PMC: A Privacy-preserving Deep Learning Model Customization Framework for Edge Computing

BINGYAN LIU*, MOE Key Lab of HCST, Dept of Computer Science, School of EECS, Peking University
YUANCHUN LI*†, Microsoft Research
YUNXIN LIU, Microsoft Research
YAO GUO†, MOE Key Lab of HCST, Dept of Computer Science, School of EECS, Peking University
XIANGQUN CHEN, MOE Key Lab of HCST, Dept of Computer Science, School of EECS, Peking University

Deep learning models have been deployed to a wide range of edge devices. Since the data distribution on edge devices may differ from the cloud where the model was trained, it is typically desirable to customize the model for each edge device to improve accuracy. However, such customization is hard because collecting data from edge devices is usually prohibited due to privacy concerns. In this paper, we propose **PMC**, a **p**rivacy-preserving **m**odel **c**ustomization framework to effectively customize a CNN model from the cloud to edge devices without collecting raw data. Instead, we introduce a method to extract statistical information from the edge, which contains adequate domain-related knowledge for model customization. PMC uses Gaussian distribution parameters to describe the edge data distribution, reweights the cloud data based on the parameters, and uses the reweighted data to train a specialized model for the edge device. During this process, differential privacy can be enforced by adding computed noises to the Gaussian parameters. Experiments on public datasets show that PMC can improve model accuracy by a large margin through customization. Finally, a study on user-generated data demonstrates the effectiveness of PMC in real-world settings.

CCS Concepts: • **Human-centered computing → Ubiquitous and mobile computing**; • **Computing methodologies → Neural networks**; • **Security and privacy → Privacy protections**.

Additional Key Words and Phrases: edge computing, neural networks, domain adaptation, model compression, differential privacy

ACM Reference Format:
Bingyan Liu, Yuanchun Li, Yunxin Liu, Yao Guo, and Xiangqun Chen. 2020. PMC: A Privacy-preserving Deep Learning Model Customization Framework for Edge Computing. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 4, 4, Article 139 (December 2020), 25 pages. https://doi.org/10.1145/3432208

*The first two authors contributed equally. This work was done while Bingyan Liu was an intern at Microsoft Research.
†Correspondence to: Yuanchun Li, Yao Guo.

Authors' addresses: Bingyan Liu, lby_cs@pku.edu.cn, MOE Key Lab of HCST, Dept of Computer Science, School of EECS, Peking University, Beijing, China, 100871; Yuanchun Li, Yuanchun.Li@microsoft.com, Microsoft Research, Beijing, China, 100871; Yunxin Liu, Yunxin.Liu@microsoft.com, Microsoft Research, Beijing, China, 100871; Yao Guo, yaoguo@pku.edu.cn, MOE Key Lab of HCST, Dept of Computer Science, School of EECS, Peking University, Beijing, China, 100871; Xiangqun Chen, cherry@pku.edu.cn, MOE Key Lab of HCST, Dept of Computer Science, School of EECS, Peking University, Beijing, China, 100871.

2474-9567/2020/12-ART139 $15.00
https://doi.org/10.1145/3432208

1 INTRODUCTION

Convolutional neural networks (CNNs) have made remarkable progress in a wide range of computer vision tasks, such as image classification [17, 21, 46], object detection [12, 28] and media interpretation [9, 52]. Recently, it becomes increasingly attractive to deploy cloud-trained CNN models to edge devices, to protect user privacy and reduce inference latency. For example, Liu *et al.* [30] proposed to deploy deep learning (DL) models to smart cameras for real-time video surveillance. Hochsteler *et al.* [19] proposed to use deep learning in embedded vehicular systems. Many smartphone manufacturers and app developers have also started using on-device deep learning techniques to provide low-latency and privacy-preserved services [3, 54].

The deep learning models typically need to be customized before deploying to edge devices, in order to address the following two challenges: *resource constraints* and *data shift*. First, edge devices have inherent *resource constraints*, i.e. the computing power and memory space of edge devices are usually limited, thus the model size should be reduced to satisfy the specific constraints on edge devices. *Model compression* has become the most common solution to address this challenge. Typical model compression methods include weight pruning [16], filter pruning [25, 37], quantization [60, 61], and knowledge distillation [18, 22], etc. Second, the *data shift* challenge refers to the situation where the training data (in the cloud) is statistically different from the deployment environment (the edge device), which may lead to significant accuracy decrease. *Domain adaptation* techniques [31, 49] are designed to address this type of issue, which typically requires joint-training on data from the source domain and target domain. In our case, it means that the cloud must collect raw data from the edge device to train a specialized model.

