\mathcal{R} = \lceil 2^{\mu_X} \rceil$, then $X_j = \sum_{p=1}^{\mathcal{R}} a_{j,p} \cdot X_p^{'}$, where $X_p^{'}$ is the p-th principle channel of X, and $\overline{X^{'}}_p$ denote a flatten vector from $X_p^{'}$. Then, the i-th output channel is given by

$$Y_i = \sum_{j=1}^{N} W_{i,j} \circledast X_j = \sum_{p=1}^{\mathcal{R}} (\sum_{j=1}^{N} W_{i,j} \cdot a_{j,p}) \circledast X_p^{'}.$$

All channels in Y can be then written as follows

$$Y = \{Y_1, \cdots, Y_M\} = \sum_{p=1}^{\mathcal{R}} \left\{ (\sum_{j=1}^{N} W_{1,j} \cdot a_{j,p}) \circledast X_p^{'}, \cdots \right\}.$$

Let

$$Y_p^{'} = \left\{ (\sum_{j=1}^{N} W_{1,j} \cdot a_{j,p}), \cdots, (\sum_{j=1}^{N} W_{M,j} \cdot a_{j,p}) \right\} \circledast X_p^{'},$$

then $Y = \sum_{p=1}^{\mathcal{R}} Y_p^{'}$. Since $W_{i,j}$ is a 1×1 kernel, $Y_p^{'}$ can be written as

$$Y_p^{'} = \left\{ (\sum_{j=1}^{N} W_{1,j} \cdot a_{j,p}), \cdots, (\sum_{j=1}^{N} W_{M,j} \cdot a_{j,p}) \right\} \cdot X_p^{'}.$$

For any two principal channels, $X'_{p_1}, X'_{p_2}, p_1 \neq p_2$, $\left\langle \overline{X'}_{p_1}, \overline{X'}_{p_2} \right\rangle = 0$. Therefore, Y'_{p_1}, Y'_{p_2} are constructed by two orthogonal channels. Y only has at most $\mathcal{R}$ principle channels. Therefore, $\mu_Y \leq \log \mathcal{R} = \log \lceil 2^{\mu_X} \rceil$.

$\square$

Theorem 2 implies that low-rank structure is still preserved in the outputs for convolutional layers with 1×1 kernels. Therefore, before a convolutional layer, if $\lceil 2^{\mu} \rceil$ principal channels are used in TEEs, the same number of channels is still sufficient to approximate the outputs.

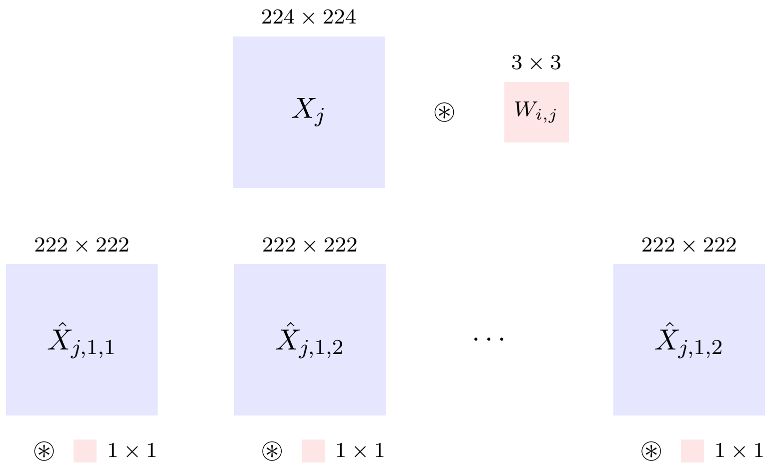


Figure 2.4: An illustration of the conversion from $k \times k$ convolution to 1×1 convolutions with k^2 different input channels.

For a convolutional layer with a $k \times k$ kernel, it can be viewed as a 1×1 convolutional layer with k^2 different "input channels". Each "input channel" is a patch of X_j, denoted as $\left\{ \hat{X}_{j,q,r} | 1 \leq q, r \leq k \right\}$, as shown in Figure 2.4. Given $X \in R^{N \times h \times w}$, $W \in R^{M \times N \times k \times k}$, the i-th output channel can be rewritten as follows

$$Y_i = \sum_{j=1}^{N} W_{i,j} \circledast X_j = \sum_{j=1}^{N} \sum_{q=1,r=1}^{k,k} W_{i,j,q,r} \circledast \hat{X}_{j,q,r}, \tag{2.11}$$

where $W_{i,j,q,r}$ is a 1×1 kernel after this conversion.

Therefore, a $k \times k$ convolution with N input channels is equivalent to a 1×1 convolution with $N \cdot k^2$ "input channels": $\left\{\hat{X}_{j,q,r}|1 \le j \le N, 1 \le q, r \le k\right\}$. To simplify the notation, we denote $\hat{X}_{:,q,r}$ as the (q,r)-th patch of all channels, and $\hat{X}_{j,:,:}$ as all patches of the j-th channel.

We now show in Theorem 3 that, knowing the SVD-channel entropy of $\hat{X}_{j,:,:}$ for $j = 1, \cdots, N$, the SVD-channel entropy of the outputs with $k \times k$ convolution can be accordingly bounded.

Theorem 3. *Given data X with SVD-channel entropy μ_X, $\mathcal{R} = \lceil 2^{\mu_X} \rceil$, $\mu_{\hat{X}_{j,:,:}} = \hat{\mu}_j$ for $1 \le j \le N$ and WLOG suppose that $\hat{\mu}_1 \le \hat{\mu}_2 \le \cdots \le \hat{\mu}_N$, then the SVD-channel entropy of $\hat{X}$ satisfies*

$$\mu_{\hat{X}} \le \log(\sum_{j=1}^{\mathcal{R}} \left\lceil 2^{\hat{\mu}_j} \right\rceil) \cong \mu_X + \bar{\mu},$$

where $\bar{\mu} = \frac{\sum_{j=1}^{\mathcal{R}} \hat{\mu}_j}{\mathcal{R}}$. Furthermore, the SVD-channel entropy of the output Y after convolution with $W \in R^{M \times N \times k \times k}$ is upper-bounded as

$$\mu_Y \le \log(\textstyle\sum_{j=1}^{\mathcal{R}} \left\lceil 2^{\hat{\mu}_j} \right\rceil)$$

Proof. Flatten $\hat{X}_{:,q,r}$, X as $\overline{\hat{X}}_{:,q,r}$ and $\overline{X}$. Then, $\overline{\hat{X}}_{:,q,r}$ can be regarded as $\overline{X}$ with at most $k^2 - (\frac{k+1}{2})^2$ columns reset as zeros. We use a set S_0 to denote these columns. Let $\mathcal{R} = \log \lceil 2^{\mu_X} \rceil$, then each row in $\overline{X}$ can be written as $\overline{X}_j = \sum_{p=1}^{\mathcal{R}} a_{j,p} \overline{X}_p^{'}$. Therefore, each row in $\overline{\hat{X}}_{:,q,r}$ can be written as $\overline{\hat{X}}_{j,q,r} = \sum_{p=1}^{\mathcal{R}} a_{j,p} \overline{\hat{X}}_{p,q,r}^{'}$, where $\overline{\hat{X}}_{p,q,r}^{'}$ is the same as $\overline{X}_p^{'}$ except for values in columns S_0 are zeros. Therefore, $\overline{\hat{X}}_{:,q,r}$ has at most $\mathcal{R}$ principle components. Namely, $\hat{X}_{:,q,r}$ has at most $\mathcal{R}$ principle channels. Hence, $\mu_{\hat{X}_{:,q,r}} \le \mathcal{R} = \log \lceil 2^{\mu_X} \rceil$. □

Based on Theorem 3, with $k \times k$ kernels, the SVD-channel entropy of the outputs increases with $\mu_{\hat{X}_{j,:,:}}$ for $j = 1, \cdots, \mathcal{R}$. Intuitively, if k is larger, more patches are included, then $\mu_{\hat{X}_{j,:,:}}$ will be higher. More numerical analyses are presented in Section 2.4.2.

BatchNorm layer – Batch normalization is commonly used in CV models to reduce the *internal covariate shift.* According to Theorem 4, the SVD-channel entropy of outputs is almost the same as that of inputs in a BatchNorm layer.

Theorem 4. *Given data X with SVD-channel entropy μ_X, then the SVD-channel entropy of output Y after a batch normalization layer is upper-bounded as*

$$\mu_Y \leq \log(\lceil 2^{\mu_X} \rceil + 1).$$

Proof. Let $\mathcal{R} = \lceil 2^{\mu_X} \rceil$, then input $X_j = \sum_{p=1}^{\mathcal{R}} a_{j,p} \cdot X_p^{'}$, where $X^{'}$ is the principle channels of X. With batch normalization operator, we have

$$Y_j = \frac{X_j - E[X_j]}{\sqrt{V[X_j] + \epsilon}} \cdot \gamma_i + \beta_i,$$

where $E[X_j]$ and $V[X_j]$ is the mean and variance in channel j across batches, γ_j and β_j are learnable parameters in BatchNorm layers and ϵ is a constant for numerical stability. We can then re-write Y_i in terms of $X_p^{'}$ as

$$\begin{aligned} Y_i &= \frac{\sum_{p=1}^{\mathcal{R}} a_{j,p} \cdot X_p^{'} - E[X_j]}{\sqrt{V[X_j] + \epsilon}} \cdot \gamma_j + \beta_j \\ &= \sum_{p=1}^{\mathcal{R}} \frac{\gamma_j a_{j,p}}{\sqrt{V[X_j] + \epsilon}} X_p^{'} - \frac{\gamma_j E[X_j]}{\sqrt{V[X_j] + \epsilon}} + \beta_j \end{aligned}$$

$$= \sum_{p=1}^{\mathcal{R}} \frac{\gamma_j a_{j,p}}{\sqrt{V[X_j] + \epsilon}} X_p^{'} - \left(\frac{\gamma_j E[X_j]}{\sqrt{V[X_j] - \epsilon}} - \beta_j\right) \cdot \mathbf{1}$$

Therefore, the output channel Y_j can be reconstructed by at most $\mathcal{R}$ principal channels in X, plus a constant vector $\mathbf{1}$. Hence $\mu_Y \leq \log \lceil 2^{\mu_X} \rceil + 1$. $\square$

Hence, the low-rank structure of inputs is still preserved after batch normalization.

ReLU layer – As a non-linear layer, ReLU is the most commonly used operator in DNNs. Theoretically bounding the SVD-channel entropy after ReLU is infeasible. Instead, we empirically measure the SVD-channel entropy after a ReLU and show that the ReLU layers also preserve the low-rank structure.

Pooling layer – Pooling operations include max and average pooling. Max pooling is a also non-linear operator. Similarly, we provide an empirical experiment that shows that a max pooling layer does not greatly change the low-rank structure either. Besides, the max and the average pooling layers usually deliver similar performance. As stated in Theorem 5, the SVD-channel entropy after an average pooling layer is less than the one before pooling. Therefore, pooling layers still preserve the low-rank structure.

Theorem 5. *For an average pooling layer, given input $X \in R^{N\times h\times w}$ with SVD-channel entropy μ_X, the SVD-channel entropy in output Y satisfies*

$$\mu_Y \leq \log \lceil 2^{\mu_X} \rceil .$$

Proof. Let $\mathcal{R} = \lceil 2^{\mu_X} \rceil$, then input $X_j = \sum_{p=1}^{\mathcal{R}} a_{j,p} \cdot X_p^{'}$, where $X^{'}$ is the principle channels of X. With $k \times k$ average operator, we have

$$\begin{aligned} Y_i(h,w) &= \frac{1}{k^2}\sum_{h^{'}=1}^{k}\sum_{w^{'}=1}^{k} X_i((h-1)k+h^{'},(w-1)k+w^{'}) \\ &= \frac{1}{k^2}\sum_{h^{'}=1}^{k}\sum_{w^{'}=1}^{k}\sum_{p=1}^{\mathcal{R}} a_{i,p}X_p^{'}((h-1)k+h^{'},(w-1)k+w^{'}) \\ &= \sum_{p=1}^{\mathcal{R}} a_{i,p}\cdot\frac{\sum_{h^{'},w^{'}=1}^{k} X_p^{'}((h-1)k+h^{'},(w-1)k+w^{'})}{k^2} \end{aligned}$$

Therefore, each channel in Y can be seen as an accumulation of $\mathcal{R}$ principle channels obtained by conducting average pooling on $X^{'}$. Therefore, Y can be constructed by at most $\mathcal{R}$ principle channels. Hence, $\mu_Y \leq \log\mathcal{R} = \log\lceil 2^{\mu_X}\rceil$ □

2.4 Empirical Evaluation

In this section, we evaluate AsymML in terms of the training accuracy, running time, robustness against attacks and information leakage. We perform our experiments on the following models and datasets.

For models, we consider VGG-16, VGG-19 [84], ResNet-18 and ResNet-34 [39]. For datasets, we consider the CIFAR-10 [51] and the ImageNet [23] datasets.

We illustrate the implementation of AsymML in detail in Section 2.4.1. Then, in Section 2.4.2, we study the effect of the kernel size on the SVD-channel entropy. In Section 2.4.3, we compare between AsymML and the baselines in terms of the running time.

2.4.1 Implementation

We implement AsymML in a heterogeneous system with an Intel SGX enabled Xeon CPUs [21] and NVIDIA RTX5000 GPUs working as untrusted accelerators. AsymML first reads a predefined model (e.g. from PyTorch), decomposes it into trusted and untrusted parts, and then offloads them to TEEs and GPUs respectively. The operations in TEEs (e.g. $\text{CONV}_{\text{priv}}$, ReLU, and Pooling) are supported by dedicated trusted functions. In order to reduce the CPU-GPU communications, a Pooling is fused with the preceding ReLU layer. Similar optimization also is applied to convolutional layers and the following batch normalization layers.

During training, SGX and GPU contexts are created for the trusted and the untrusted operations. We use PyTorch as a high-level coordinator to distribute the computations, activate GPU/SGX context, compute loss, and update the model parameters.

During a forward pass, outputs of a convolutional layer in GPUs and TEEs will be merged in the following ReLU (and Pooling) layer in TEEs. A lightweight SVD is then applied to decompose the output activations into low-rank and residual parts, which are then fed into the next convolution layer.

During a backward pass, computing $\nabla_X \mathcal{L}$ for ReLU and Pooling layers is performed in TEEs, while computing $\nabla_X \mathcal{L}$ for convolutional layers is performed in GPUs. The partial gradients $\nabla^{(\mathrm{T})}_{W_{i,j}} \mathcal{L}$ and $\nabla^{(\mathrm{U})}_{W_{i,j}} \mathcal{L}$ in a convolutional layer are computed in GPUs and TEEs respectively and merged into TEEs.

2.4.2 SVD-channel entropy vs kernel size

In this subsection, we investigate the effect of the kernel size on the SVD-channel entropy. As Theorem 3 states, the upper bound of the SVD-channel entropy of outputs in a convolutional layer increases with the kernel size. More patches are generated with large kernels in each channel (See Fig. 2.4), therefore this leads to increasing SVD-channel entropy along the patches ($\mu_{\hat{X}_{j,:,:}}$).

We present an empirical result on the SVD-channel entropy $\mu_{\hat{X}_{j,:,:}}$ with kernel sizes ranging from 1 to 11. In this experiment, the input data X are randomly sampled from ImageNet. As detailed in Section 2.3, for $k \times k$ kernels, k^2 input patches are generated. We compute the SVD-channel entropy along all patches.

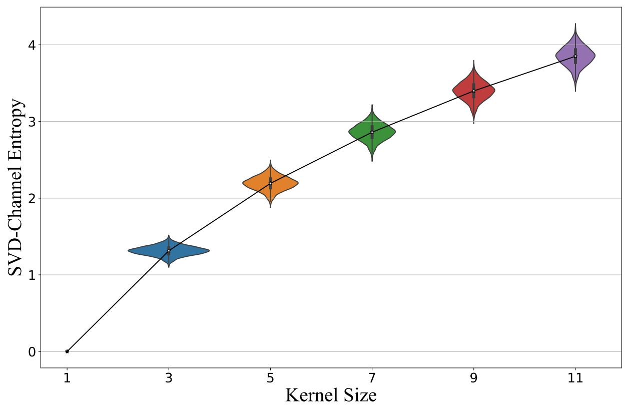


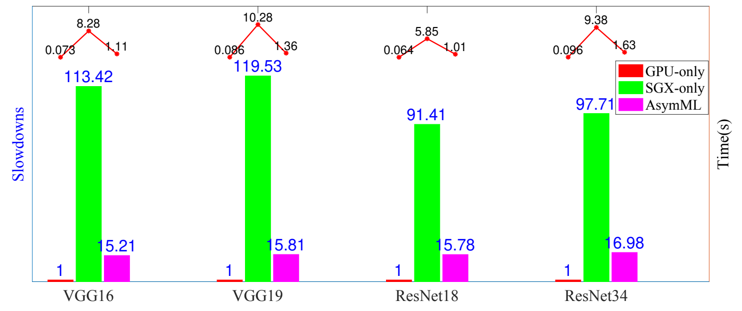
Figure 2.5: The SVD-channel entropy along patches of a images is shown. The line plot shows the mean value, while the violin plot shows the distribution. A wide violin indicates large number of images are around a particular SVD-channel entropy level.

Fig. 2.5 shows the SVD-channel entropy across 10K randomly sampled images with different kernel size. The line plot indicates the mean value, while the violin plots show the distribution. With 3×3 kernels, the SVD-channel entropy increases by around 1, which implies that given $\mathcal{R}$ principal channels in inputs,

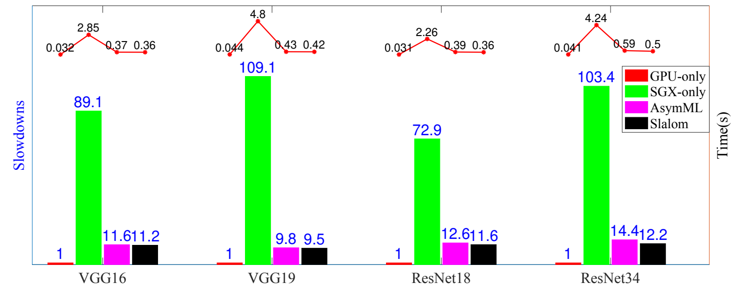
$2\mathcal{R}$ principal channels are sufficient for outputs after a convolution layer with 3×3 kernel. Although it is noted that the SVD-channel entropy increases with the kernel size, small kernel sizes are commonly used in modern DNNs such as VGG and ResNet. Therefore, the SVD-channel entropy increment is well-managed.