Unfortunately, collecting data from edge devices is usually difficult and undesirable due to privacy concerns. Specifically, the raw data in edge devices may contain confidential user information and cannot be uploaded to the cloud. Thus, existing domain adaptation techniques that require raw data from edge devices cannot be applied. As a result, only model compression techniques can be applied in most of these model deployment processes. As the same edge-independent model is deployed to different edge devices, it results in inevitable accuracy loss.

To address this issue, we propose **PMC**, a novel cloud-to-edge model deployment framework to achieve ***p****rivacy-preserving* ***m****odel* ***c****ustomization*. Instead of collecting raw data from the edge device, we require the edge device to upload only privacy-insensitive statistics. Then we simulate an edge dataset in the cloud by reweighting training samples based on the collected statistics. By utilizing the reweighted data in the model compression and retraining process, we generate a new model with a smaller size and higher accuracy on the target edge device.

PMC faces several key challenges. The first challenge is how to characterize the data distribution on the edge device with a small set of statistics, given the fact that the edge dataset typically contains a set of unlabelled images. Second, how can we guarantee the privacy of uploaded statistics, i.e. make sure no private user information can be inferred from the statistics? Third, how do we use the statistics in the cloud to customize the model? Specifically, how can we generate accurate models for different edge devices based on these statistics?

PMC addresses the first challenge mainly based on the following insight: *By feeding a dataset into a CNN, the activation values of its intermediate channels usually convey information about the data distribution in the dataset, which can be roughly modeled with Gaussian distributions.* This phenomenon enables us to infer edge data distribution based on only a few statistics (Gaussian distribution parameters) collected from a CNN running on the edge device.

We further adopt *differential privacy* techniques to provide rigorous privacy guarantee. The reason is that although the statistics are already much less sensitive as compared to the raw data, they can still be exploited to reveal sensitive user information in some cases [45]. Specifically, we introduce the concept of *property-based adjacent datasets*, where two datasets differ on an arbitrary privacy property. By using the Laplace mechanism [44] that obfuscates the statistics by adding random noises drawn from a Laplace distribution, we can make it difficult to distinguish adjacent datasets (i.e. infer privacy properties) based on the statistics. The noise intensity is

controlled by the privacy budget ϵ. By default, our framework uses $\epsilon = 5$ (close to those used by large companies like Apple and Google), which offers strong privacy protection without sacrificing the effectiveness of the statistics.

To address the third challenge, we introduce a data reweighting scheme to generate a new dataset from the cloud dataset based on the collected edge statistics. The samples that are more statistically similar to the edge data are assigned higher weights and have a higher chance to be selected into the new dataset. The generated dataset can be regarded as a simulation of the edge data distribution. By pruning the cloud model based on the reweighted dataset, we can generate a compressed model with higher accuracy for the edge device.

To summarize, our approach includes the following four main steps: (1) *Extracting statistics* that can represent the edge data distribution, using a partial model deployed to the edge device. (2) *Adding noises* to the statistics to provide formal privacy guarantee, while keeping the statistics effective in representing data distribution. (3) *Reweighting cloud data* based on the privacy-preserving statistics extracted from the edge to generate a dataset that simulates the data distribution on the edge device. (4) *Retraining the model* based on the simulated edge dataset to generate a customized model for final edge deployment.

We evaluate PMC on simulated cloud-to-edge deployment scenarios, as it is hard to conduct experiments in real-world applications. We simulated each scenario by crafting a large cloud dataset and several small edge datasets using public-available datasets [42, 51, 56]. The edge datasets have different data distributions for simulating the real-world situations where the edge devices differ from each other in camera models, running environments, applications, *etc.* The results show that PMC outperforms traditional compression-based deployment pipelines and other personalization methods by up to 6.33% and 4.00%, respectively, while offering formal privacy guarantees.

To validate the effectiveness of our approach in real-world scenarios, we further conducted a study based on a dataset collected from 27 users. We asked each of our users to take several photos in different scenes (indoor, outdoor, portrait). The personally identifiable information (PII) in the photos was blurred to protect privacy. Since these images were not labelled and did not have any downstream applications, we could not compute the model customization accuracy as on the large-scale public datasets. Instead, we examined whether the activation values extracted from intermediate channels can be used to distinguish different scenes and different users, which is the theoretical basis of our approach. The results validated the effectiveness of the extracted activation values in describing edge data distribution, which indicates that our approach is also applicable in real-world cloud-to-edge deployment scenarios.