2.4.3 Runtime analysis

We conduct training and inference on ImageNet and compare the runtime with two baselines: GPU-only, TEE-only. Following the standard data pre-processing procedure, we resize each image into $3 \times 224 \times 224$. The batch size is set to 32. We compute the runtime as the average time of a forward and backward pass.



(a) The training time of `AsymML` compared to the SGX-only and the GPU-only executions is shown.



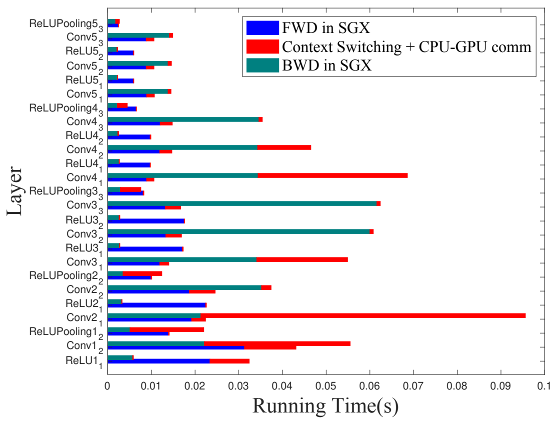
(b) The inference time of `AsymML` compared to the SGX-only execution, the GPU-only execution and Slalom is shown.

Figure 2.6: Training and Inference performance on ImageNet. Bar plot shows slowdowns compared to GPU-only execution; red dotted lines are the corresponding running time.

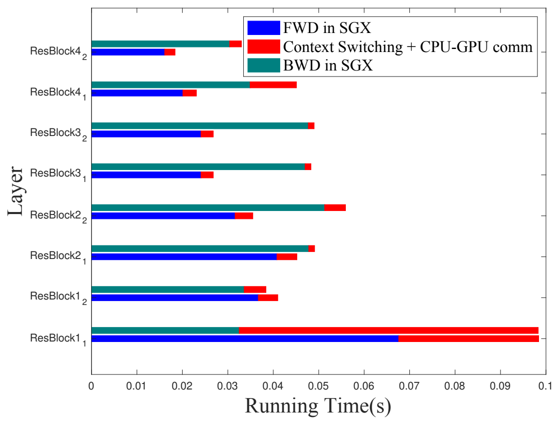
Training runtime. For training, we compare the runtime performance with the TEE-only and the GPU-only executions. Fig. 2.6a shows the actual runtime (red plots) and the relative slowdowns (bar plots) compared to the GPU-only execution. Compared to the TEE-only method, `AsymML` achieves $7.5 - 7.6\times$

speedup on VGG-16/VGG-19, and up to $5.8 - 5.8\times$ speedup on ResNet-18/34. By encoding most information in TEEs with very low-rank representations, `AsymML` achieves significant performance gains compared to TEE-only executions. Although `AsymML` shows around $15\times$ slowdown compared to *untrusted* GPU-only execution, `AsymML` protects the privacy of the training datasets unlike the GPU-only execution.

Inference runtime. We also compare `AsymML` to Slalom [89] in terms of inference time. While a part of the convolution is performed in TEEs, inference using `AsymML` is just slightly slower than Slalom. Therefore, the low-rank representation in TEEs does not incur significant additional costs.



(a) The runtime of VGG-16 is shown.



(b) The runtime of ResNet-18 is shown.

Figure 2.7: An illustration of the runtime breakdown in VGG-16 and ResNet-18 is shown.

Runtime breakdown. To better identify the main bottlenecks on a heterogeneous platform, we profile the running time of the forward and backward passes in `AsymML`. We break down the running time into computation time and communication time. We show the runtime breakdown for VGG-16 and ResNet-18 in Fig. 2.7.

- **Forward pass**. For the forward pass, the blue bars show the time spent in TEEs, while the red bars show CPU-GPU communication. In the early convolutional layers, due to the large number of features, the communication from GPUs to CPUs brings notable costs, which then becomes marginal in later layers.

- **Backward pass.** For the backward pass, the green bars show the time in TEEs, and the red bars show CPU-GPU communication. The additional cost for CPU-GPU communication is much more dominant, especially in early convolution layers. The reason is that during the backward pass, not only are input gradients $\nabla_X \mathcal{L}$ but also parameter gradients $\nabla_{W_{i,j}} \mathcal{L}$ transferred between CPUs and GPUs.

2.5 Conclusion

In this paper, we have proposed an asymmetric decomposition framework, `AsymML`, to decompose DNN models and offload computations onto trusted and untrusted fast hardware. The trusted part aims to preserve the main information in the data with manageable computation cost, and the untrusted part undertakes most computations. In such a way, `AsymML` makes the best use of each platform in a heterogeneous setting. We have then presented a theoretical analysis showing that the low-rank structure is preserved in NNs. Our extensive experiments show that AsymML achieves gain up to $7.6\times$ in training. We also show that `AsymML` provides strong privacy protection under model and gradient inversion attacks.

Chapter 3

Delta: Private Learning with Asymmetric Flows

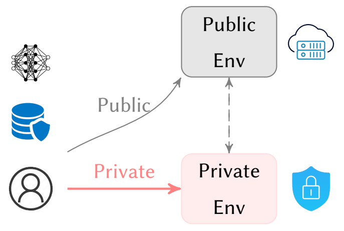
Data privacy is of great concern in cloud machine-learning service platforms, when sensitive data are exposed to service providers. While private computing environments (e.g., secure enclaves), and cryptographic approaches (e.g., homomorphic encryption) provide strong privacy protection, their computing performance still falls short compared to cloud GPUs. To achieve privacy protection with high computing performance, we propose `Delta`, a new private training and inference framework, with comparable model performance as non-private centralized training. `Delta` features two asymmetric data flows: the main information-sensitive flow and the residual flow. The main part flows into a small model while the residuals are offloaded to a large model. Specifically, `Delta` embeds the information-sensitive representations into a low-dimensional space while pushing the information-insensitive part into high-dimension residuals. To ensure privacy protection, the low-dimensional information-sensitive part is secured and fed to a small model in a private environment. On the other hand, the residual part is sent to fast cloud GPUs, and processed by a large model. To further enhance privacy and reduce the communication cost, `Delta` applies a random binary quantization technique along with a DP-based technique to the residuals before sharing them with the public platform. We theoretically show that `Delta` guarantees differential privacy in the public environment and greatly reduces the complexity in the private environment. We conduct empirical analyses on CIFAR-10, CIFAR-100 and ImageNet datasets and ResNet-18 and ResNet-34, showing that Delta achieves strong privacy protection, fast training, and inference without significantly compromising the model utility.

3.1 Introduction

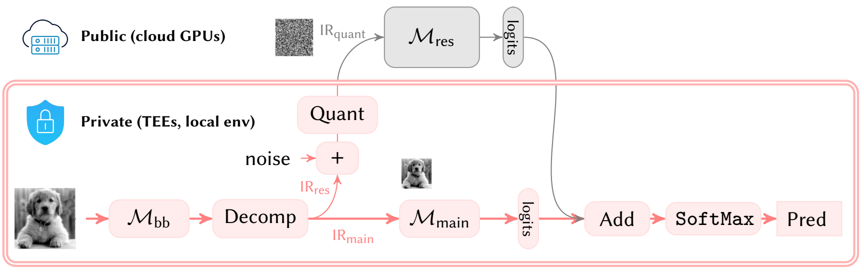
In the current machine learning (ML) era, cloud ML services with high-end GPUs have become indispensable. On the other hand, ensuring data privacy is one of the most critical challenges in the ML platforms. During training, privacy breaches may occur if training data is exposed to ML service providers, increasing vulnerability to potential attacks. Additionally, users' inference queries can also be susceptible to attacks when accessing ML services with sensitive data [72, 65]. In particular, an untrusted ML platform can cache, learn, and leak queries without users' awareness [73].

Related Works Overview. While prior privacy-preserving machine learning (PPML) frameworks [1, 86, 67, 62, 71, 8] mitigate privacy concerns in training and inference, they also come with different tradeoffs. Differential privacy (DP) based methods perturb the data before outsourcing to an untrusted cloud [66, 103, 98] to ensure privacy, but they usually result in degraded model utility even under moderate privacy constraints [9]. On the other hand, crypto-based techniques [86, 67], that provide data protection with encryption schemes, have not yet proved efficient and scalable to large models due to their prohibitive complexities.

PPML with private environments (e.g., trusted execution environments (TEEs), local environments) presents a promising solution by physically isolating the running computing environments. Such private environments are, however, usually resource-constrained compared to public cloud services with high-end GPUs, resulting in lower computing performance [76, 34]. However, the prior works based on leveraging these private environments along with the public GPUs also incur high complexity [89, 70, 69, 36]. One reason for these methods being inefficient is the heavy communication between the private and the public environments.



(a) Problem Setup.



(b) An overview of `Delta`.

Figure 3.1: Overview of `Delta`: the backbone $\mathcal{M}_{\text{bb}}$ acts as a feature extractor. The features are decomposed into low-dimensional (information-sensitive) and high-dimensional (residual) parts: IR_{main} and IR_{res}. IR_{main} is fed to a small model $\mathcal{M}_{\text{main}}$, while IR_{res} are outsourced to a large model $\mathcal{M}_{\text{res}}$. $\mathcal{M}_{\text{bb}}$ and $\mathcal{M}_{\text{main}}$ run in a resource-constrained private environment, whereas $\mathcal{M}_{\text{res}}$ is offloaded to a public environment while ensuring privacy through a DP scheme. While only the forward pass is shown, backpropagation is also private (See Sec 3.4.4).

Proposed Solution Overview. In this paper, we *consider a private training and inference setting where users can access both private and public environments*, as shown in Figure 3.1a. We propose a new private training and inference framework, `Delta`, that achieves strong privacy protection with high model utility and low complexity. The core idea of `Delta` originates from an observation that the intermediate representations (IRs) in ML models exhibit an *asymmetric* structure. Specifically, the primary sensitive information is usually encoded in a low-dimensional space, while the high-dimensional residuals contain very little information.

Inspired by such an observation, we design a two-way training framework that respectively learns low-dimensional but information-sensitive features IR_{main} with a *small* model $\mathcal{M}_{\text{main}}$, and learns the high-dimensional residuals IR_{res} with a *large* model $\mathcal{M}_{\text{res}}$, as illustrated in Figure 3.1b. Given a model, `Delta` selects a few front layers as a backbone ($\mathcal{M}_{\text{bb}}$) for extracting features and generating intermediate representations. `Delta` uses singular value decomposition (SVD) and discrete cosine transformation (DCT) to extract the low-dimensional information-sensitive representation IR_{main} and the residuals IR_{res}. We design a new low-dimensional model ($\mathcal{M}_{\text{main}}$) to learn the information-sensitive IR_{main}. While $\mathcal{M}_{\text{bb}}$ and $\mathcal{M}_{\text{main}}$ run in a private environment, the rest of the model ($\mathcal{M}_{\text{res}}$) learns residuals in a public environment. `Delta` further applies DP perturbation and binary quantization on the residuals, leading to further privacy protection and a communication reduction.

Owing to the asymmetric structure in IR, `Delta` guarantees differential privacy on IR_{res} with much smaller noise added compared to the naive scheme that directly adds noise to IR (naive-DP), leading to much-improved model utility. In both training and inference phases, `Delta` ensures that only the residual part is public, while the information-sensitive part is secured in private environments.

`Delta` is a generic PPML solution that can be flexibly deployed in several scenarios. For instance, in a cloud ML platform with private TEEs and public GPUs [7, 6], $\mathcal{M}_{\text{bb}}$ and $\mathcal{M}_{\text{main}}$ can run inside TEEs to preserve privacy, while $\mathcal{M}_{\text{res}}$ runs in GPUs to speed up the computations. In general distributed settings, `Delta` can let clients train $\mathcal{M}_{\text{bb}}$ and $\mathcal{M}_{\text{main}}$ locally, while a cloud server performs side training with $\mathcal{M}_{\text{res}}$.

In summary, our contributions are as follows.

1. We propose a PPML framework for training and inference with a much-improved privacy-utility trade-off compared to the naive-DP methods, a low computing complexity in the private environments and a low communication cost.

2. We design an asymmetric decomposition layer that extracts the low-dimensional information-sensitive representations using SVD and DCT, and design a low-complexity model for resource-constrained computing environments.

3. We provide a formal differential privacy analysis for the proposed framework. In addition, we show empirically that `Delta` provides strong privacy protection against model inversion and membership inference attacks.

4. We conduct comprehensive evaluations, showing that `Delta` leads to a better privacy-utility trade-off than the naive DP-based method that directly adds noise to IR. Specifically, under the same privacy budget, `Delta` improves the accuracy by up to 31%. Moreover, `Delta` greatly reduces the running time compared to other PPML solutions.

3.2 System Model

We start by describing the problem setting, the threat model, and our notations.

Problem Setting. We consider a setting where users have private resource-constrained environments (e.g., cloud TEEs, local CPUs/GPUs), but they can also access public, untrusted, and high-end services (e.g., cloud GPUs) to accelerate training and inference. The goal is to protect users' training and inference data, while maintaining computing performance and high model utility. Note that this setting differs from DP-SGD [1, 53, 102], where the goal is to protect the training data from attacks on gradients or models.

Threat Model. Our threat model is similar to the model considered in prior works leveraging private environments [70, 89, 76]. Specifically, we assume that the private environment is protected against all untrusted, unauthorized access to data and models inside. However, denial-of-service and side-channel attacks are out of our scope. On the other hand, the untrusted, public environment is semi-honest (honest-but-curious). That is, the untrusted public server follows the training and inference protocol faithfully but may attempt to learn as much as possible from what it receives.

Remark 2. *While we only consider the semi-honest model in our work, verifiable computing techniques [32, 2] can be incorporated to enhance Delta against malicious parties.*

Notations. We denote tensors by capital bold letters as $\mathbf{X}$, matrices by capital letters as X, and vectors by small bold letters as $\boldsymbol{x}$. X^i denotes a tensor slice $\mathbf{X}(i,:)$. $\|.\|$ denotes the Frobenius norm or, in general, the square root of the sum of squares of elements in a tensor. "$\cdot$" denotes matrix multiplication or, in general, batch matrix multiplication. "$\circledast$" denotes a convolution operation. X^* indicates a transpose.

3.3 Asymmetric Structure in IRs

We first observe that intermediate representations (IRs) in neural networks (NNs) exhibit highly asymmetric structures in multiple dimensions. These asymmetric structures are essential for an asymmetric decomposition that embeds the privacy-sensitive information into low-dimensional space.

3.3.1 Asymmetric Structure in Channel Dimension

In NNs such as convolutional neural networks (CNNs), each layer's input and output consists of multiple channels, denoted as $\mathbf{X} \in \mathbb{R}^{c \times h \times w}$, where c is the number of channels and h and w denote the height and the width.

We analyze the channel correlation by first flattening $\mathbf{X}$ as $X \in \mathbb{R}^{c \times hw}$ and then performing singular value decomposition (SVD) on the flattened tensor X as

$$X = \sum_{i=1}^{c} s_i \cdot \boldsymbol{u}_i \cdot \boldsymbol{v}_i^*,$$

where s_i is the i-th singular value, $\boldsymbol{u}_i \in \mathbb{R}^c$ and $\boldsymbol{v}_i \in \mathbb{R}^{hw}$ are the i-th left and right singular vectors, respectively. We reshape $\boldsymbol{v}_i$ to the original dimensions as a *principal channel* $V^i \in \mathbb{R}^{h \times w}$, and $\boldsymbol{u}_i$ as a tensor $\mathit{U}^i \in \mathbb{R}^{c \times 1 \times 1}$.

We then obtain a $\underline{\text{l}}$ow-$\underline{\text{r}}$ank representation of $\mathbf{X}$ as follows

$$\mathbf{X}_{\text{lr}} = \sum_{i=1}^{r} s_i \cdot \mathit{U}^i \cdot V^i,$$

where r denotes the number of principal channels in $\mathbf{X}_{\text{lr}} \in \mathbb{R}^{c \times h \times w}$. Figure 3.2a shows the normalized error $\frac{\|\mathbf{X}-\mathbf{X}_{\text{lr}}\|}{\|\mathbf{X}\|}$ versus r after the first convolutional layer in ResNet-18 (based on ImageNet). We observe that $\mathbf{X}_{\text{lr}}$ with a small r is sufficient to approximate $\mathbf{X}$. That is, most information in $\mathbf{X}$ can be embedded into $\mathbf{X}_{\text{lr}}$ in a low-dimensional space. We notice that `3LegRace` [70] also investigated such a property. However, unlike `3LegRace`, we leverage the low-rank property to design a new low-complexity model (See Sec. 3.4.3).

3.3.2 Asymmetric Structure in Spatial Dimension

The asymmetric structure of the IR also exists over the spatial dimension due to correlations among pixels in each channel $X^i \in \mathbb{R}^{h\times w}$. We use discrete cosine transform (DCT) to analyze the spatial correlation. Specifically, we obtain frequency components using $t \times t$ block-wise DCT [44] as,

$$C^i = \mathrm{DCT}(X^i, T) = \left\{ \, C^i_{k,j} = T \cdot X^i_{k,j} \cdot T^* \, \right\}_{k,j=1}^{h/t, w/t},$$

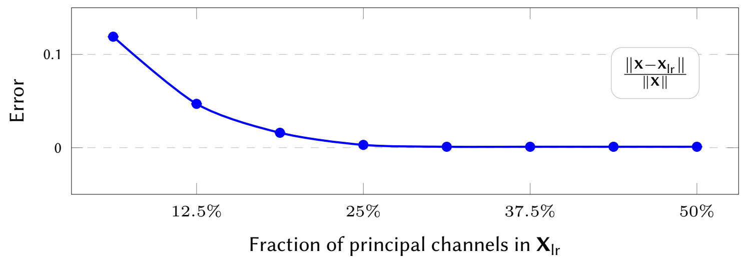
where $C^i_{k,j} \in \mathbb{R}^{t\times t}$, $X^i_{k,j} = X^i(\, kt - t : kt, jt - t : jt \,)$ and $T \in \mathbb{R}^{t\times t}$ is the DCT transformation matrix. $C^i \in \mathbb{R}^{h\times w}$ is obtained by simply concatenating $C^i_{k,j}$ for $k \in 1, ..., h/t, j \in 1, ..., w/t$. Then, we obtain a low-frequency representation, X^i_{lf}, using inverse $t' \times t'$ block-wise DCT as,

$$X^i_{\text{lf}} = \mathrm{IDCT}(C^i_{\text{lf}}, T_{\text{lf}}) = \left\{ \, X^i_{\text{lf},k,j} = T^*_{\text{lf}} \cdot C^i_{\text{lf},k,j} \cdot T_{\text{lf}} \, \right\}_{k,j=1}^{h/t, w/t},$$

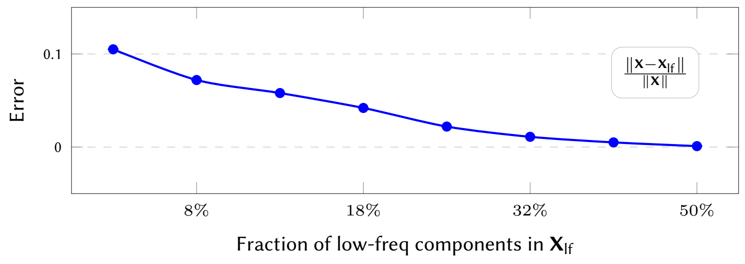
where $t' < t$, $X^i_{\text{lf}} \in \mathbb{R}^{\frac{h}{t}t' \times \frac{w}{t}t'}$, $C^i_{\text{lf},k,j} = C^i_{k,j}(0 : t', 0 : t')$, and $T_{\text{lf}} \in \mathbb{R}^{t'\times t'}$ is the DCT matrix. Note that t'^2 represents the number of low-frequency components in $X^i_{\text{lf},k,j}$.

Owing to the spatial correlation, $\mathbf{X}_{\text{lf}}$ can sufficiently approximate $\mathbf{X}$ using a few low-frequency components. Figure 3.2b shows the error $\frac{\|\mathbf{X}-\mathbf{X}_{\text{lf}}\|}{\|\mathbf{X}\|}$ versus the number of low-frequency components in $\mathbf{X}_{\text{lf}}$ after the first convolution layer in ResNet-18. We can also observe that most information in $\mathbf{X}$ can be embedded into $\mathbf{X}_{\text{lf}}$ with low-frequency components.

The asymmetric structure also exists in other tasks, such as language models. Observing the asymmetric structures in different dimensions, we aim to design an *asymmetric learning* framework that learns privacy-sensitive low-dimensional IR with a low-complexity model in a private environment, while sending residuals to a larger model trained on an untrusted platform.



(a) Error of the low-rank approximation vs. $r/c \times 100\%$.

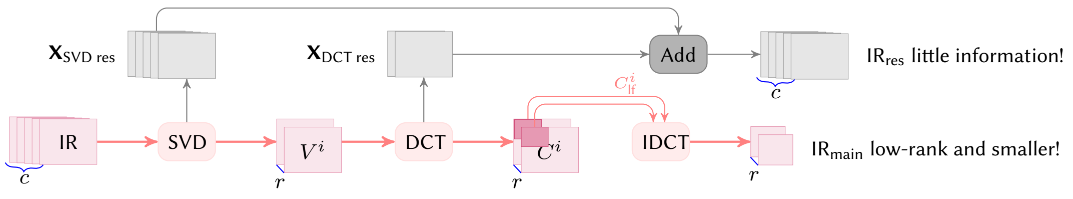


(b) Error of the low-frequency approximation vs. $(t'/t)^2 \times 100\%$.

Figure 3.2: Asymmetric structures along channel and spatial dimension (based on ResNet-18 on ImageNet). Most information in $\mathbf{X}$ can be embedded into low-rank and low-frequency representations.

3.4 Methedology: Private Asymmetric Learning

Overview. At a high level, as shown in Fig. 3.1, given a model, `Delta` keeps the first few layers in a private environment as a backbone model $\mathcal{M}_{\text{bb}}$ and a main model $\mathcal{M}_{\text{main}}$. `Delta` then offloads all remaining layers to a public environment as residual model $\mathcal{M}_{\text{res}}$. Specifically, `Delta` uses the backbone model $\mathcal{M}_{\text{bb}}$ as a feature extractor to compute intermediate representations (IRs). `Delta` then *asymmetrically* decomposes the IRs, obtaining a low-dimensional information-sensitive part (IR_{main}) and residuals IR_{res}. `Delta` designs a new low-dimensional model $\mathcal{M}_{\text{main}}$ for IR_{main} to reduce computation complexity in the private environment. On the other hand, the residuals IR_{res} are perturbed and quantized before being outsourced to the untrusted public environment that hosts $\mathcal{M}_{\text{res}}$. At last, output logits from $\mathcal{M}_{\text{main}}$ and $\mathcal{M}_{\text{res}}$ are added in the private environment, leading to the final predictions. These final predictions are not disclosed to the public environment. Hence, during training and inference, `Delta` allows only minimal residual information to be leaked to the public environment.



$\mathbf{X}_{\text{SVD res}}$
$\mathbf{X}_{\text{DCT res}}$
Add
IR_{res} little information!
c
C^i_{lf}
IR
SVD
V^i
DCT
C^i
IDCT
IR_{main} low-rank and smaller!
c
r
r
r

Figure 3.3: The asymmetric IR decomposition is shown (See Figure 3.1 for the whole pipeline). We use SVD and DCT to encode channel and spatial information into a low-dimensional representation, and offload the residuals to public environments. The low-dimensional IR_{main} has fewer channels and smaller sizes but still encode most sensitive information. The residuals IR_{res} have the same dimension as the original IR.

3.4.1 Asymmetric IR Decomposition

As observed in Section 3.3, IRs after $\mathcal{M}_{\text{bb}}$ exhibit asymmetric structures in multiple dimensions. Hence, we can decompose IRs such that most information is encoded in the low-dimensional IR denoted as IR_{main}.

Singular Value Decomposition (SVD). Given an IR $\mathbf{X} \in \mathbb{R}^{c \times h \times w}$ obtained from the backbone model $\mathcal{M}_{\text{bb}}$, we first apply SVD as explained in Sec. 3.3.1 to obtain the principal channels $\left\{V^i \in \mathbb{R}^{h \times w}\right\}_{i=1}^{c}$ and the corresponding coefficients $\left\{U^i \in \mathbb{R}^{c \times 1 \times 1}\right\}_{i=1}^{c}$. We then select the r most principal channels as a low-rank representation of $\mathbf{X}$ as in Sec. 3.3.1. The rest of the channels are saved as SVD residuals as

$$\mathbf{X}_{\text{SVD res}} = \mathbf{X} - \mathbf{X}_{\text{lr}} = \sum_{i=r+1}^{c} s_i \cdot U^i \cdot V^i. \tag{3.1}$$

Discrete Cosine Transform (DCT). After the decomposition along channels, we further decompose V^i over the spatial dimension using DCT.

Specifically, for each principal channel V^i, we first obtain the frequency-domain coefficients as $C^i = \text{DCT}(V^i, T)$. Then, we only use the low-frequency component to reconstruct a representation as $V^i_{\text{lf}} = \text{IDCT}(C^i_{\text{lf}}, T_{\text{lf}}) \in \mathbb{R}^{\frac{h}{t}t' \times \frac{w}{t}t'}$ as in Sec 3.3.2. C^i_{lf} has a reduced dimension and keeps only top-left low-frequency coefficients in C^i (as shown in Figure 3.3). $T_{\text{lf}} \in \mathbb{R}^{t' \times t'}$ corresponds to DCT transformation matrix with reduced spatial dimension. The rest of the high-frequency components are saved as DCT residuals as

$$V^i_{\text{DCT res}} = \text{IDCT}(C^i_{\text{res}}, T), \tag{3.2}$$

where C^i_{res} denotes C^i with zeros on the top-left corner.

After SVD and DCT, we obtain the privacy-sensitive low-dimensional features as

$$\text{IR}_{\text{main}} = \sum_{i=1}^{r} s_i \cdot U^i \cdot V^i_{\text{lf}}. \tag{3.3}$$

On the other hand, the residuals to be offloaded to the untrusted public environment are given as

$$\begin{aligned} \text{IR}_{\text{res}} &= \mathbf{X} - \text{IR}_{\text{main}} = \mathbf{X}_{\text{SVD res}} + \mathbf{X}_{\text{DCT res}} \\ &= \sum_{i=r+1}^{c} s_i \cdot U^i \cdot V^i \quad + \quad \sum_{i=1}^{r} s_i \cdot U^i \cdot V^i_{\text{DCT res}}. \end{aligned} \tag{3.4}$$

IR_{res} is further normalized as

$$\text{IR}_{\text{res,norm}} = \text{IR}_{\text{res}} / \max(1, \|\text{IR}_{\text{res}}\|_2 / C), \tag{3.5}$$

where C is a scaling parameter for ℓ_2 normalization. The normalization is necessary to bound *sensitivity* for DP. For simplification, we denote the normalized residuals as IR_{res}.

Hence, IR_{main} has fewer principal channels and smaller spatial dimensions but contains most information in IR. The non-principal channels and high-frequency components, on the other hand, are saved in IR_{res}. IR_{main} and IR_{res} are then respectively fed into a small model $\mathcal{M}_{\text{main}}$ in a private environment and a large model $\mathcal{M}_{\text{res}}$ in a public environment.

3.4.2 Residuals Perturbation and Quantization

While the low-dimensional representation IR_{main} has the most important information in the IR, IR_{res} might still contain some information such as a few high-frequency components. Furthermore, as the communication from a private environment (e.g., TEEs) to a public environment (e.g., GPUs) is usually slow, sending floating-point high-dimensional residuals can significantly increase the communication overhead and prolong the total running time.

In this section, we perturb IR_{res} with a Gaussian mechanism and then apply binary quantization on the perturbed IR_{res} to reduce the inter-environment communication cost.

Given a DP budget ϵ, we consider the Gaussian mechanism, and compute the corresponding noise parameter σ (See Section 3.5). For each tensor IR_{res}, we generate a noise tensor $\mathbf{N} \in \mathbb{R}^{c\times h\times w} \sim \mathcal{N}(0, \sigma^2 \cdot \mathbf{I})$, and add it to IR_{res} in the private environment. With noisy residuals, IR_{noisy}=$\text{IR}_{\text{res}} + \mathbf{N}$, we apply a binary quantizer as follows

$$\text{IR}_{\text{quant}} = \text{BinQuant}(\text{IR}_{\text{noisy}}) = \begin{cases} 0 & \text{IR}_{\text{noisy}} < 0, \\ 1 & \text{IR}_{\text{noisy}} \geq 0. \end{cases} \tag{3.6}$$

As a result, the tensor to be offloaded to the public environment is a binary representation of the residuals. Compared to floating-point values, such a binary representation reduces communication by $32\times$. Owing to the asymmetric decomposition, the values in IR_{res} are usually close to zero. Hence, a small noise is sufficient to achieve strong privacy protection (See formal analysis in Section 3.5).

3.4.3 Model Design for Low-Dimensional Representations

Knowing that IR_{main} has a low-rank as given in Equation (3.3), in this section, we show that developing an efficient $\mathcal{M}_{\text{main}}$ with low computation complexity is attainable.

In linear layers such as convolutional layer, low-rank inputs lead to low-rank outputs [70]. Assuming inputs and outputs have rank r and q, we split the convolution layer into two sub-layers as shown in

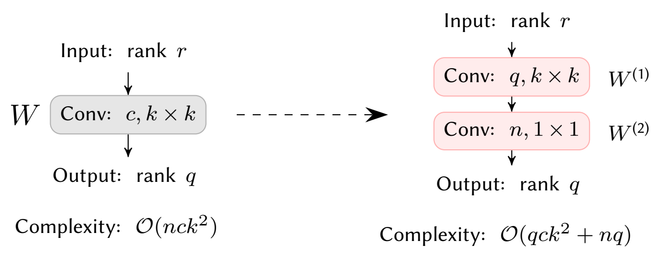


Figure 3.4: Model design for the low-dimensional IR_{main}. Knowing the rank in data, the number of channels in convolution layers can be reduced, leading to a reduction in computation complexity.

Figure 3.4 (right). The first layer has q ($k \times k$) kernels to learn the principal features, whereas the second layer has c (1×1) kernels to combine the principal channels. We further add a kernel orthogonality regularization [96] to the first layer to enhance the orthogonality of the output channels.

Such a design comes with fundamental reasoning. Theorem 6 shows that, with knowledge of the ranks of inputs and outputs in a convolutional layer, it is possible to optimize a low-dimensional layer as in Figure 3.4 (right) such that it results in the same output as the original layer.

Theorem 6. *For a convolution layer with weight $\mathbf{W} \in \mathbb{R}^{n\times c\times k\times k}$ with an input $\mathbf{X}$ with rank r and output $\mathbf{Y}$ with rank q, there exists an optimal $\mathbf{W}^{(1)} \in \mathbb{R}^{q\times c\times k\times k}, \mathbf{W}^{(2)} \in \mathbb{R}^{n\times q\times 1\times 1}$ in the low-dimensional layer such that the output of this layer denoted as $\mathbf{Y}^{'}$ satisfies*

$$\min_{\mathbf{W}^{(1)},\mathbf{W}^{(2)}} \left\| \mathbf{Y} - \mathbf{Y}^{'} \right\| = 0. \tag{3.7}$$

Proof. Given an input tensor $\mathbf{X} \in \mathbb{R}^{c\times h\times w}$ with rank r, $\mathbf{W} \in \mathbb{R}^{n\times c\times k\times k}$, and output $\mathbf{Y} \in \mathbb{R}^{n\times h\times w}$ with rank q, we can write the convolution $\mathbf{Y} = \mathbf{W} \circledast \mathbf{X}$ in a matrix form as

$$Y = W \cdot X, \tag{3.8}$$

where $W \in \mathbb{R}^{n\times ck^2}$, $X \in \mathbb{R}^{ck^2\times hw}$ [90, 70]. Similarly for the low-dimensional convolution layer, we can write its output $\mathbf{Y}'$ as follows

$$Y' = W^{(2)} \cdot W^{(1)} \cdot X,$$

where $W^{(2)} \in \mathbb{R}^{n\times q}$ and $W^{(1)} \in \mathbb{R}^{q\times ck^2}$.

According to Theorem 2 in [70], the output's rank is decided by the rank of X. Therefore, we can decompose X using SVD as

$$X = U \cdot V,$$

where $U \in \mathbb{R}^{ck^2\times q}$ is a matrix with orthogonal columns, and $V \in \mathbb{R}^{q\times hw}$ is a matrix with orthogonal rows. With the low-rank decomposition, we can express the difference between the outputs Y and Y' as

$$\begin{aligned} Y - Y' &= W \cdot U \cdot V - W^{(2)} \cdot W^{(1)} \cdot U \cdot V \\ &= (W \cdot U - W^{(2)} \cdot W^{(1)} \cdot U) \cdot V. \end{aligned}$$

Let $Z = W \cdot U - W^{(2)} \cdot W^{(1)} \cdot U$, owing to the low-rank of the matrix U, the original kernel matrix W is reduced to a low-dimensional form. Therefore, we can express the kernel matrix W as follows $W = W^U + R$, where $W^U = W \times UU^*$ is a low-rank matrix obtained based on the principal components of U, while R is a residual matrix based on the principal components that are orthogonal to U. That is, $R \cdot U = 0$.

Then the difference $Z = W \cdot U - W^{(2)} \cdot W^{(1)} \cdot U$ can be written as follows

$$\begin{aligned} Z &= \left[(W^U + R) - W^{(2)} \cdot W^{(1)}\right] \cdot U \\ &= (W^U - W^{(2)} \cdot W^{(1)}) \cdot U. \end{aligned}$$

Since $\texttt{Rank}\left(W^U\right) \leq q$, then there is a solution such that

$$\min_{W^1, W^2} \left\| W^U - W^{(2)} \cdot W^{(1)} \right\| = 0,$$

where the pair $W^{(1)}, W^{(2)}$ is one of the rank-q decompositions of W^U. Hence, $\min_{W^1, W^2} \left\| \mathbf{Y} - \mathbf{Y}' \right\| = 0$. □

Remark 3. *While the low-dimensional layer in Figure 3.4 (right) shares similar architecture as low-rank compression methods [43, 42, 106], low-rank compression inevitably incurs information loss in outputs. However, the layer in Figure 3.4 (right) can theoretically preserve all information given low-rank inputs and outputs.*

3.4.4 Private Backpropogation

While the asymmetric IR decomposition together with the randomized quantization mechanism ensures privacy in forward passes, information of IR_{main} can still be leaked through backpropagation on logits. In this section, we further propose a private backpropagation that removes the logits of $\mathcal{M}_{\text{main}}$ from the gradients to $\mathcal{M}_{\text{res}}$.

In detail, the gradients to $\mathcal{M}_{\text{main}}$ are calculated through a standard backpropagation algorithm that uses logits from both $\mathcal{M}_{\text{main}}$ and $\mathcal{M}_{\text{res}}$. While gradients to $\mathcal{M}_{\text{res}}$ are calculated solely using $\mathcal{M}_{\text{res}}$ logits. Hence, the logits from $\mathcal{M}_{\text{main}}$ will not be revealed to the outside. Specifically, given the output logits from $\mathcal{M}_{\text{main}}$ and $\mathcal{M}_{\text{res}}$: $\boldsymbol{z}_{\text{main}}$ and $\boldsymbol{z}_{\text{res}}$, we compute $\texttt{Softmax}$ for i-th label $\mathcal{M}_{\text{main}}$ and $\mathcal{M}_{\text{res}}$ as

$$\mathcal{M}_{\text{main}} : \boldsymbol{o}_{\text{tot}}(i) = e^{\boldsymbol{z}_{\text{main}}(i) + \boldsymbol{z}_{\text{res}}(i)} / \sum\nolimits_{j=1}^{L} e^{\boldsymbol{z}_{\text{main}}(j) + \boldsymbol{z}_{\text{res}}(j)},$$
$$\mathcal{M}_{\text{res}} : \boldsymbol{o}_{\text{res}}(i) = e^{\boldsymbol{z}_{\text{res}}(i)} / \sum\nolimits_{j=1}^{L} e^{\boldsymbol{z}_{\text{res}}(j)},$$

where L denotes the number of labels in the current task. Following the backpropagation in $\texttt{Softmax}$, we compute gradients to $\mathcal{M}_{\text{main}}$ and $\mathcal{M}_{\text{res}}$: $\boldsymbol{g}_{\text{main}}$, $\boldsymbol{g}_{\text{res}}$, as

$$\boldsymbol{g}_{\text{main}} = \boldsymbol{o}_{\text{tot}} - \boldsymbol{y}, \quad \boldsymbol{g}_{\text{res}} = \boldsymbol{o}_{\text{res}} - \boldsymbol{y},$$

where $\boldsymbol{y}$ denotes one-hot encoding for labels.