This paper makes the following key research contributions:

- We propose the idea of performing model customization without requiring uploading private user data from the edge. Instead of relying on raw data, we can use only data distribution statistics from the edge to perform efficient model customization. The distribution itself can be further protected by adding noises through differential privacy.
- Based on the idea, we design and implement a novel framework, named PMC, to achieve privacy-preserving cloud-to-edge model customization for CNNs. Using only distribution statistics from the edge data, PMC can effectively customize the cloud model for different edge devices by simultaneously adapting the model architecture and weights.
- We conduct extensive experiments based on public datasets and state-of-the-art model architectures. The results demonstrate the effectiveness of PMC in cloud-to-edge model customization. We also perform a user study on real-world data collected from 27 users, which demonstrates the effectiveness of our method in real-world settings.

2 BACKGROUND AND MOTIVATION

2.1 Convolutional Neural Networks

Convolutional neural network (CNN) is first introduced by LeCun *et al.* in 1998 [23]. Recently it has been widely used for image processing, classification, segmentation, *etc.*

Typically, a modern CNN (e.g. ResNet [17]) is organized in layers, including convolutional (Conv) layers, batch normalization (BN) layers, pooling layers, activation functions and fully connected (FC) layers, each of which represents a unique type of mathematical operations. Among them, the Conv layer is the core building block of a CNN. Its parameters consist of a set of learnable filters (or kernels) that employ a mathematical operation (called convolution) to capture local information. The input and output of a Conv layer are called feature maps (except the first Conv layer whose input is the image). A feature map has size H×W×C, where H, W, C are the height, weight, and the number of channels in the feature map. Each channel typically captures a unique feature, although not interpretable to humans. These terms (layer, filter, channel, *etc.*) will be used to explain our approach in the following sections.

2.2 Cloud-to-edge Model Deployment

Deploying cloud-trained deep learning models to edge devices is increasingly popular for edge computing scenarios in recent years. There are mainly two aspects to consider during the deployment: latency and accuracy. Model compression techniques, such as weight pruning and filter pruning [15, 25], can be used to reduce model inference latency on edge devices. As for the accuracy aspect, the issue is mainly about the statistical distribution difference between the cloud data and the edge data. Domain adaptation [31, 49] is the most relevant technique proposed to address such data shift issue.

However, existing domain adaptation techniques are not applicable in the cloud-to-edge deployment scenario, because it requires joint-training on cloud data and edge data, which is typically unacceptable due to privacy concerns. This paper aims to address this issue, i.e. *how can we adapt the cloud-trained model to the edge devices without compromising privacy?* The problem can be formulated as follows:

Definition 1. ***(Cloud-to-edge model customization)*** *Suppose D_{cloud} and D_{edge} are two datasets in the cloud and the edge device, respectively, and M_{cloud} is a CNN model trained on D_{cloud}. The goal of cloud-to-edge model customization is to generate a customized model M_{edge} based on D_{cloud}, D_{edge} and M_{cloud} specifically for D_{edge}.*

As compared with the domain adaptation problem that is widely discussed in the AI community [31, 49], we make the following three assumptions for the cloud-to-edge model customization problem:

(1) The dataset in the source domain (D_{cloud}) is large and has a mixture distribution, which contains some samples with the same distribution as the target domain (D_{edge}). This assumption is intuitive since the cloud would usually collect large-scale data from various sources.
(2) The customization process cannot utilize any sensitive information from D_{edge} due to privacy concerns. Here, sensitive information refers to any privacy property that may appear in the samples in D_{edge} (more formal definition in Section 3.3).
(3) Usually, the customized model M_{edge} should be smaller than M_{cloud} due to the resource constraints on edge devices.

3 THE PMC FRAMEWORK

This paper proposes PMC, a privacy-preserving approach to the cloud-to-edge model customization problem defined in Definition 1. In this section, we first give an overview of the PMC framework and then describe each of the core components in detail.