With the separate backpropagation on gradients, $\mathcal{M}_{\text{main}}$ avoids revealing its logit to $\mathcal{M}_{\text{res}}$, while still using $\mathcal{M}_{\text{res}}$'s logits for its own backpropagation.

3.4.5 Training Procedure

The model training using `Delta` consists of two stages.

Stage 1. We train the backbone, $\mathcal{M}_{\text{bb}}$, and the main model, $\mathcal{M}_{\text{main}}$, with IR_{main} only. IR_{res} is ignored and not shared with the public. Therefore, there is no privacy leakage at this stage, as all data and model parameters are kept in the private environment. After stage 1, we cache all residual data IR_{res} from SVD and DCT decomposition, and apply the randomized quantization mechanism once on IR_{res}.

Stage 2. We freeze the backbone model, $\mathcal{M}_{\text{bb}}$, and continue to train the main model, $\mathcal{M}_{\text{main}}$, and the residual model, $\mathcal{M}_{\text{res}}$. As $\mathcal{M}_{\text{bb}}$ is frozen, we directly sample residual inputs for $\mathcal{M}_{\text{res}}$ from the cached residual data. While for $\mathcal{M}_{\text{bb}}$ and $\mathcal{M}_{\text{main}}$, we fetch data from the raw datasets, apply SVD and DCT decomposition, and send IR_{main} to $\mathcal{M}_{\text{main}}$.

The training procedure is provided in algorithm 2.

3.5 Privacy Analysis

This section provides the differential privacy analysis for `Delta`. As the backbone model $\mathcal{M}_{\text{bb}}$, the low-dimensional model $\mathcal{M}_{\text{main}}$ and final predictions remain private, the only public information are the residuals and the residual model $\mathcal{M}_{\text{res}}$ at stage 2 during training. Therefore, we analyze the privacy leakage of the perturbed residuals at stage 2.

Algorithm 2: `Delta` training procedure

```
Require: $\text{ep}_1$: #epochs at stage 1, $\text{ep}_2$: #epochs at stage 2
Require: $\epsilon$: privacy constraint
1:  Initialize $\mathcal{M}_{\text{bb}}$, $\mathcal{M}_{\text{main}}$, $\mathcal{M}_{\text{res}}$
2:  for $t = 1, \cdots, \text{ep}_1$ do
3:      for a batch from the dataset do
4:          Train $\mathcal{M}_{\text{bb}}$ and $\mathcal{M}_{\text{main}}$.
5:      end for
6:  end for
7:  Freeze $\mathcal{M}_{\text{bb}}$ and cache all residual data $\text{IR}_{\text{res}}$.
8:  Apply the randomized quantization based on Eq (3.6).
9:  for $t = 1, \cdots, \text{ep}_2$ do
10:     for a batch from the dataset, cached residuals do
11:         Train $\mathcal{M}_{\text{main}}$ and $\mathcal{M}_{\text{res}}$.
12:     end for
13: end for
```

Given neighboring datasets $\mathcal{D} = \left\{X^1, \cdot, X^i, \cdot, X^N\right\}$ and $\mathcal{D}^{'} = \left\{X^1, \cdot, 0, \cdot, X^N\right\}$ with i-th record removed, the global ℓ_2-sensitivity is defined as $\Delta_2 = \sup_{\mathbf{X}_{\text{res}}, \mathbf{X}^{'}_{\text{res}}} \left\|\mathbf{X}_{\text{res}} - \mathbf{X}^{'}_{\text{res}}\right\|_F \leq C$, where $\mathbf{X}_{\text{res}}, \mathbf{X}^{'}_{\text{res}}$ denote residuals obtained from $\mathcal{D}$ and $\mathcal{D}^{'}$.

We provide our DP guarantee in Theorem 7.

Theorem 7. *`Delta` ensures that the perturbed residuals and operations in the public environment satisfy (ϵ, δ)-DP given noise $\mathbf{N} \sim \mathcal{N}(0, 2C^2 \cdot \log(2/\delta')/\epsilon' \cdot \mathbf{I})$ given sampling probability p, and $\epsilon = \log(1 + p(e^{\epsilon'} - 1)), \delta = p\delta'$.*

Proof. The proof relies on Theorem 6 in [48], the subsampling theorem in [11] (Theorem 9) (replicated in Theorem 3 and 4), and the post-processing rule of DP.

First, we show that the mechanism is (ϵ, δ)-DP if $\sigma^2 = 2C^2 \cdot \log(2/\delta')/\epsilon'$. Since the mechanism of obtaining IR_{noisy} is a Gaussian mechanism, and the variance is $2C^2 \cdot \log(2/\delta')/\epsilon'$, then it follows that the mechanism ensures (ϵ, δ)-DP by Theorem 3 and Theorem 4.

The residual model $\mathcal{M}_{\text{res}}$ in the public environment and outputs after $\mathcal{M}_{\text{res}}$ are the post-processing of IR_{noisy}. Since the post-processing does not affect the DP budget [28], any operation in the public environment has the same privacy budget as the Gaussian mechanism. □

Theorem 8. *(Theorem 6 in [48]) Given noise with $\sigma^2 = 2C^2 \cdot \log(2/\delta)/\epsilon$, the Gaussian mechanism is (ϵ, δ)-DP.*

Theorem 9. *(Theorem 9 [11]) Given a randomized mechanism $\mathcal{M}'$ with privacy parameter (ϵ', δ'), and $\mathcal{M}$ with sampling probability p, for any $\epsilon' > 0, \delta' > 0$, we have $\epsilon = \log(1 + p(e^{\epsilon'} - 1))$, and $\delta = p\delta'$.*

Remark 4. *(Training with Cached Residual Improves DP). Sampling an image multiple times does not incur additional privacy leakage. As described in Sec. 3.4.5, the perturbation is only performed once before training the residual model $\mathcal{M}_{res}$. When training the residual model $\mathcal{M}_{res}$, `Delta` directly samples input for $\mathcal{M}_{res}$ from the perturbed cached residuals, with no need to perform perturbation on the fly [80].*

3.6 Empirical Evaluations

In this section, we evaluate `Delta` in terms of model accuracy, running time, and resilience against attacks.

Datasets and Models. We use CIFAR10/100 [51] and ImageNet [23]. For models, we choose ResNet-18 and ResNet-34 [39]. Hyperparameters are provided in Table 3.1 and 3.2.

Table 3.1: Hyperparameters in training ResNet-18 on CIFAR-10/100.

epochs	b	lr	wd	`orth reg`	r	t/t'
150	64	0.1	2e-4	8e-4	8	16/8

b: batch size, lr: initial learning rate, wd: weight decay.
r: #principal channels in IR_{main}, t/t': DCT/IDCT block sizes.
`orth reg`: kernel orthogonalization regularization.

Table 3.2: Hyperparameters in training ResNet-18/34 on ImageNet.

epochs	b	lr	wd	`orth reg`	r	t/t'
100	256	0.1	2e-5	0	12	14/7

Model Configuration. For ResNet models, $\mathcal{M}_{\text{bb}}$ consists of the first convolution layer, while all *ResBlocks* [39] and the fully-connected layer are offloaded in $\mathcal{M}_{\text{res}}$. On the other hand, $\mathcal{M}_{\text{main}}$'s details are provided in Table 3.3.

Table 3.3: Model parameters of $\mathcal{M}_{\text{main}}$ for ResNet-18 and ResNet-34 with 8 principal channels in IR_{main}.

ResNet-18				ResNet-34			
Resblock	n	k	q	*Resblock*	n	k	q
1,2	64	3	16	1-3	64	3	16
3,4	256	3	32	4-9	256	3	32
5,6	512	3	64	10-11	512	3	64

Asymmetric Decomposition. For the SVD, based on [70], we keep $r = 8$ principal channels in IR_{main} for CIFAR-10/100 and 12 for ImageNet to keep $> 95\%$ information in the private environment. For the DCT, we set $t, t' = 16, 8$ on CIFAR, and $14, 7$ on ImageNet to avoid noticeable information leakage in the residuals. Note that while decomposition with larger r and t' leads to less information leakage, it also incurs more computations in the private environment. Therefore, we choose the parameters to maintain a reasonable trade-off between privacy leakage and complexity.

3.6.1 Model Accuracy

In this section, we evaluate model accuracy. We train ResNet-18 on CIFAR-10/100, and ResNet-18/34 on ImageNet. We first compare the accuracy of the following schemes.

- $\mathcal{M}_{\text{main}}$: Train $\mathcal{M}_{\text{bb}}$+$\mathcal{M}_{\text{main}}$ with no residuals.
- $\mathcal{M}_{\text{main}}$+$\mathcal{M}_{\text{res}}$: Train $\mathcal{M}_{\text{bb}}$ + ($\mathcal{M}_{\text{main}}$, $\mathcal{M}_{\text{res}}$) with DP.
- Orig: Train the original model without `Delta`.

Figure 3.5 and 3.6 show the final accuracy on CIFAR-10/100 and ImageNet. First, owing to the effective asymmetric IR decomposition and the low-dimensional model, $\mathcal{M}_{\text{main}}$ already gives an accuracy that is close to the original model on both CIFAR-10/100 and ImageNet datasets. With adding $\mathcal{M}_{\text{res}}$, `Delta` achieves a comparable accuracy as the original model. With the residual information further protected by Gaussian noise, we observe that `Delta` strikes a much improved privacy-utility trade-off, with slight performance degradation under a low privacy budget (small ϵ).

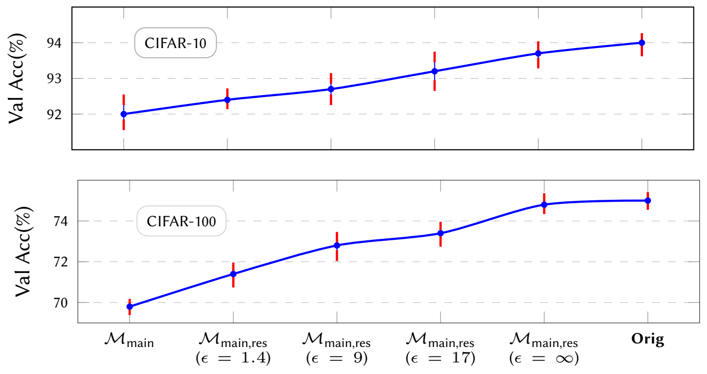


Figure 3.5: Val acc of ResNet-18 on CIFAR-10, CIFAR-100. $\mathcal{M}_{\text{main}}$ gives accuracy close to the original model. With adding $\mathcal{M}_{\text{res}}$, `Delta` achieves comparable accuracy as the original model. By adding noise to IR_{res}, `Delta` achieves strong DP while still preserving the model performance.

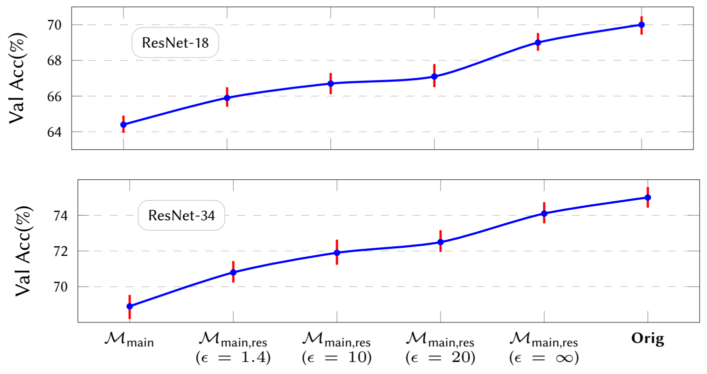


Figure 3.6: Val acc of ResNet-18 and ResNet-34 on ImageNet.

To further study the privacy-utility trade-off, we compare `Delta` with a scheme that directly adds noise to IR of the original model (naive-DP) under the same privacy guarantee. As in Equation (3.5), we perform normalization on IR before perturbation. As shown in Table 3.4, training with noise added to IR incurs significant performance degradation given the same ϵ. With the asymmetric IR decomposition, `Delta` offers a much better privacy-utility trade-off.

Table 3.4: Model accuracy of ResNet-18 by perturbing IR and IR_{res} under the same privacy budget ($\epsilon = 1.4$).

Dataset	`Delta`: perturb IR_{res}	naive-DP: perturb IR
CIFAR-10	92.4%	69.6% (↓ −22.8)
CIFAR-100	71.4%	48.3% (↓ −23.1)
ImageNet	65.9%	34.4% (↓ −31.5)

3.6.2 Running Time Analysis

In this section, we compare running time using `Delta` with a private training framework `3LegRace` [70] and private inference framework `Slalom` [89]. As a reference, we also include the time of running the original model solely in a private environment (`Priv-only`).

Theoretical Complexity. Table 3.5 lists the theoretical computational complexities by MACs (i.e., multiply-accumulate) of ResNet-18 on CIFAR-10/100, and ResNet-34 on ImageNet during the inference phase with a batch of size 1. For SVD, we use an approximation algorithm [70] that only computes the first r principal channels. First, we can see that SVD and DCT only account for a very small fraction of the total computations, which aligns with the real running time in Table 3.7. On the other hand, compared to MACs in $\mathcal{M}_{\text{res}}$, the computation complexity of $\mathcal{M}_{\text{bb}}$+$\mathcal{M}_{\text{main}}$ is much smaller, only accounting for $\sim 10\%$ of MACs in $\mathcal{M}_{\text{res}}$. This shows that, with the asymmetric decomposition that embeds most information into a low-dimensional representation, the low-dimensional model effectively reduces the computation cost of the resource-constrained private environments.

Table 3.5: MACs of ResNet-18 and ResNet-34 during forward passes with batch size 1. Compared to the residual model, the complexity of the backbone and the main model is much smaller.

Model	$\mathcal{M}_{\text{bb}}$+$\mathcal{M}_{\text{main}}$	SVD	DCT	$\mathcal{M}_{\text{res}}$
ResNet-18	48.3 M	0.52 M	0.26 M	547M
ResNet-34	437 M	1.6 M	0.7 M	3.5G

Real Running Time. We test ResNet-18 on CIFAR-100 with batch size 32 and average per-iteration time across one epoch. We use Intel SGX [21] as a private environment, and Nvidia RTX 5000 as an untrusted public environment.

Table 3.6: Running time of `Delta` and other baselines (ResNet-18/CIFAR-100, $b = 32$). `Delta` achieves significant speedup compared to `3LegRace`, and `Slalom`. Each cell denotes (time (ms), speedup)

Task	`Priv-only`	`3LegRace`	`Slalom`	`Delta`
Train	1372	237 (6×)	-	62 (22×)
Infer.	510	95 (5×)	84 (6×)	20 (25×)

Table 3.7: Time breakdown on ResNet-18/CIFAR-100. Delta significantly shrinks the time gap between the private and public environments.

	$\mathcal{M}_{\text{bb}}$	Decomp.	Parallel $\mathcal{M}_{\text{main}}/\mathcal{M}_{\text{res}}$
Forward	1 ms	3 ms	5/16 ms
Backward	2 ms	1 ms	11/39 ms

Table 3.6 shows the per-iteration training and inference time for Delta and the baselines. In both phases, owing to the effective asymmetric decomposition, the required computation in the private environment is greatly reduced. Hence, Delta achieves significant speedups compared to running a model solely in the private environment (Priv-only). Moreover, unlike the existing private inference method Slalom and training method 3Legrace, Delta obviates the need for frequent and costly inter-environment communication. Hence, Delta offers much faster training and inference. Table 3.7 further lists the running time breakdown for Delta. We observe that the time on $\mathcal{M}_{\text{bb}}$ and IR decomposition is marginal compared to $\mathcal{M}_{\text{main}}$ and $\mathcal{M}_{\text{res}}$. While the theoretical computation complexity in $\mathcal{M}_{\text{main}}$ is much lower than $\mathcal{M}_{\text{res}}$, the real running time of $\mathcal{M}_{\text{main}}$ still dominates the forward and backward passes. Nevertheless, compared to Slalom and 3LegRace, the time gap between the private and public environments is significantly shrunk.

3.6.3 Protection against Attacks

This section evaluates Delta against two privacy attacks: a model inversion attack called SecretRevealer[104], and a membership inference attack called ml-Leaks [81].

Model Inversion Attacks. SecretRevealer can leverage prior knowledge, such as blurred images, when reconstructing training samples. In our case, we allow the attack to use the quantized residuals as the prior knowledge when reconstructing images. Other model inversion attacks [85, 19, 46, 4] need confidence scores/predicted labels from the whole model, hence they do not apply to our setting.

Following the attack protocol, we first take quantized residuals as a prior knowledge and train a generative model G. Then, we optimize the latent inputs z to minimize the loss on the residual model and the discriminator used to penalize unrealistic images (L_D): $z^* = \arg\min_z L_D(G(z)) + \lambda L_{\mathcal{M}_{\text{res}}}(G(z))$, where λ controls the weights of $L_{\mathcal{M}_{\text{res}}}$.

Table 3.8: `Delta`'s performance against `SecretRevealer` on ResNet-18/CIFAR-100. Without DP, the attacker gains some prior information. With noise added, the quality of reconstruction degrades significantly.