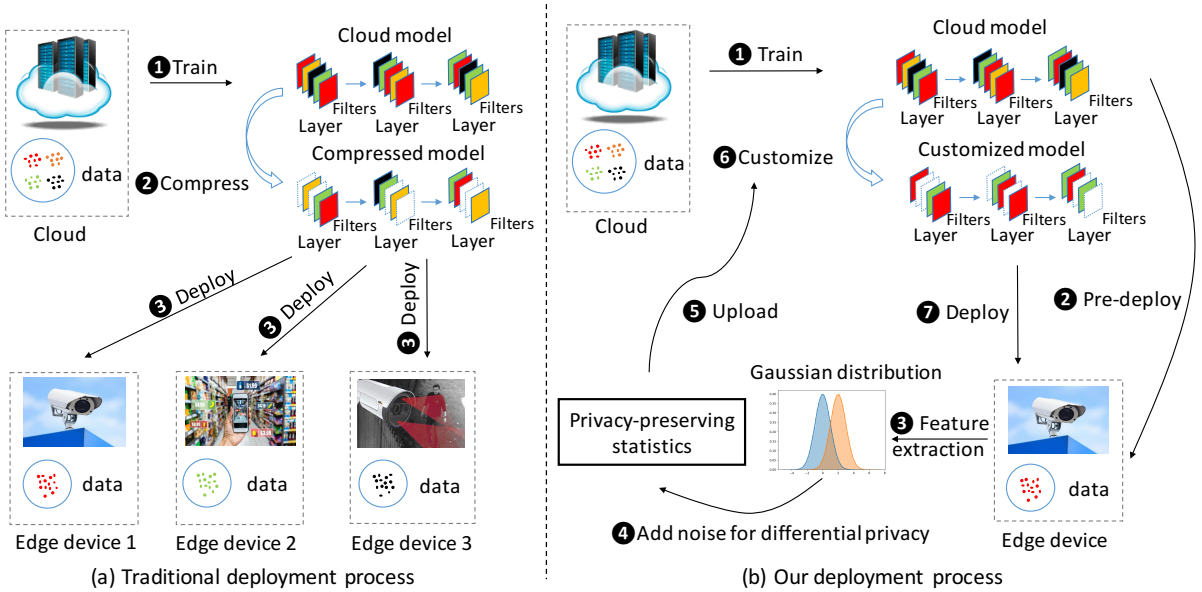

Fig. 1. Comparison of the traditional cloud-to-edge model deployment process and our deployment process. In the traditional process, the same compressed model is deployed to different edge devices with different data distributions (here the color represents the distribution), leading to potential loss of accuracy. Our method generates a customized model for each edge device based on privacy-preserving statistics extracted from the edge device, which can achieve higher accuracy.

3.1 Overview

Figure 1 presents the comparison between the traditional cloud-to-edge model deployment process and the PMC-enabled deployment process.

As illustrated in Figure 1(a), the traditional model deployment procedure mainly involves three steps, including (1) training the model with the data collected from various sources in the cloud, (2) compressing the cloud-trained model, and (3) deploying the compressed model to each edge device. The data distributions of different edge devices (represented with different colors) are not taken into consideration in the traditional deployment process. As a result, the models deployed to different devices are the same. Since the compressed model was trained based on the cloud data, it may not fit well on the edge data, which may come with a different data distribution. This will result in potential accuracy loss when the same model is deployed to different edge devices.

Taking the data distribution difference and privacy requirement into consideration, the workflow of our PMC framework (Figure 1(b)) includes the following steps. In *Step 1*, we train the model on the cloud as the traditional method does. Then the cloud-trained model is partially pre-deployed according to the resource limitation of a certain edge device (*Step 2*). In *Step 3*, we use the pre-deployed model to extract statistics (represented as Gaussian distribution parameters) that corresponds to the edge data distribution. To provide a rigorous privacy guarantee, we adopt differential privacy techniques and add computed noise to the extracted statistics before uploading them to the cloud (*Step 4* and *Step 5*). In *Step 6*, we customize the cloud model by (1) generating a new edge-specific dataset by reweighting the cloud data with the uploaded statistics and (2) compressing and retraining the model using the new dataset. Finally, in *Step 7*, we deploy the customized model to the edge device. As compared with the model generated with traditional deployment pipeline, our model is expected to achieve higher accuracy, since the weights and structures of our model are tailored for the edge data distribution.

Specifically, we first design a *statistics extraction function* $f(D_{edge})$ for describing the edge data distribution and then process it to compute $f'(D_{edge})$ with privacy guarantee. Then we upload the results generated with $f'(D_{edge})$ to the cloud and utilize them to compute a weight $w(x)$ for each sample x in D_{cloud}, such that we can generate a new dataset D'_{cloud} by sampling the cloud dataset based on the weight of each sample. With the new dataset, we finally compress and fine-tune the original model M_{cloud} to get the customized model M_{edge}. In the following sections, we will explain these steps in more detail.