No DP		DP with $\epsilon = 1.4$		No residuals	
SSIM	$Acc_{\mathcal{M}}$	SSIM	$Acc_{\mathcal{M}}$	SSIM	$Acc_{\mathcal{M}}$
0.18	6.75%	0.09	2.13%	0.08	1%

Table 3.8 shows `Delta`'s performance against the model inversion attack on ResNet-18/CIFAR-100. We use the structural similarity index (SSIM) to measure the similarity between the reconstructed and the original images [94]. We also measure the target model's accuracy $Acc_{\mathcal{M}}$ given the reconstructed images as inputs. High $Acc_{\mathcal{M}}$ indicates the target model regarding the reconstructed images by the attacker close to the original training samples. First, we observe that given quantized residuals without noise added, the attack can generate images that have slightly high accuracy on the target model. This indicates that the attacker can leverage the residual information during attacks. The observation is also visually reflected in reconstructed samples in Figure 3.7. Compared to the original training samples, some reconstructed images reveal the outline of objects (e.g., row 1, col 3). However, with DP, the attacker fails to use the residual information to generate images similar to the training samples. In particular, the accuracy of generated images on the target model is significantly reduced and also SSIM between the reconstructed and the original samples. As a result, the attacker behaves like the case that generates images without prior knowledge of the residuals (col 3 in Table 3.8).

Membership Inference Attacks. We choose `ML-Leaks`, a strong membership inference attack that infers membership based on confidence scores [81]. In our case, we allow the attack to use confidence scores from $\mathcal{M}_{\text{res}}$. Other membership inference attacks either required labels [20, 60, 59] or susceptible to noise [16]. As the predicted labels are secure with `Delta` and residuals are perturbed, these methods do not apply to our setting.

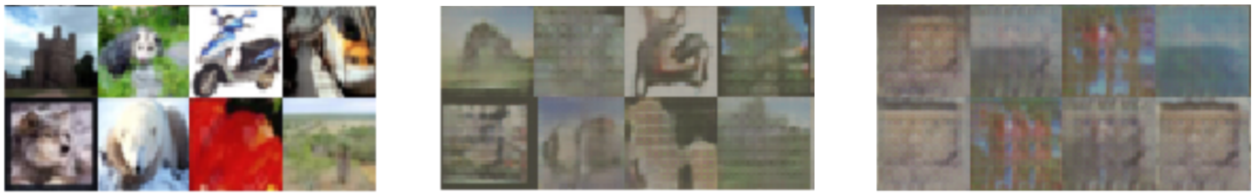

(a) Original samples (b) Recon. (no noise) (c) Recon. ($\epsilon = 1.4$)

Figure 3.7: Visualization of reconstructed images compared to the original training samples. With no DP, the model inversion attack recovers certain features ((b), row 1, col 3). However, when adding noise to the residual, the generated images are drastically affected.

Following the procedure in [81], we feed private and public samples to the shadow model, and obtain confidence score vectors from the residual model $\mathcal{M}_{\text{res}}$. We then use the vectors from public and private datasets to train an attacker model, which learns to classify whether the confidence score vector comes from public or private datasets.

Table 3.9: Membership inference attack on Delta (ResNet-18, CIFAR-100). The DP mechanism is essential to provide further protection for the residual.

	Attack w. $5k$ samples		Attack w. $10k$ samples	
	No DP	$\epsilon = 1.4$	$\epsilon = \infty$	$\epsilon = 1.4$
Acc	0.56	0.52	0.60	0.55
F1	0.68	0.57	0.73	0.68

Table 3.9 lists the attack performance on CIFAR-100 with ResNet-18. To align with `ML-Leaks`'s protocol, we use $5k$ and $10k$ samples from the training dataset as a public dataset. The rest of the samples are used as a private dataset to train the target model (using `Delta`). We train the attack model for 50 epochs, with an initial learning rate of 0.1.

We observe that with perturbed residuals, attacks through $\mathcal{M}_{\text{res}}$'s outputs result in a poor performance compared to training without noise added. It indicates the DP mechanism provides further protection for the residuals, and prevents attackers from inferring membership. Furthermore, as the number of public samples reduces, the attack performance degrades further, implying the attack also heavily depends on prior knowledge via accessing a subset of the target dataset. Such an observation reveals one critical limitation of the attacks. That is, they need to get access to a subset of the target data to obtain a good estimate of the target data distributions. Otherwise, the attack performance collapses. However, in real scenarios, private data can be completely out of attackers' reach, rendering the target distribution's estimation impossible. As a result, the attacks can easily fail.

3.7 Conclusion

We proposed a generic private training and inference framework, `Delta`, with strong privacy protection, high model accuracy, and low complexity. `Delta` decomposes the intermediate representations into asymmetric flows: information-sensitive and residual flows. We design a new low-dimensional model to learn the information-sensitive part in a private environment, while outsourcing the residual part to a large model in a public environment. A DP mechanism and binary quantization scheme further protect residuals and improve inter-environment communication efficiency. Our evaluations show that `Delta` achieves strong privacy protection while maintaining model accuracy and computing performance. While we evaluate `Delta` in a TEE-GPU environment, `Delta` can be generalized to other setups such as federated settings, with resource-constrained client side as a private environment, and the server as a public environment.

Chapter 4

Efficient Private Learning in Federated Settings

Federated Learning (FL) is emerging as a popular, promising decentralized learning framework that enables collaborative training among clients, with no need to share private data between them or to a centralized server. However, considering many edge clients do not have sufficient computing, memory, or communication capabilities, federated learning of large models still faces significant bottlenecks. To keep such *weak but crucial* clients in the loop, prior works either consider a heterogeneous-client setting where clients train models with different sizes; or offload training to the server. However, the heterogeneous-client setting requires some clients to train full model, which is not aligned with the resource-constrained setting; while the latter ones break privacy promises in FL when sharing intermediate representations or labels with the server. To overcome these limitations, in this work, we formulate a realistic, but much less explored, cross-device FL setting in which no client can train a full large model nor is willing to share any intermediate information with the remote server. Under such a formulation, we develop a principal sub-model (`PriSM`) training methodology to collaboratively train a full large model, while assigning each client a small sub-model that is a *probabilistic* low-rank approximation to the full server model. When creating sub-models, `PriSM` first performs a principal kernel analysis in the orthogonal kernel space to obtain *importance* of each kernel. Then, `PriSM` adopts a novel importance-aware sampling process to select a subset of kernels (i.e., a kernel with high importance is assigned with a higher sampling probability). This sampling process ensures each sub-model is still a *low-rank approximation* to the full model, while all sub-models together achieve *nearly full coverage* on the principal kernels. To further improve memory efficiency while still preserving accuracy, `PriSM` also exploits low-rank structure in intermediate representations and allows each sub-model to learn only a subset of them. Our evaluations on various datasets and models (CNNs, LSTMs, Transformers) under different resource-constrained settings demonstrate that `PriSM` yields an accuracy improvement of up to 10% compared to existing works. More importantly, `PriSM` does not incur significant accuracy degradation compared to full-model training (e.g., only $\sim 2\%$ accuracy drops for ResNet-18/CIFAR-10 when clients train only $0.2\times$ sub-models).

4.1 Introduction

Federated Learning (FL) is emerging as a popular paradigm for decentralized and privacy-preserving machine learning as it allows clients to perform ML optimization jointly without directly sharing local data [63, 47]. Thus, it enables privacy protection on local data, and leverages distributed local training to attain a better global model. This creates opportunities for many edge devices rich in data to participate in joint training without data sharing. For example, resource-limited smart home devices can train local vision or language models using private data, and achieve a server model that generalizes well to all users via FL [75].

Despite significant progress in FL in the recent past, several crucial challenges still remain when moving to the edge. In particular, limited computation, memory, and communication capacities prevent clients from learning large models for leveraging vast amounts of local data at the clients. This problem is getting increasing attention in current literature [26, 41, 100, 40, 64, 92, 37]. For example, recent works propose a sub-model training methodology by assigning clients with different subsets of the full model depending on their available resources [26, 41, 100, 57]. However, these works have an underlying assumption that some of the clients have sufficient resources to train a nearly full large model. In particular, methods like FedHM [100] that adapt low-rank compression to FL incur more memory footprint for intermediate representations, even for small sub-models compared to the full model training. As a result, server model size is limited by the clients with maximum computation, memory, and communication capacities. To overcome resource constraints on clients, other works [92, 87, 37] change the training paradigm by splitting a model onto server and clients. The computational burden on the clients is therefore relieved as the dominant part of the burden is offloaded to the server. However, such a methodology requires sharing of intermediate representations and/or labels with the server, which directly leaks input information and potentially compromises privacy promises of FL [29, 104].

Our Contributions. Unlike prior works, this work considers even more constrained and realistic settings at the edge, in which no client is capable of training a large model nor is willing to share any intermediate data and/or labels with the server. To this end, we propose a novel Principal Sub-Model (`PriSM`) training methodology. At a high level, `PriSM` *allows each client to only train a small sub-model, while still attaining a full model on the server and achieving comparable accuracy as the standard training*.

The cornerstone of `PriSM` is the models' inherent low-rank structure, which is commonly used in reducing compute costs [49, 24]. However, naive low-rank approximation in FL [100], where all clients only train top-k kernels, incurs a notable accuracy drop, especially in very constrained settings. In Figure 4.1, we delve into the matter by showing the number of principal kernels required in the orthogonal space to accurately approximate each convolution layer in the first two *ResBlocks* in ResNet-18 [39] during FL training*. We observe that even at the end of the FL training, around half of the principal kernels are still needed to sufficiently approximate each convolution layer. We have similar findings for the remaining convolution layers (See Sec 4.3.4). Therefore, to avoid the reduction in server model capacity, it is essential to ensure that all server-side principal kernels are collaboratively trained on clients, especially when each client can only train a very small sub-model (e.g., $< 50\%$ of the server model).

*See Sec 4.3.4 for further details, especially for calculating the required number of principal kernels.

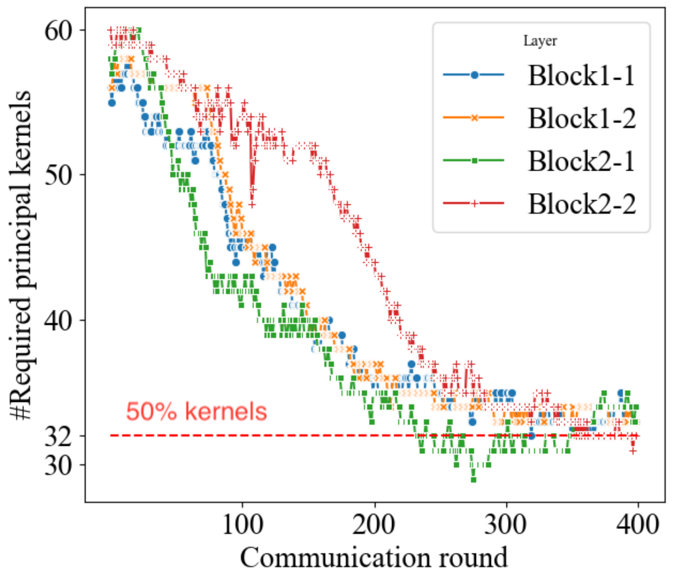


Figure 4.1: #principal kernels in the orthogonal space required to accurately approximate convolution layers in the first two ResBlocks in ResNet-18 during FL training. Block$i$-j indicates j-th convolution layer in i-th ResBlock. The server model gradually attains a low-rank structure. However, even in the final model, half of the principal kernels are still required to approximate most layers.

Based on the above observations, we develop an effective probabilistic strategy to select a subset of kernels and create a sub-model for each client as shown in Figure 4.2. More specifically, `PriSM` first converts the model into orthogonal space where original convolution kernels are decomposed into principal kernels using singular value decomposition (SVD). To approximate the original server model, `PriSM` utilizes a novel sampling process, where a principal kernel with a larger singular value has a higher sampling probability. The probabilistic process ensures that all sub-models can together provide nearly full coverage of the principal kernels, thus reaching the near full-model training performance with significantly reduced costs on local computation and communication during sub-model aggregation. `PriSM` further improves memory efficiency by exploiting low-rank structure in intermediate activations and allows each client to learn only a subset of these representations while still preserving training performance. Thus, computation, memory, and communication bottlenecks at the edge are effectively resolved.

In the following, we summarize our contributions as follows:

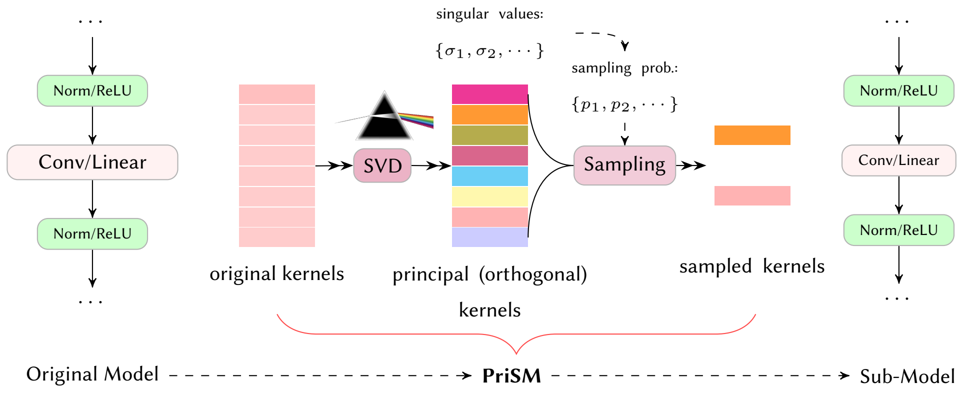


Figure 4.2: Creating clients' sub-models. `PriSM` samples a subset of principal kernels to create a client's sub-model with less computation and communication overhead. The importance-aware sampling scheme (based on singular values) ensures every sub-model approximates the full large model, and all sub-models together provide nearly full coverage of the principal kernels. `PriSM` further improves memory efficiency by allowing each sub-model to learn only a subset of intermediate representations.

- We propose a novel algorithm `PriSM` to overcome resource constraints of large model training in federated settings. `PriSM` uses a novel probabilistic sampling scheme to select a subset of the kernel and create a sub-model for each resource-constrained client.

- We present comprehensive empirical evidence showing that kernel sampling is crucial to preserve models' performance, rather than simply fixed kernel dropout.

- We conduct extensive evaluations of `PriSM` on vision and language tasks under resourced-constrained settings where no client is capable of training the large full model. In particular, we consider both resource constraints and heterogeneity in system capacities as well as data distribution.

- We further provide detailed insights into the performance gains attained by `PriSM` via 1) analyzing server model's rank structure during training; 2) investigating different sampling methods; 3) profiling the kernel sampling process; 4) breaking down costs in the system.

Our results demonstrate that `PriSM` delivers consistently better performance compared to prior works, especially when participating clients have very limited capacities. For instance, on ResNet-18/CIFAR-10 (see Figure 4.5), we show that `PriSM` only incurs around marginal accuracy drop for i.i.d and non-i.i.d datasets under a constrained setting where clients train sub-models with only 20% of the principal kernels, accounting for $\sim 5\%$ of the full server model. Compared to other solutions, `PriSM` improves the accuracy by up to 10%.

4.2 Methodology

In this section, we first motivate our proposal, Principal random Sub-Model training (`PriSM`), with an observation of orthogonality in neural network layers. Then, we describe the details of `PriSM`.

Notations– $\|\cdot\|_F$: Frobenius norm. σ_i: i-th singular value in a matrix. $\circledast$: convolution. $\cdot$: matrix multiplication. $\langle\cdot,\cdot\rangle$: sum of element-wise multiplication or inner product. $tr(A)$: trace of a matrix.

4.2.1 Motivation: An Observation on Orthogonality

Without loss of generality, we consider a convolution layer with kernels $W \in \mathbb{R}^{N\times M\times k\times k}$ and input $X \in \mathbb{R}^{M\times h\times w}$, where N and M denote the number of output channels and input channels, k is kernel size, and $h\times w$ is the size of the input image along each channel. Based on a common technique *im2col* [90], the convolution layer can be converted to matrix multiplication as $\overline{Y} = \overline{W}\cdot\overline{X}$, where $\overline{W} \in \mathbb{R}^{N\times Mk^2}$ and $\overline{X} \in \mathbb{R}^{Mk^2\times hw}$. For kernel decorrelation, we apply singular value decomposition (SVD) to map kernels into orthogonal space as: $\overline{W} = \sum_{i=1}^{N}\sigma_i\cdot\boldsymbol{u}_i\cdot\boldsymbol{v}_i^T$, where $\{\boldsymbol{u}_i\}_{i=1}^N$, $\{\boldsymbol{v}_i\}_{i=1}^N$ are two sets of orthogonal vectors[†]. The convolution can be decomposed as

$$\overline{Y} = \sum_{i=1}^{N}\overline{Y}_i = \sum_{i=1}^{N}\sigma_i\cdot\boldsymbol{u}_i\cdot\boldsymbol{v}_i^T\cdot\overline{X}. \tag{4.1}$$

[†]We assume w.l.o.g $\overline{W}$ is a tall matrix.

For $\forall i \neq j$, it is easy to verify that $\langle \overline{Y}_i, \overline{Y}_j \rangle = \sigma_i \cdot \sigma_j \cdot tr(\overline{X}^T \cdot \boldsymbol{v}_i \cdot \boldsymbol{u}_i^T \cdot \boldsymbol{u}_j \cdot \boldsymbol{v}_j^T \cdot \overline{X}) = 0$, namely the output features $\overline{Y}_i$ and $\overline{Y}_j$ are orthogonal. Therefore, if we regard $\overline{W}_i = \sigma_i \cdot \boldsymbol{u}_i \cdot \boldsymbol{v}_i^T$ as a principal kernel, different principal kernels create orthogonal output features. To illustrate this, Figure 4.3 shows an input image (left) and the outputs (right three) generated by principal kernels. We can observe that principal kernels captures different features and serve different purposes.

As revealed in [97, 10, 93], imposing orthogonality on kernels leads to better training performance. This motivates us to initiate the training with a set of orthogonal kernels. Furthermore, to preserve kernel orthogonality during training, it is critical to constantly refresh the orthogonal space through re-decomposition. The above intuitions based on the observation on orthogonality play a key role in `PriSM`, which is described in the following section.