3.2 Extracting Statistics

The goal of statistics extraction is to represent the edge data distribution with statistical data, such that we do not need to upload the raw edge data, which may contain user privacy. Because the cloud typically has a large dataset that can potentially cover the features in edge datasets, we may be able to reconstruct a dataset based on its distribution and simulate the respective dataset on each edge device. This is the key observation that leads to our proposed framework that can realize model customization without directly using the raw edge data.

Furthermore, as the distribution itself can still reveal critical user privacy, ideally we need to further protect the distribution as well. One reason for us to use the statistics to represent data distribution is that the statistics include only a few parameters (e.g. Gaussian distributions), which will be much easier to protect with techniques such as differential privacy.

Another key challenge remaining is how to estimate the distribution of edge data. Because we do not assume any prior knowledge about the edge and possible data distributions, we are unable to calculate the distribution straightforwardly (e.g. building a classifier to predict the distribution category). In fact, in many cases, the data distribution is hard to be explicitly described. To solve this issue, we resort to an indirect measurement method, which examines the activation values of a channel produced by feeding edge samples as inputs. Since a Conv filter in a CNN may capture a certain type of feature in the input sample (e.g. a filter's corresponding channel may be activated if the input image has a dark background), the pattern of the activation values of a channel produced by feeding a set of samples into the model may reflect an aspect of the overall data distribution of the dataset (e.g. whether the dataset mainly contains dark-background images). Thus, the question becomes how to statistically summarize the activation values of a channel produced by a dataset.

To start, we first describe how to compute the activation value of a channel produced by a single sample. Let $O(x) \in \mathbb{R}^{C\times H\times W}$ be the feature map extracted from a convolutional layer given input $x \in D_{edge}$. For a channel c in the feature map, the channel activation value is the mean of all values in the channel, which can be computed using a global average pooling operation:

$$z_c(x) = \frac{1}{H \times W} \sum_{m=1}^{H} \sum_{n=1}^{W} O_{c,m,n}(x) \tag{1}$$

In order to examine whether and how such channel-wise activation values can be used to estimate domain-specific data distributions, we inspect the activation values of two arbitrary channels for three sub-datasets belonging to two domains in the Office-Home [51] dataset. For each dataset D and channel c, we plot the histogram of $z_c(x)$ for all samples in D, as shown in Figure 2. We highlight two observations according to the figure. First, the datasets belonging to the same domain (Product_1 and Product_2) show similar histograms while datasets from different domains (Art and Product) show the discrepancy, which indicates that the channel activation values convey domain-specific data distribution information. Second, the activation values of a channel for a dataset can be roughly modeled as a Gaussian distribution, meaning that we can represent them with the mean μ and variance σ of the Gaussian distribution. Specifically:

$$\mu_c(D) = \frac{\sum_{x\in D} z_c(x)}{|D|}$$

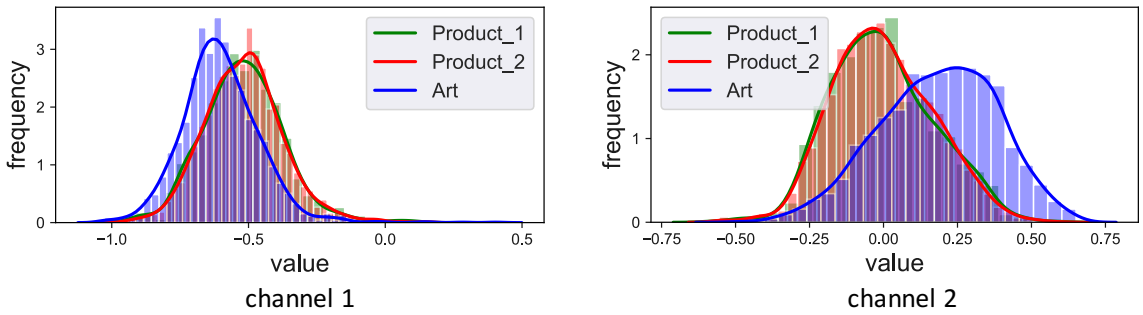

Fig. 2. Histograms of two randomly selected channels for three sub-datasets from the Office-Home dataset. Product_1 and Product_2 are two subsets of the Product domain. Art is another domain with a different distribution.

$$\sigma_c(D) = \frac{\sum_{x \in D} (z_c(x) - \mu_c(D))^2}{|D|}$$

Since a channel only describes a small aspect of the dataset, we usually need to use multiple channels to model the overall data distribution more precisely. Thus, we select k channels $C = c_1, c_2, ..., c_k$ from a convolutional layer's feature map and extract their corresponding Gaussian distribution parameters

$$\mu(D) = \{\mu_{c_1}(D), \mu_{c_2}(D), ..., \mu_{c_k}(D)\}$$
$$\sigma(D) = \{\sigma_{c_1}(D), \sigma_{c_2}(D), ..., \sigma_{c_k}(D)\}$$

as the representation of the dataset D. The selection of the layer and the channels will be discussed later in Section 4.6.