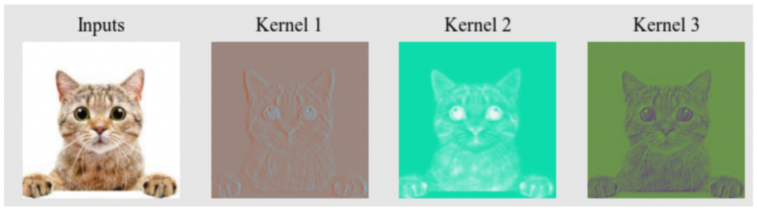


Figure 4.3: Orthogonal features generated by principal kernels. Different principal kernels capture different features (Kernel 1: outline of the cat, Kernel 2 and 3: textures but on distinct regions).

Remark 5. *Besides the convolution layer discussed above, the layer decomposition using SVD also applies to general linear layers (e.g., MLPs, LSTMs, Transformers), where weights $W \in \mathbb{R}^{N \times M}$. In particular, we can directly apply SVD and obtain the principal components.*

4.2.2 `PriSM`: Principal Probabilistic Sub-Model Training

Motivated by the observation that output features are orthogonal given orthogonal kernels, we propose `PriSM`, a new sub-model training method that directly trains orthogonal kernels in clients. Particularly, `PriSM` introduces two key components to ensure training performance with sub-models under very constrained settings. First, considering different orthogonal kernels and their contributions to outputs are weighed by the corresponding singular values, PriSM devises a novel importance-aware sampling scheme to create client sub-models to achieve computation and communication efficiency. In particular, the sampling scheme brings two benefits: i) each sub-model is a low-rank approximation of the server model; ii) the conglomerate of the sampled sub-models enables nearly full coverage of orthogonal kernels. Second, to further improve the memory efficiency while maintaining training performance, PriSM exploits low-rank structure in activations and allows each client to learn only a subset of intermediate representations in each layer. We start with unfolding sub-model creation, and then describe the training procedure in PriSM.

4.2.2.1 Sub-Model Creation

This section explains how PriSM creates different sub-models for local clients.

Computation and communication efficiency via sub-model sampling. For a convolution layer with principal kernels $\{\overline{W}_i\}_{i=1}^N$, and the corresponding singular values $\{\sigma_i\}_{i=1}^N$, we randomly sample r principal kernels denoted by $\overline{W}^c$ with sampling probability for i-th kernel as follows:

$$p_i = \frac{\sigma_i^\kappa}{\sum_{j=1}^N \sigma_j^\kappa}. \tag{4.2}$$

Here, for a given layer, r is decided by c-th client's resource budget, and κ is a smooth factor controlling the probability distribution in sampling. Indices of selected kernels are denoted as $\mathcal{I}_c$.

The convolution output is calculated as

$$\overline{Y} = \sum_{i \in \mathcal{I}_c} \overline{Y}_i = \sum_{i \in \mathcal{I}_c} \sigma_i \cdot \boldsymbol{u}_i \cdot \boldsymbol{v}_i^T \cdot \overline{X}. \tag{4.3}$$

By performing the sampling process for every convolution layer, we create a *random* low-rank model for each client. *Important* kernels with large singular values are more likely to be chosen, and all sub-models can together provide nearly full coverage of principal kernels. The resulting sub-model hence has convolution layers with fewer kernels, reducing required computations and communication overhead (when aggregating models). Other element-wise layers, such as ReLU and batch normalization, remain the same.

Memory efficiency via learning a subset of features. We further exploit low-rank structure in intermediate representations. For a sub-model above, each convolution layer is decomposed into two sublayers: $Conv_{\mathrm{U}}$ and $Conv_{\mathrm{V}}$ with kernels $\{\boldsymbol{u}_i\}_{i\in\mathcal{I}_c}$ and $\{\boldsymbol{v}_i\}_{i\in\mathcal{I}_c}$ respectively. Without further optimization, output from $Conv_{\mathrm{U}}$ consumes the same memory as the original layer, causing high memory pressure at the edge. However, inputs to $Conv_{\mathrm{U}}$, $\overline{X} \in \mathbb{R}^{r\times hw}$, and kernels $\overline{W} \in \mathbb{R}^{N\times r}$ are both low-rank. Therefore, based on the rank inequality of matrix multiplication [12], we obtain

$$\mathrm{Rank}\overline{Y} \leq \min\left(\mathrm{Rank}\overline{X}, \mathrm{Rank}\overline{W}\right) \leq r. \tag{4.4}$$

Therefore, the output of $Conv_{\mathrm{U}}$ is also low-rank. Such a low-rank structure reflects correlation among output channels, indicating channel redundancy in outputs when $r < N$. Based on the observation, we further allow $Conv_{\mathrm{U}}$ to compute only a subset of output channels, therefore reducing memory footprints of activations while still preserving necessary information. Typically, given input to $Conv_{\mathrm{U}}$ with r input channels, `PriSM` only computes r output channels. During implementation, we replace the select $\boldsymbol{u}_i$ with its subset $\boldsymbol{u}_i(1:r)$ when computing output features from $Conv_{\mathrm{U}}$. For simplicity, we will still use $\boldsymbol{u}_i$ in the rest of the paper to denote the subset $\boldsymbol{u}_i(1:r)$. The unselected channels are cached in $\hat{\boldsymbol{u}}_i$, and are used in model aggregation.

4.2.2.2 Training

This section details the training procedure in `PriSM`. We describe each component below.

Local training. On each client, during local training, the sub-model with parameter $\{\sigma_i, \boldsymbol{u}_i, \boldsymbol{v}_i\}_{i\in\mathcal{I}_c}$ are updated. The sub-model also consists of trainable layers such as BatchNorm but with fewer channels denoted in $\mathcal{I}_c$. `PriSM` allows σ_i to be trained in local training, thus ensuring changes regarding each principal kernel's importance are captured in local clients. In addition, σ_i will be merged into $\boldsymbol{u}_i, \boldsymbol{v}_i$ as $\boldsymbol{u}_i^{'} = \sqrt{\sigma_i}\boldsymbol{u}_i, \boldsymbol{v}_i^{'} = \sqrt{\sigma_i}\boldsymbol{v}_i$ to reduce memory footprints.

Sub-model aggregation. On the server side, with sub-models obtained from clients, we aggregate i-th principal kernel as follows:

$$\overline{W}_i = \left(\left[\sum_{c\in\mathcal{C}} \alpha_i^c \boldsymbol{u}_i^{'c}; \quad \hat{\boldsymbol{u}}_i\right]\right) \cdot \left(\sum_{c\in\mathcal{C}} \alpha_i^c \boldsymbol{v}_i^{'c}\right)^T, \tag{4.5}$$

where $\mathcal{C}$ denotes the subset of active clients, α_i is the aggregation coefficient for i-th kernel. We propose a weighted averaging scheme: if i-th kernel is selected and trained by C_i clients, then $\alpha_i^c = 1/C_i$. Furthermore, the unselected output channels $\hat{\boldsymbol{u}}_i$ are concatenated with the aggregated $\boldsymbol{u}_i$ before reconstructing the orthogonal kernel to preserve model capacity. If $\overline{W}_i$ was not selected by any client, it remained unchanged in the server. The full model in the original space is constructed by converting each 2-dimensional $\overline{W}_i$ to the original dimension $\mathbb{R}^{M\times k\times k}$ and combining them.

Orthogonal space refresh. After model aggregation, we perform SVD on the updated kernels $\overline{W}$ to preserve orthogonality among the principal kernels. Thus, in the next communication round, the importance-aware sampling can still create low-rank sub-models for different clients..

We further use two additional techniques to improve learning efficiency in the orthogonal space: activation normalization, and regularization on orthogonal kernels.

Activation normalization. We apply batch normalization without tracking running statistics; namely, the normalization always uses current batch statistics in the training and evaluation phases. Each client applies normalization separately with no sharing of statistics during model aggregation. Such an adaptation is effective in ensuring consistent outputs between different sub-models and avoids potential privacy leakage through the running statistics [5].

Regularization. When learning a factorized model on a client, applying weight decay to $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ separately results in poor final accuracy. Inspired by [50], for training on client c, we add regularization to the subset of kernels as follows:

$$reg = \frac{\lambda}{2} \left\| \sum_{i \in \mathcal{I}_c} \boldsymbol{u}_i^{'} \cdot \boldsymbol{v}_i^{'T} \right\|_F^2 , \tag{4.6}$$

where λ is the regularization factor, $\mathcal{I}_c$ denotes the subset of principal kernels on client c.

Algorithm 3 presents an overall description of `PriSM`. We only show the procedure on a single convolution layer with kernels W for the sake of simplifying notations.

In the following remarks, we differentiate `PriSM` from Dropout and Low-Rank compression.

Remark 6. ***`PriSM` vs Dropout.*** *`PriSM` shares some computation similarity with model training using regular dropout [26, 41]. However, methods with regular dropout simply select a fixed subset of kernels with size depending on a client's resource constraints, and suffer severe performance degradation, esp. with a high dropout probability. In contrast, `PriSM` performs importance-aware sampling in the orthogonal space. Each sub-model approximates the full model, and different sub-models do not create significant inconsistency. Importantly, PriSM obtains a final global model with full capacity, even though only allows clients to train sub-models.*

Remark 7. ***`PriSM` vs Low-Rank Compression.*** *`PriSM` is not a low-rank compression method. Low-rank compression methods such as FedHM [100] aim to construct a smaller server model by completely discarding some kernels even though they can still contribute to training performance (See Figure 4.1 in Section 4.1 and*

Figure 4.8 in Section 4.3.4). `PriSM` randomly select sub-models so that every kernel is possible to be learned. Furthermore, `PriSM` achieves memory efficiency by exploiting low-rank properties in intermediate representations, which is not seen in other low-rank methods.

Algorithm 3: `Delta`: Principal Probabilistic Sub-Model Training

```
Input: layer parameters $W$, client capacities.
1:  Decompose $W$ into orthogonal kernel using SVD $\rightarrow \{\overline{W}_i\}_{i=1}^{N}$.
2:  for communication round $t = 1, \cdots, T$ do
3:      Choose a subset of clients $\rightarrow \mathcal{C}$.
4:      for each client $c \in \mathcal{C}$ do
5:          Compute the sub-model size for client $c \rightarrow |\mathcal{I}_c|$.
6:          Sub-model: Obtain a sub-model following the procedure in Sec 4.2.2.1 $\rightarrow \mathcal{I}_c, \overline{W}^c$.
7:          Local training: Perform LocalTrain $\leftrightarrow \mathcal{I}_c, \overline{W}^c$.
8:      end for
9:      Sub-model aggregation: Aggregate parameters based on Eq. (4.5) $\overline{W} \leftarrow \{\overline{W}^c\}_{c\in\mathcal{C}}$.
10:     Orthogonal space refresh: Perform SVD on $\overline{W}$.
11: end for

    |———————————————————————————————————
12: LocalTrain $\leftrightarrow \mathcal{I}_c, \overline{W}^c$
13: for local iteration $k = 1, \cdots, K$ do
14:     Sample an input batch from the local dataset $\rightarrow \mathcal{D}_k$.
15:     Perform the forward and backward pass $\leftarrow \mathcal{D}_k, \overline{W}^c$.
16:     Update the local sub-model using SGD $\rightarrow \overline{W}^c$.
17: end for
    |———————————————————————————————————
```

4.3 Empirical Evaluations

In this section, we present numerical analyses of `PriSM`. First, we study the necessity of probabilistic kernel sampling in preserving models' capacity and accuracy via centralized training. Then, we evaluate `PriSM` under resource-constrained settings where no clients can train the large full model. Furthermore, we consider both homogeneous and heterogeneous client settings. Specifically, in homogeneous settings, all clients have the same limited computation, memory, and communication capacity, while in heterogeneous settings, clients' capacities might vary. We also compare `PriSM` with three other baselines: ordered dropout in orthogonal space (OrthDrop), ordered dropout in original space (OrigDrop), and SplitMix [40]. At a high level, our results demonstrate that `PriSM` achieves comparable accuracy to full-model training even when only training very small sub-models on all clients. Additionally, we provide more insights into the superior performance of `PriSM` by 1) analyzing the server model's rank structure; 2) profiling the kernel sampling process during training; 3) investigating cost breakdowns in a federated system. Finally, we conduct ablation studies including analyzing several potential sampling strategies and impacts of training hyperparameters.

Baselines. Prior methods such as FjORD [41] and HeteroFL [26] select sub-models from the original kernel space, for which we denote as OrigDrop. In implementation, we follow the procedure in HeteroFL. On the other hand, we use OrthDrop to denote selecting fixed top-k kernels from the orthogonal space such as in FedHM [100]. For SplitMix, we follow the procedure in [40] to create atom sub-models and evaluate the ensemble of all sub-models on the server side.

Models and Datasets. We train ResNet-18 on CIFAR-10 [51], CNN on FEMNIST [15] and LSTM model on IMDB [61]. We also train a transformer model DeiT[88] on CIFAR-10. Further details are listed in Table 4.1, 4.2, 4.3.

Data Distribution. For CIFAR-10 and IMDB, we uniformly sample an equal number of images for each client when creating i.i.d datasets. For non-i.i.d datasets, we first use Dirichlet function Dir(α) [77] to create sampling probability for each client and then sample an equal number of training images for clients. We create two different non-i.i.d datasets with $\alpha = 1$ and $\alpha = 0.1$, where a smaller α indicates a higher degree of non-i.i.d. For FEMNIST, we directly use the dataset without any additional preprocessing.

FL Setting. We simulate an FL setting with 100 clients with 20 random clients active in each communication round. Each client trains its model for 2 local epochs in each round. We use SGD with momentum during training. The learning rate is initially 0.1 for ResNet-18 and DeiT and 0.01 for CNN/FEMNIST, and decayed by a cosine annealing schedule.

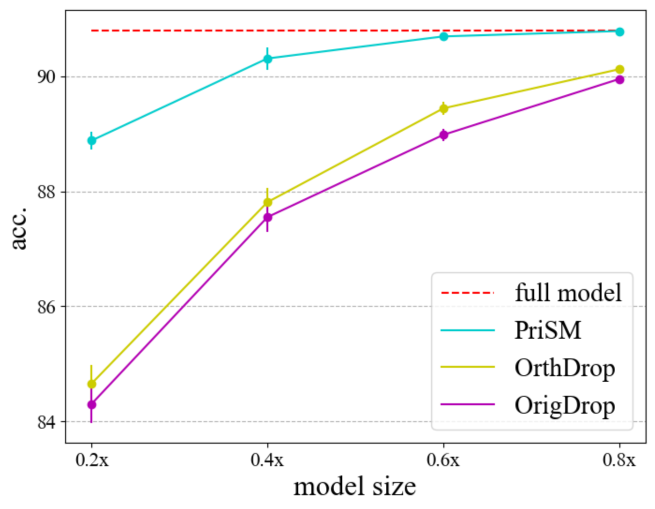


Figure 4.4: Accuracy of VGG11 on CIFAR-10 with different sub-models. `PriSM` delivers much better performance under very constrained settings (e.g., training with only $0.2\times$ sub-models) compared to OrthDrop and OrigDrop.

Table 4.1: DeiT/CIFAR-10

Module	kernel size	#kernel	#layers	#heads
Embedding	4	192	-	-
Attention	-	-	12	3

Table 4.2: CNN/FEMNIST

Module	#kernels	size	stride	ReLU
Conv1	64	5	1	✓
Pooling1	-	2	2	✗
Conv12	64	3	1	✓
Pooling2	-	2	2	✗
Classification	10	-	-	✗

Table 4.3: LSTM/IMDB

Module	input size	output size	hidden size	#layers
Embedding	1001	64	-	-
LSTM	64	256	256	2
FC	256	1	-	-

4.3.1 Probabilistic Sub-Model Training in Centralized Settings

We first demonstrate that probabilistic kernel sampling is essential to preserve models' capacity and maintain accuracy in centralized settings. While centralized training is not our goal, it provides direct evidence that `PriSM` reduces training complexities without significantly sacrificing accuracy.

We use VGG11 as an example and train the model on CIFAR-10. During training, in each iteration, we sample a subset of kernels and create a sub-model as described in `PriSM`. In different iterations, different subsets of kernels will be sampled and trained. We adopt the following hyperparameters during training(SGD, *batch size*=128, *weight decay*=1e-4, *epochs*=150). To further demonstrate the benefits of kernel sampling, we also compared the accuracy with OrigDrop and OrthDrop, where we respectively extract a fixed subset of kernels and create sub-models in the original and orthogonal kernel space.

Figure 4.4 shows the accuracy of VGG11 on CIFAR-10 with different sub-models. We observe that `PriSM` achieves much higher accuracy compared to the other two baselines, especially under very constrained settings (e.g., $0.2\times$ sub-models). Hence, compared to these compression methods that reduce models' capacity, `PriSM` preserves models' capacity via the probabilistic kernel sampling, therefore maintaining model accuracy.

4.3.2 Performance on Constrained Homogeneous Clients

In this setting, we assume that all clients have the same limited capacity. We vary the client sub-model size from 0.2 to 0.8 of the full server model, where $0.x$ indicates that only a $0.x$ subset of the principal kernels are sampled in each convolution layer from the server model (denoted as *keep ratio*). Accordingly, for a given layer, $r = 0.x \times N$, where N is the number of output channels in the layer. In Table 4.4, we list the computation, memory and communication footprints for sub-models of ResNet-18. DeiT/CIFAR-10, CNN/FEMNIST, and LSTM/IMDB also enjoy similar cost reductions. It is worth noting that `PriSM` incurs

Table 4.4: Model size, MACs, activation memory for sub-models in `PriSM` (batch size: 32). Unlike conventional low-rank methods, `PriSM` greatly reduces model size, computations, and activation memory when training with very small sub-models.

Model		Full	0.8×	0.6×	0.4×	0.2×
		`PriSM`				
ResNet-18	Params	11 M	7.9 M (72%)	4.5 M (41%)	2 M (18%)	.5 M (4.5%)
	MACs	35 G	25.5 G (73%)	14.7 G (42%)	6.87 G (20%)	2 G (5.6%)
	Mem	31.5 M	37 M (115%)	28.3 M (90%)	18.9 M (60%)	9.5 M (30%)
		OrthDrop (FedHM [100])				
	Params	11 M	9.9 M (90%)	7.4 M (67%)	4.9 M (44%)	2.5 M (22%)
	MACs	35 G	28.8 G (82%)	22.4 G (64%)	16 G (46%)	7.4 G (23%)
	Mem	31.5 M	44.9 M (143%)	40.5 M (129%)	36.1 M (115%)	31.7 M (101%)

much smaller costs compared to the deterministic compression methods(FedHM), thanks to the memory-efficiency design.