Combining $\mu(D)$ and $\sigma(D)$ together, we obtain the statistical information extracted from the edge device to describe the edge data distribution. Formally, the statistics extraction function can be defined as follows:

Definition 2. ***(Statistics extraction function)*** *Given a dataset D, the function to extract information from D to estimate D's distribution is given by:*

$$f(D) = \{\mu(D)\} \bigcup \{\sigma(D)\} \tag{2}$$

For the edge dataset D_{edge}, $f(D_{edge}) \in \mathbb{R}^{2k}$ can be utilized by the cloud to customize the model. In practice, to compute $f(D_{edge})$, we first pre-deploy the original cloud-trained model to the edge device and compute the statistics by feeding data samples into the model. Since we do not care about the model prediction accuracy in this step, the pre-deployed model can be a partial model that only contains the necessary channels (and their preceding layers), in order to save computation cost on the edge device. How to select channels for statistics extraction will be elaborated in Section 4.6.

3.3 Adding Noise for Differential Privacy

Directly uploading the edge-specific statistics (i.e. $f(D_{edge})$) to the cloud may still be problematic, since a curious server or a malicious third party may try to exploit privacy properties from the uploaded data. For example, an attacker may build a classifier to infer whether a certain person or object appears in the edge dataset [45]. Although the statistical information is intuitively privacy-preserving as it evens out most of the sensitive information, there is still a lack of rigorous or quantified privacy guarantees. Differential privacy is a common technique to offer such formal privacy guarantees. It has been used by prior work [1] to avoid privacy leakage through

gradients and learned parameters during training. In this paper, we also use differential privacy to prevent private information on the edge device from being inferred through the collected statistics.

Differential privacy can promise a formal and quantified privacy guarantee with a flexible setting. The definition of ϵ-differential privacy is described as follows:

Definition 3. ***(ϵ-differential privacy)*** *Suppose $\mathcal{A}$ is a randomized algorithm that takes a dataset as input. The algorithm $\mathcal{A}$ is said to provide ϵ-differential privacy if for any subset $S \subseteq Range(\mathcal{A})$ it holds:*

$$Pr[\mathcal{A}(D) \in S] \leq e^{\epsilon} * Pr[\mathcal{A}(D') \in S] \tag{3}$$

in which D and D' are adjacent datasets that differ on a single element (e.g. the record of one person). A smaller value of ϵ enforces a stronger privacy guarantee.

To enforce ϵ-differential privacy for a deterministic function g, a typical solution is to inject computed random noise to g. For example, the Laplace mechanism [44] is given by

Theorem 1. ***(Laplace mechanism)*** *Suppose function g with sensitivity $\Delta g = \max_{D,D'} \|g(D) - g(D')\|_1$. $\mathcal{A}$ is ϵ-differentially private if:*

$$\mathcal{A}(D) = g(D) + Lap(\Delta g/\epsilon) \tag{4}$$

where $Lap(\Delta g/\epsilon) \in \mathcal{R}^N$ is random noise drawn from a Laplace distribution centered at 0 with scale $\Delta g/\epsilon$.

In our setting, we want to enforce ϵ-differential privacy for the function $f(D)$ defined in Definition 2, which is used to extract statistics from an edge device. The formal privacy guarantee for $f(D)$ is nontrivial because there is no agreement on what exactly is private information on an edge device. For example, in an image dataset collected with a smart camera, most people would believe each individual's face is private, while some may say a car license plate that frequently appears in the scene is sensitive information, too.

Privacy property definition. In PMC, we leave the flexibility of defining privacy properties to the edge device owner. Specifically, the edge device owner can manually make a list of privacy properties $P = \{p_1, p_2, ...\}$. The properties can be informal, such as "Alice's face", "car license 1234", *etc*, as long as the edge owner can count or estimate how many data samples in the edge dataset contain each property.

Based on the privacy properties, we further refine the definition of adjacent databases in our setting:

Definition 4. ***(Property-based adjacent datasets)****. Given a privacy property p, D_p is the set of all samples in D that contain property p. $D - D_p$ is obtained by excluding all samples in D_p from D. We say D and $D - D_p$ are adjacent on property p.*

The goal of privacy protection is to disallow the cloud to infer privacy properties based on $f(D)$, where D is the edge dataset. In other words, the cloud should not be able to differentiate adjacent datasets D and $D - D_p$ based on the uploaded information. According to Theorem 1, such a goal can be achieved by adding a computed noise to $f(D)$.