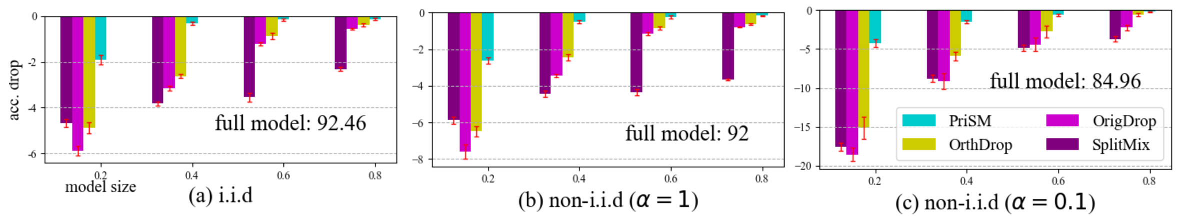


Figure 4.5: Accuracy drops on ResNet18/CIFAR-10 on homogeneous clients compared to full-model training. PriSM constantly delivers better performance compared to OrigDrop, OrthDrop, and SplitMix, significantly outperforming them under very constrained realistic edge settings. (bar: mean; line: std)

ResNet-18/CIFAR-10. Figure 4.5 shows final validation accuracy drops of ResNet-18 with different sub-models on i.i.d and non-i.i.d datasets compared to full-model training. We note that `PriSM` constantly delivers better performance than the other three baselines. The performance gap is more striking under very constrained settings. For instance, when only $0.2\times$ sub-models are supported on clients, `PriSM` attains comparable accuracy as full-model training, and achieves up to $10\%/14\%/13\%$ performance improvement compared to OrthDrop, OrigDrop and SplitMix on non-i.i.d dataset with $\alpha = 0.1$. We also make two key observations. First, training with sub-models in the orthogonal space provides better performance than in the original space, which aligns with our intuition in Section 4.2.1. Second, our importance-aware sampling strategy for creating sub-models is indispensable as demonstrated by the notable performance gap between `PriSM`, OrthDrop and SplitMix.

CNN/FEMNIST. We adopt a CNN model as in [41], which consists of two convolution layers (See Table 4.3 in Appendix for details). During training, we simulate 100 clients, in which each client contains 10 users' data from FEMNIST [15]. Figure 4.6 shows accuracy drops of sub-model training with different sizes. Similar as in the ResNet experiments, `PriSM` incurs very small accuracy drops even under very constrained settings. In particular, $0.2\times$ sub-model training only results in less than 1% accuracy drops, greatly outperforming the other three baselines.

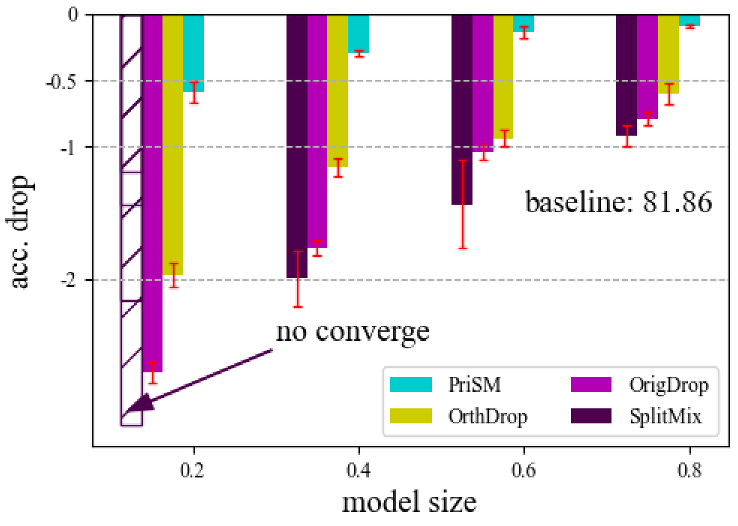


Figure 4.6: Accuracy drops on FEMNIST on homogeneous clients compared to full-model training. `PriSM` incurs smaller accuracy drops under very constrained settings than the other three baselines.

LSTM/IMDB. The LSTM model used in FL training is detailed in Table 4.3. During training, we simulate 100 clients, in which each client is assigned 375 training samples. We create local datasets with two distributions using the same method as in CIFAR-10: i.i.d and non-i.i.d ($\alpha = 0.1$). Figure 4.7 shows accuracy drops of sub-model training on IMDB in homogeneous client settings. While the task on IMDB is just a binary classification problem, `PriSM` still achieves the best final server model accuracy on both i.i.d and non-i.i.d datasets. On i.i.d datasets, training using sub-models achieves comparable accuracy as full-model training, even only using very small sub-models such as $0.2\times$. On the other hand, on non-i.i.d dataset, OrigDrop suffers notable accuracy drops compared to `PriSM`.

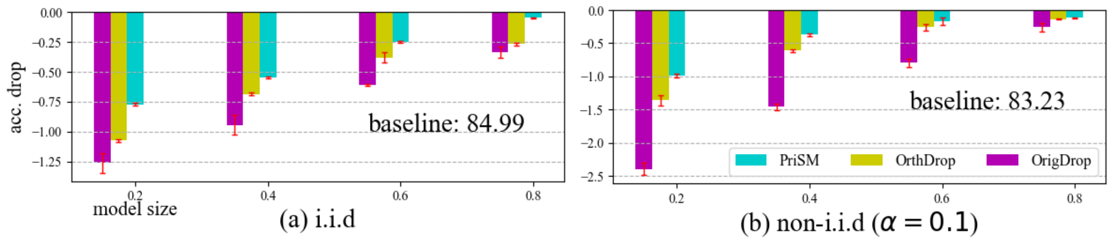


Figure 4.7: Accuracy drops on IMDB on homogeneous clients compared to full-model training.

Transformer/CIFAR-10. `PriSM` also applies to the transformer architecture. To that end, we test `PriSM` on DeiT [88], an efficient image transformer model. It consists of 12 attention layer, and each layer has 3 attention heads. Detailed architecture is provided in Table 4.1. We use a pre-trained model on ImageNet and adopt the same hyperparameter settings as in ResNet-18/CIFAR-10. Table 4.5 shows the final global model performance with different sub-model training. Similar to ResNet-18/CIFAR-10, `PriSM` maintains comparable accuracy on both i.i.d and non-i.i.d datasets even under very constrained settings such as when clients can only train $0.2\times$ or $0.4\times$ sub-models. Therefore, thanks to the probabilistic sampling process and memory-efficiency design, `PriSM` preserves the server model's capacity while significantly reducing training costs.

Table 4.5: Training performance of DeiT/CIFAR-10 resource-constrained clients.

Distribution	Full Model (Baseline)	$0.8\times$	$0.6\times$	$0.4\times$	$0.2\times$
i.i.d	92.04	91.92 ± 0.18	91.36 ± 0.26	90.12 ± 0.31	88.34 ± 0.28
non-i.i.d ($\alpha = 1$)	91.14	90.84 ± 0.15	90.50 ± 0.18	89.72 ± 0.21	87.87 ± 0.32

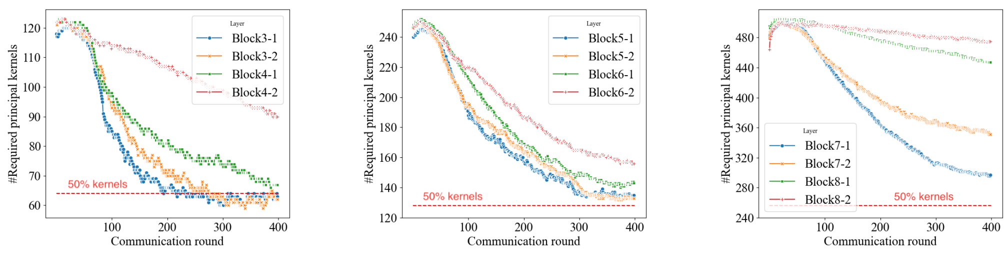
4.3.3 Performance on Constrained Heterogeneous Clients

To simulate clients with varying limited capacity, we simulate the following setting: 40% clients train $0.4\times$ sub-models, and 60% clients train $0.2\times$ sub-models. No participating client trains the full model. For baseline methods, we follow the same strategy as in Section 4.3.2.

Table 4.6 lists the final accuracy achieved by different methods under the heterogeneous setting. `PriSM` greatly outperforms the baseline methods even when $0.4\times$ sub-models are supported on a small fraction of clients. Furthermore, similar to the results in Section 4.3.2, the benefits of training in the orthogonal space and importance-aware sampling strategy are also observed in heterogeneous client settings.

Table 4.6: Training performance of ResNet-18/CIFAR-10 on heterogeneous clients.

Distribution	Full Model (Baseline)	OrigDrop	OrthDrop	`PriSM`
i.i.d	92.46	88.86 ± 0.17	89.57 ± 0.16	90.63 ± 0.14
non-i.i.d (1)	92	86.91 ± 0.24	88.78 ± 0.22	89.89 ± 0.2
non-i.i.d (0.1)	84.96	76.38 ± 0.92	78.37 ± 0.99	82.58 ± 0.51



(a) ResBlock 3, 4 (128 kernels). (b) ResBlock 5, 6 (256 kernels). (c) ResBlock 7, 8 (512 kernels).

Figure 4.8: The number of principal kernels required to accurately approximate each convolution layer in ResBlocks 3-8 in ResNet-18 (Results of ResBlocks 1 and 2 are discussed in Figure 4.1). The server model gradually attains a low-rank structure. However, even in the final model, half of the principal kernels are still required to approximate most layers.

4.3.4 Insights into `PriSM`

We now focus on providing further insights into `PriSM` by analyzing some of its key aspects. To this end, we first examine the low-rank structure of models during training, and pinpoint the cause behind the accuracy gap between fixed and random kernel dropout strategies in the orthogonal space. Thereafter, we study the sampling process and cost breakdown in the FL system

Model's rank during training. To analyze the server model's rank structure, we adopt a similar method as in [70] to calculate the required number of principal kernels to accurately approximate each layer as $2^{-\log(\sum_i p_i \log p_i)}$. Here, p_i is calculated as in Eq. (4.2) with $\kappa = 1$. Figure 4.8 shows the number of required kernels for each layer in ResNet-18 during full-model FL (Block$i$-j: j-th convolution layer in i-th ResBlock). We observe that a randomly initialized model is not low-rank, rather the server model attains a low-rank structure gradually. Therefore, sub-models with fixed top-k principal kernels inevitably cause reductions in the server model capacity. Furthermore, even at the end of the training, around half the principal kernels are still required to approximate most layers. In fact, some layers require even more principal kernels. Therefore, our probabilistic sampling scheme is essential in preserving the server model capacity during FL training with sub-models.

Kernel sampling profiling.

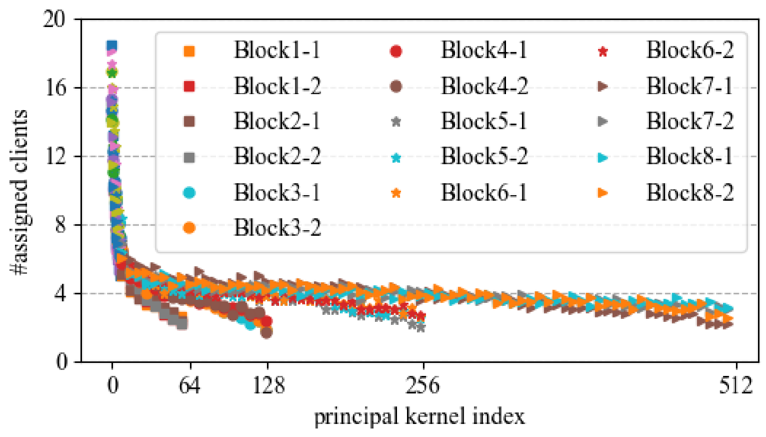


Figure 4.9: Average number of clients assigned for each orth. kernel during training. Kernels with large singular values get more chances to be chosen.

Figure 4.9 shows the average number of clients assigned for each orthogonal channel in one communication round. Each client trains a $0.2\times$ sub-model of ResNet-18 on CIFAR-10 with i.i.d distributions as in

Sec 4.3.2. We observe that each kernel is selected by at least one client in each round, indicating every kernel will be activated and trained on clients in each round. Furthermore, orthogonal kernels with larger σ get more chances to be chosen, which ensures sub-models on all clients consistently approximate the full model.

Runtime breakdown. In this section, we investigate the relative cost of SVD by breaking down the model training time into four stages: sub-model creation, local training, model aggregation, and obtaining orthogonal kernels (SVD). We choose a large model, ResNet-18, as the target model. We run the server and client process on NVIDIA RTX 5000. Table 4.7 lists each stage's average time in one communication round. We observe that the time of SVD is nearly negligible compared to the time spent on local training. Therefore, even for large models, the relative overhead of SVD is still very small.

Table 4.7: Training time breakdown for ResNet-18 on CIFAR-10.

stage	sub-model create	local train	aggregate	SVD
time	3.27 s	36.82 s	0.11 s	0.96 s

4.3.5 Ablation Study: Different Sampling Methods

We also conducted an additional study on the effects of different sampling strategies on the final serve model accuracy. In addition to the sampling method in PriSM (Eq (4.2)), we also try two other methods: `uniform` sampling and `softmax-based` sampling. In `uniform sampling`, we regard each channel as equally important with a uniform sampling probability. On the other hand, the `softmax-based` sampling assigns each channel a probability that is calculated by applying the softmax function to the singular values. Hence, it also assigns high sampling probabilities for channels with large singular values.

Table 4.8 shows the final validation accuracy of the global model. We observe that training with `uniform` sampling incurs significant accuracy drops in both i.i.d and non-i.i.d data distributions. It indicates that importance-aware sampling is crucial to avoid a large variation between sub-models and the full model. On the other hand, while the `softmax-based` sampling also considers each kernel's importance, the sampling method in PriSM performs better in choosing the right sub-model and covering the whole kernel space. We also study the effects of two hyperparameters: the number of local epochs and active clients. Each of these two parameters affects the sampling process when creating sub-models for clients. Specifically, given a fixed number of total iterations, FL training with a small number of local epochs per communication round performs a more frequent sampling process, thus making more orthogonal kernels to be selected and trained. Similarly, the FL training with a large number of active clients per round can also activate more orthogonal kernels. At a high level, fewer local epochs (given fixed total iterations) or more active clients in each round result in higher sampling probability for each kernel. Therefore, training performance is improved as shown in the results.

Table 4.8: Global model accuracy with different sampling strategies.

sampling	i.i.d	non-i.i.d ($\alpha = 1$)	non-i.i.d ($\alpha = 1$)
`uniform`	69.54 ± 0.21	67.28 ± 0.51	57.5 ± 0.67
`softmax-based`	84.74 ± 0.49	83.19 ± 0.43	72.69 ± 0.62
PriSM	90.57 ± 0.25	89.36 ± 0.21	80.72 ± 0.59

4.4 Conlusion

We have considered a practical yet under-explored problem of federated learning in a resource-constrained edge setting, where no participating client has the capacity to train a large model. As our main contribution, we propose the `PriSM` training methodology, that empowers resource-limited clients by enabling them to train smaller sub-models, while still allowing the server to reconstruct the full model. Importantly, `PriSM` utilizes a novel sampling approach to obtain sub-models for the clients, all of which together provide near-full model coverage. `PriSM` further improves memory efficiency by exploiting low-rank structure in intermediate activations. Our extensive empirical results on diverse models and datasets demonstrate that `PriSM` performs significantly better than the prior baselines, especially when each client can train only a very small sub-model. For instance, compared to full-model training, `PriSM` only incurs $\sim 2\%$ accuracy drop for ResNet-18/CIFAR-10 when clients train only $0.2\times$ sub-models, while yielding an accuracy improvement of up to 10% compared to existing works. We further present insights into performance gains of `PriSM` compared to other baseline methods and demonstrate the necessity of importance-aware kernel sampling in sub-model training.

Chapter 5

Conslusion

This thesis aims to mitigate the privacy-utility-complexity trilemma in private machine learning. In particular, this thesis focuses on two directions: efficient private machine learning in private and public environments, and efficient private machine learning in federated settings.

Efficient Private Machine Learning in Private and Public Environments. This thesis first investigates a generic setting with a private and a public execution environment. The private environment provides strong data protection during training and inference. However, the private environment incurs additional complexity for protection or lacks sufficient computing capacity, leading to low computing performance. The public environment features high-end computing resources (e.g., GPUs), which offer much faster computing compared to the resource-constrained private environment. On the other hand, the public environment exposes data or models to the public without any protection.

With the heterogeneous setting, this thesis proposes a methodology, `AsymML`, that effectively exploits their advantages: strong protection in the private environment, and fast execution in the public environment. `AsymML` develops a unique data decomposition method that distributes data and computation into the private and public environment. Specifically, `AsymML` secures most information in the private environment with a low-dimensional representation, and offloads residual information into the public environment.

`AsymML` further decomposes the computation flow (models' forward and backward passes) to minimize complexity in the private environments, and minimize information leakage in the public environments. Along this direction, this thesis further proposes a fully asymmetric computation flow, `Delta`, to mitigate the bottleneck of inter-environment communication. `Delta` features two fully separated data flows, with one associated with the private environment and another one associated with the public environment. Compared to `AsymML`, `Delta` trains two separate models. A small model learns the low-dimensional representation in the private environments; a large model learns the high-dimensional residual representation in the public environment. Final predictions are obtained by aggregating the predictions from each model. Therefore, communication is minimized by eliminating layer-wise inter-environment data exchanges as in `AsymML`. The proposed model training and inference framework can be generalized to many real-world settings with private and public environments, such as systems with TEEs/GPUs, and client-server setups.

Efficient Private Machine Learning in Federated Settings. The second part of this thesis aims to mitigate the privacy-utility-complexity trilemma in federated learning (FL) settings. This part considers a typical FL setting of training large models at the resource-constrained edge.