Theorem 2. *Suppose D is the edge dataset and f is the function to extract information from D as in Definition 2, ϵ-differential privacy is enforced if a random noise drawn from distribution $Lap(\widehat{\Delta f}/\epsilon)$ is added to each value in $f(D)$ before uploading to the cloud, where*

$$\widehat{\Delta f} = \max_{p \in P} \frac{|D_p|}{|D|} * \sum_{c_i \in C} \{\Delta z_{c_i}(D) + \Delta z_{c_i}(D)^2\} \tag{5}$$

$|D|$ and $|D_p|$ are the total numbers of samples in D and D_p. $\Delta z_{c_i}(D) = z_{c_i}^{max}(D) - z_{c_i}^{min}(D)$ is the range of channel c_i's descriptor (as in Equation 1) across all samples in D.

Proving Theorem 2 is equivalent to proving that $\widehat{\Delta f}$ given by Equation 5 is an upper bound of f's sensitivity $\Delta f = \max_{p \in P} \|f(D) - f(D - D_p)\|_1$. We include the proof in the appendix.

Such mechanism is easy for edge device owners to apply based on their privacy requirements. First, the edge device owners need to prepare a dataset D to compute the statistics. They can optionally remove sensitive information from D to reduce privacy risks in the first place. Then the edge device owner needs to compute the Laplace distribution parameter $\widehat{\Delta f}$ based on their definition of privacy properties. Fortunately, according to Equation 5, they only need to estimate an upper bound of $\max_{p \in P} |D_p|$, which can be done by counting the number of data samples in D that contain a common privacy property. For example, if the edge device is a street camera that promises not to reveal individual car information, then the presence of each car would be a privacy property, and $\max_{p \in P} |D_p|$ is the maximum number of images in D with the same car in it. Similarly, $\max_{p \in P} |D_p|$ is the maximum number of samples provided by a single user if the edge data is collected from many end-users. By default, we use $\max_{p \in P} |D_p| = 1$ where each data sample is considered private while any two samples in D do not share common private properties, which is also the most common assumption in existing research [1] and products [2]. The final step is to calculate $f(D)$ and $\widehat{\Delta f}$ by feeding each sample $x \in D$ to the model once and decide the privacy budget ϵ (e.g. Apple and Google set ϵ to 2-8 in their products [2, 62]). For the edge dataset D_{edge}, the information uploaded to the cloud is

$$f'(D_{edge}) = f(D_{edge}) + Lap(\widehat{\Delta f}/\epsilon) \tag{6}$$

3.4 Customizing the Cloud Model

In this subsection, we customize the cloud model based on the uploaded statistics ($f'(D_{edge})$). The customization process includes a *data reweighting* step and a *customizing* step.

Data reweighting. We utilize $f'(D_{edge})$ to reweight each sample in the cloud to simulate the edge data distribution. Our solution is inspired by Sugiyama *et al.* [47] who used reweighting to address the *covariate shift* problem caused by different distributions. Specifically, we compute a weight score for each sample $x \in D_{cloud}$:

$$w(x) = \frac{p_{edge}(x)}{p_{cloud}(x)} \tag{7}$$

where $p_{edge}(x)$ and $p_{cloud}(x)$ represent the probability density of the edge and cloud datasets, respectively. $w(x)$ is higher if x conforms better to the edge data distribution. Unlike most existing reweighting schemes that devote to accurately estimate probability densities based on the raw data, we directly use the channel-wise Gaussian distribution whose probability density function is defined as

$$p(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{(x-\mu)^2}{2\sigma^2}) \tag{8}$$

Note that the cloud now already has the Gaussian distribution parameters uploaded from the edge device (i.e. $f'(D_{edge}) = \{\mu'(D_{edge})\} \bigcup \{\sigma'(D_{edge})\}$. We can then calculate the parameters of the same set of channels $C = c_1, c_2, ..., c_k$ for the cloud dataset (i.e. $\mu(D_{cloud})$ and $\sigma(D_{cloud})$), following the same process in Section 3.2. Based on these parameters, the cloud is able to compute the weight of each sample x according to Equations 7 and 8:

$$w(x) = \frac{\prod_{c_i \in C} p(x; \mu'_{c_i}(D_{edge}), \sigma'_{c_i}(D_{edge}))}{\prod_{c_i \in C} p(x; \mu_{c_i}(D_{cloud}), \sigma_{c_i}(D_{cloud}))} \tag{9}$$

Finally, we generate a new dataset D'_{cloud} by sampling data from the original cloud dataset according to the weight of each sample. The generated dataset would have similar data distribution to the edge device data.