This thesis proposes a new sub-model training methodology that reduces training complexity while still preserving models' utility as full-model training. The proposed method, `PriSM`, creates different low-rank approximations of the original model for different participating clients via performing a sampling process on the original model. The sampling process ensures each sub-model approximates the original model so that multiple clients still share similar initial model states in every training cycle. On the other hand, owing to the sampling process, different parts of the original model are distributed to multiple clients. Therefore, it ensures all sub-models together still provide nearly full coverage of the original models, leading to much-improved model utility. Compared to other compression methods, `PriSM`, though trains compressed models on clients, can still recover the full-model capacity on the server side.

References

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

[2] Ramy E Ali, Jinhyun So, and A Salman Avestimehr. “On polynomial approximations for privacy-preserving and verifiable relu networks”. In: *arXiv preprint arXiv:2011.05530* (2020).

[3] Orly Alter, Patrick O Brown, and David Botstein. “Singular value decomposition for genome-wide expression data processing and modeling”. In: *Proceedings of the National Academy of Sciences* 97.18 (2000), pp. 10101–10106.

[4] Shengwei An, Guanhong Tao, and et al. “Mirror: Model inversion for deep learning network with high fidelity”. In: *Proceedings of the 29th Network and Distributed System Security Symposium*. 2022.

[5] Mathieu Andreux, Jean Ogier du Terrail, Constance Beguier, and Eric W Tramel. “Siloed federated learning for multi-centric histopathology datasets”. In: *Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning*. Lima, Peru: Springer, 2020, pp. 129–139.

[6] Amazon AWS. *AWS AWS Nitro Enclaves*. `https://aws.amazon.com/ec2/nitro/nitro-enclaves/`. Accessed: 2023-11-09.

[7] Microsoft Azure. *Azure Confidential Computing Enclaves*. `https://learn.microsoft.com/en-us/azure/confidential-computing/confidential-computing-enclaves`. Accessed: 2023-11-09.

[8] Sara Babakniya, Souvik Kundu, Saurav Prakash, Yue Niu, and Salman Avestimehr. “Revisiting Sparsity Hunting in Federated Learning: Why does Sparsity Consensus Matter?” In: *Transactions on Machine Learning Research* (2023).

[9] Eugene Bagdasaryan, Omid Poursaeed, and Vitaly Shmatikov. “Differential privacy has disparate impact on model accuracy”. In: *Advances in neural information processing systems* 32 (2019).

[10] Randall Balestriero et al. “A spline theory of deep learning”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 374–383.

[11] Borja Balle and et al. “Privacy amplification by subsampling: Tight analyses via couplings and divergences”. In: *Advances in Neural Information Processing Systems* 31 (2018).

[12] Sudipto Banerjee and Anindya Roy. *Linear algebra and matrix analysis for statistics.* Vol. 181. Crc Press Boca Raton, 2014.

[13] James C Bezdek and Richard J Hathaway. "Convergence of alternating optimization". In: *Neural, Parallel & Scientific Computations* 11.4 (2003), pp. 351–368.

[14] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. *Language Models are Few-Shot Learners.* 2020. arXiv: 2005.14165.

[15] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. "Leaf: A benchmark for federated settings". In: *arXiv preprint arXiv:1812.01097* (2018).

[16] Nicholas Carlini and et al. "Membership inference attacks from first principles". In: *2022 IEEE Symposium on Security and Privacy (SP).* IEEE. 2022, pp. 1897–1914.

[17] *ChatGPT.* https://chatgpt.com/. Accessed: 2024-06-24.

[18] *ChatGPT Data Leakage.* https://openai.com/index/march-20-chatgpt-outage/. Accessed: 2024-06-24.

[19] Si Chen, Mostafa Kahla, and et al. "Knowledge-enriched distributional model inversion attacks". In: *Proceedings of the IEEE/CVF international conference on computer vision.* Virtual: IEEE, 2021, pp. 16178–16187.

[20] Christopher A Choquette-Choo and et al. "Label-only membership inference attacks". In: *International conference on machine learning.* Virtual: PMLR, 2021, pp. 1964–1974.

[21] Victor Costan and Srinivas Devadas. "Intel SGX explained". In: *Cryptology ePrint Archive* (2016).

[22] Hong-Ning Dai, Qiu Wang, Dong Li, and Raymond Chi-Wing Wong. "On eavesdropping attacks in wireless sensor networks with directional antennas". In: *International Journal of Distributed Sensor Networks* 9.8 (2013), p. 760834.

[23] 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.* Ieee. 2009, pp. 248–255.

[24] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. "Exploiting linear structure within convolutional networks for efficient evaluation". In: *Advances in neural information processing systems* 27 (2014).

[25] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* 2019. arXiv: 1810.04805.

[26] Enmao Diao, Jie Ding, and Vahid Tarokh. "HeteroFL: Computation and communication efficient federated learning for heterogeneous clients". In: *International Conference on Learning Representations* (2021).

[27] Cynthia Dwork. "Differential privacy". In: *International colloquium on automata, languages, and programming*. Springer. 2006, pp. 1–12.

[28] Cynthia Dwork, Aaron Roth, et al. "The algorithmic foundations of differential privacy". In: *Foundations and Trends® in Theoretical Computer Science* 9.3–4 (2014), pp. 211–407.

[29] Ege Erdoğan, Alptekin Küpçü, and A Ercüment Çiçek. "UnSplit: Data-Oblivious Model Inversion, Model Stealing, and Label Inference Attacks against Split Learning". In: *Proceedings of the 21st Workshop on Privacy in the Electronic Society*. 2022, pp. 115–124.

[30] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. "Model inversion attacks that exploit confidence information and basic countermeasures". In: *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 2015, pp. 1322–1333.

[31] Xinbo Gao, Fei Gao, Dacheng Tao, and Xuelong Li. "Universal blind image quality assessment metrics via natural scene statistics and multiple kernel learning". In: *IEEE Transactions on neural networks and learning systems* 24.12 (2013).

[32] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. "Safetynets: Verifiable execution of deep neural networks on an untrusted cloud". In: *Advances in Neural Information Processing Systems* 30 (2017).

[33] Filip Granqvist, Congzheng Song, Áine Cahill, Rogier van Dalen, Martin Pelikan, Yi Sheng Chan, Xiaojun Feng, Natarajan Krishnaswami, Mona Chitnis, and Vojta Jina. *pfl: simulation framework for accelerating research in Private Federated Learning*. Version 0.0. 2024. URL: https://github.com/apple/pfl-research.

[34] Lucjan Hanzlik, Yang Zhang, Kathrin Grosse, Ahmed Salem, Maximilian Augustin, Michael Backes, and Mario Fritz. "Mlcapsule: Guarded offline deployment of machine learning as a service". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 3300–3309.

[35] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. "Federated learning for mobile keyboard prediction". In: *arXiv preprint arXiv:1811.03604* (2018).

[36] Hanieh Hashemi, Yongqin Wang, and Murali Annavaram. "DarKnight: An accelerated framework for privacy and integrity preserving deep learning using trusted hardware". In: *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 2021, pp. 212–224.

[37] Chaoyang He, Murali Annavaram, and Salman Avestimehr. "Group knowledge transfer: Federated learning of large CNNs at the edge". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 14068–14080.

[38] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask R-CNN". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[39] 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.* 2016, pp. 770–778.

[40] Junyuan Hong, Haotao Wang, Zhangyang Wang, and Jiayu Zhou. “Efficient Split-Mix Federated Learning for On-Demand and In-Situ Customization”. In: *International Conference on Learning Representations.* 2022.

[41] Samuel Horvath, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos Venieris, and Nicholas Lane. “FjORD: Fair and accurate federated learning under heterogeneous targets with ordered dropout”. In: *Advances in Neural Information Processing Systems* 34 (2021).

[42] Yani Ioannou and et al. “Training CNNs with low-rank filters for efficient image classification”. In: *International Conference on Learning Representations (ICLR)* (2016).

[43] Max Jaderberg, Andrea Vedaldi, and et al. “Speeding up convolutional neural networks with low rank expansions”. In: *British Machine Vision Conference (BMVC)* (2014).

[44] Anil K Jain. *Fundamentals of digital image processing.* Prentice-Hall, Inc., 1989.

[45] Petr Jizba and Toshihico Arimitsu. “Observability of Rényi’s entropy”. In: *Physical review E* 69.2 (2004), p. 026128.

[46] Mostafa Kahla, Si Chen, Hoang Anh Just, and Ruoxi Jia. “Label-only model inversion attacks via boundary repulsion”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2022, pp. 15045–15053.

[47] 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”. In: *Foundations and Trends® in Machine Learning* 14.1–2 (2021), pp. 1–210.

[48] Gautam Kamath. *Approximate Differential Privacy.* http://www.gautamkamath.com/CS860notes/lec5.pdf. Accessed: 2024-03-29.

[49] Mikhail Khodak, Neil Tenenholtz, Lester Mackey, and Nicolo Fusi. “Initialization and regularization of factorized neural layers”. In: *International Conference on Learning Representations (ICLR)* (2021).

[50] Mikhail Khodak, Neil A Tenenholtz, Lester Mackey, and Nicolo Fusi. “Initialization and Regularization of Factorized Neural Layers”. In: *International Conference on Learning Representations.* 2020.

[51] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).

[52] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012).

[53] Jaewoo Lee and et al. “Scaling up differentially private deep learning with fast per-example gradient clipping”. In: *Proceedings on Privacy Enhancing Technologies* 2021.1 (2021).

[54] Jaewoo Lee and Daniel Kifer. “Scaling up differentially private deep learning with fast per-example gradient clipping”. In: *Proceedings on Privacy Enhancing Technologies* (2021).

[55] Christian Lesjak, Daniel Hein, and Johannes Winter. “Hardware-security technologies for industrial IoT: TrustZone and security controller”. In: *IECON 2015-41st Annual Conference of the IEEE Industrial Electronics Society*. IEEE. 2015, pp. 002589–002595.

[56] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, et al. “Towards fully autonomous driving: Systems and algorithms”. In: *2011 IEEE intelligent vehicles symposium (IV)*. IEEE. 2011, pp. 163–168.

[57] Ang Li, Jingwei Sun, Xiao Zeng, Mi Zhang, Hai Li, and Yiran Chen. “Fedmask: Joint computation and communication-efficient personalized federated learning via heterogeneous masking”. In: *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. 2021, pp. 42–55.

[58] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. “Pytorch distributed: Experiences on accelerating data parallel training”. In: *arXiv preprint arXiv:2006.15704* (2020).

[59] Zheng Li and Yang Zhang. “Membership leakage in label-only exposures”. In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, pp. 880–895.

[60] Zhaobo Lu, Hai Liang, and et al. “Label-only membership inference attacks on machine unlearning without dependence of posteriors”. In: *International Journal of Intelligent Systems* 37.11 (2022), pp. 9424–9441.

[61] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. “Learning word vectors for sentiment analysis”. In: *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*. 2011, pp. 142–150.

[62] Brendan McMahan, Eider Moore, and et al. “Communication-efficient learning of deep networks from decentralized data”. In: *Artificial intelligence and statistics*. PMLR. 2017, pp. 1273–1282.

[63] 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*. PMLR. 2017, pp. 1273–1282.

[64] Yiqun Mei, Pengfei Guo, Mo Zhou, and Vishal Patel. “Resource-Adaptive Federated Learning with All-In-One Neural Composition”. In: *Advances in Neural Information Processing Systems*. 2022.

[65] Midjourney. *Midjourney*. `https://www.midjourney.com/home`. Accessed: 2023-04-21.

[66] Fatemehsadat Mireshghallah, Mohammadkazem Taram, Prakash Ramrakhyani, Ali Jalali, Dean Tullsen, and Hadi Esmaeilzadeh. “Shredder: Learning noise distributions to protect inference privacy”. In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2020, pp. 3–18.

[67] Fan Mo, Ali Shahin Shamsabadi, and et al. “Darknetz: towards model privacy at the edge using trusted execution environments”. In: *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*. 2020, pp. 161–174.

[68] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. “Darknetz: towards model privacy at the edge using trusted execution environments”. In: *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*. 2020, pp. 161–174.

[69] Krishna Giri Narra, Zhifeng Lin, and et al. “Origami inference: Private inference using hardware enclaves”. In: *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE. 2021, pp. 78–84.

[70] Yue Niu, Ramy E Ali, and Salman Avestimehr. “3LegRace: Privacy-Preserving DNN Training over TEEs and GPUs”. In: *Proceedings on Privacy Enhancing Technologies* (2022).

[71] Yue Niu, Saurav Prakash, Souvik Kundu, Sunwoo Lee, and Salman Avestimehr. “Overcoming resource constraints in federated learning: Large models can be trained with only weak clients”. In: *Transactions on Machine Learning Research* (2023).

[72] OpenAI. *Introducing ChatGPT*. https://openai.com/blog/chatgpt. Accessed: 2023-04-21.

[73] OpenAI. *March 20 ChatGPT outage: Here’s what happened.* https://openai.com/blog/march-20-chatgpt-outage. Accessed: 2023-04-21.

[74] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Ulfar Erlingsson. “Scalable Private Learning with PATE”. In: *International Conference on Learning Representations*. 2018.

[75] Sundar Pichai. “Google’s Sundar Pichai: Privacy Should Not Be a Luxury Good”. In: *New York Times*. 2019.

[76] Do Le Quoc, Franz Gregor, Sergei Arnautov, Roland Kunkel, Pramod Bhatotia, and Christof Fetzer. “Securetf: A secure tensorflow framework”. In: *Proceedings of the 21st International Middleware Conference*. 2020, pp. 44–59.

[77] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. “Adaptive federated optimization”. In: *arXiv preprint arXiv:2003.00295* (2020).

[78] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[79] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. “Trusted execution environment: What it is, and what it is not”. In: *2015 IEEE Trustcom/BigDataSE/Ispa*. Vol. 1. IEEE. 2015, pp. 57–64.

[80] Sina Sajadmanesh, Ali Shahin Shamsabadi, Aurélien Bellet, and Daniel Gatica-Perez. “Gap: Differentially private graph neural networks with aggregation perturbation”. In: *USENIX Security 2023-32nd USENIX Security Symposium*. 2023.

[81] Ahmed Salem, Yang Zhang, and et al. “ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models”. In: *Network and Distributed Systems Security (NDSS) Symposium 2019*. 2019.

[82] Alexander Sergeev and Mike Del Balso. “Horovod: fast and easy distributed deep learning in TensorFlow”. In: *arXiv preprint arXiv:1802.05799* (2018).

[83] 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)*. IEEE. 2017, pp. 3–18.

[84] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[85] Lukas Struppek, Dominik Hintersdorf, and et al. “Plug and Play Attacks: Towards Robust and Flexible Model Inversion Attacks”. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 20522–20545.

[86] Hassan Takabi, Ehsan Hesamifard, and Mehdi Ghasemi. “Privacy preserving multi-party machine learning with homomorphic encryption”. In: *29th Annual Conference on Neural Information Processing Systems (NIPS)*. 2016.

[87] Chandra Thapa, Pathum Chamikara Mahawaga Arachchige, Seyit Camtepe, and Lichao Sun. “Splitfed: When federated learning meets split learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 2022, pp. 8485–8493.

[88] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. “Training data-efficient image transformers & distillation through attention”. In: *International conference on machine learning*. PMLR. 2021, pp. 10347–10357.

[89] Florian Tramer and Dan Boneh. “Slalom: Fast, verifiable and private execution of neural networks in trusted hardware”. In: *arXiv preprint arXiv:1806.03287* (2018).

[90] Aravind Vasudevan, Andrew Anderson, and David Gregg. “Parallel multi channel convolution using general matrix multiplication”. In: *2017 IEEE 28th international conference on application-specific systems, architectures and processors (ASAP)*. IEEE. 2017, pp. 19–24.

[91] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[92] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. “Split learning for health: Distributed deep learning without sharing raw patient data”. In: *arXiv preprint arXiv:1812.00564* (2018).

[93] 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*. 2020, pp. 11505–11515.

[94] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. “Image quality assessment: from error visibility to structural similarity”. In: *IEEE transactions on image processing* 13.4 (2004), pp. 600–612.

[95] Jenna Wiens and Erica S Shenoy. “Machine learning for healthcare: on the verge of a major shift in healthcare epidemiology”. In: *Clinical infectious diseases* 66.1 (2018), pp. 149–153.

[96] Di Xie, Jiang Xiong, and Shiliang Pu. “All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6176–6185.

[97] Di Xie, Jiang Xiong, and Shiliang Pu. “All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6176–6185.

[98] Kaiyu Yang, Jacqueline H Yau, Li Fei-Fei, Jia Deng, and Olga Russakovsky. “A study of face obfuscation in imagenet”. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 25313–25330.

[99] Wen Yang, Ziqian Zheng, Guanrong Chen, Yang Tang, and Xiaofan Wang. “Security analysis of a distributed networked system under eavesdropping attacks”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 67.7 (2019), pp. 1254–1258.

[100] Dezhong Yao, Wanning Pan, Yao Wan, Hai Jin, and Lichao Sun. “FedHM: Efficient Federated Learning for Heterogeneous Models via Low-rank Factorization”. In: *arXiv preprint arXiv:2111.14655* (2021).

[101] Xun Yi, Russell Paulet, Elisa Bertino, Xun Yi, Russell Paulet, and Elisa Bertino. *Homomorphic encryption*. Springer, 2014.

[102] Da Yu, Huishuai Zhang, and et al. “Do not let privacy overbill utility: Gradient embedding perturbation for private learning”. In: *arXiv preprint arXiv:2102.12677* (2021).

[103] Tianwei Zhang, Zecheng He, and Ruby B Lee. “Privacy-preserving machine learning through data obfuscation”. In: *arXiv preprint arXiv:1807.01860* (2018).

[104] Yuheng Zhang, Ruoxi Jia, Hengzhi Pei, Wenxiao Wang, Bo Li, and Dawn Song. “The secret revealer: Generative model-inversion attacks against deep neural networks”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 253–261.

[105] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. “Pyramid scene parsing network”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2881–2890.

[106] Wei Zhou, Yue Niu, and et al. “Sensitivity-oriented layer-wise acceleration and compression for convolutional neural network”. In: *IEEE Access* 7 (2019), pp. 38264–38272.