Table 1. Detailed descriptions of the public datasets used in our experiment. The cloud dataset is simulated as the mixture of 80% of each subset, and the rest 20% of each subset is regarded as an edge dataset.

Dataset	Simulated scenario	#Classes	#Samples	Subset names & abbreviations
Office-31	different cameras	31	4,652	Amazon (A), DSLR (D), Webcam (W)
ImageCLEF_DA	different environments	12	1,800	Caltech-256 (C), ImageNet ILSVRC2012 (I), Pascal VOC 2012 (P)
Office-Home	different applications	65	15,500	Artistic images(Ar), Clipart images(Cl), Product images(Pr), Real-World images(Rw)

Customization. To customize the model based on the simulated dataset D'_{cloud}, we usually need to compress the model. As explained earlier, model compression is important to fit the model to resource-constrained edge devices. Filter pruning is a popular model compression technique, which mainly aims to prune the less-useful filters to reduce model size. Here the importance can be measured by simply calculating the filter's absolute weight sum [25] or approximating the change in the cost function with Taylor expansion [37]. In PMC, we use the Taylor-based method [37] but modify it to utilize our simulated dataset D'_{cloud} instead of the original cloud dataset D_{cloud}. This step can be omitted if the model does not need to be compressed.

Besides pruning, a fine-tuning step is further conducted to retrain the model in order to achieve good prediction performance for each edge. Here we also use the new dataset D'_{cloud} to fine-tune the original/pruned model. Finally, we obtain a customized model M_{edge} that is tailored for the edge deployment.

Note that we will not delve into more details on the pruning and fine-tuning steps in this paper, because these are relatively standard procedures taken in techniques such as domain adaptation. Interested readers can refer to related work for more information.

4 EVALUATION

We evaluated our approach by primarily looking at the following aspects:

(1) What is the accuracy of the customized model generated with our techniques? How does it compare with conventional methods? (§4.2, §4.3)
(2) How is the applicability of PMC? Specifically, what is the performance of PMC on some alternative scenarios? (§4.4)
(3) How do different privacy budgets affect the customized model's accuracy? (§4.5)
(4) How do different statistics extraction strategies affect the effectiveness of the extracted statistics in representing data distributions? (§4.6)
(5) Is the technique applicable and effective on real-world data from end-users? (§4.7)

4.1 Experiment Setup

It was hard to conduct experiments in real-world cloud-to-edge deployment scenarios, as we did not have access to any such applications. Thus, our experiments were mainly conducted on simulated scenarios based on public and collected datasets.

Simulated cloud and edge datasets. We selected three public datasets, including Office-31 [42], ImageCLEF_DA [56] and Office-Home [51], to simulate three cloud-to-edge deployment scenarios. The datasets were initially used to evaluate visual domain adaptation methods [31, 49], which address a similar problem as ours,

while having several different assumptions (as explained in Section 2.2). Each of the three datasets contains several (3 or 4) sub-datasets belonging to different domains. In our case, we assume the cloud dataset is large and has a mixture data distribution, and each edge dataset is relatively small and has a specific data distribution. Thus, we separate each sub-dataset into two parts: 80% for *cloud* and 20% for *edge*. The cloud dataset is constructed with all the *cloud* parts of the sub-datasets, and each *edge* part represents an edge dataset.

For example, Office-31 contains 4,652 images belonging to three subsets, including Amazon (A), DSLR (D), and Webcam (W), which correspond to online website images, digital SLR camera photos, and web camera photos respectively. Each image in the dataset is labeled with one of 31 office environment categories. Since the three subsets are mainly differentiated by the camera models, we can use this dataset to simulate the scenario where the edge devices have different camera models (the scenario is named *different cameras* for short). Specifically, we split each subset (e.g. the Amazon subset A) to a cloud part (e.g. A_{cl}) and an edge part (e.g. A_{ed}). The cloud dataset is combined with A_{cl}, D_{cl}, and W_{cl} and named as ADW_{cl}. Each of A_{ed}, D_{ed} and W_{ed} represents a dataset from an edge device.

Similarly, we used the ImageCLEF_DA dataset to simulate a scenario of *different environments*, where each edge device is placed in a unique environment. The Office-Home dataset is used to simulate *different applications* scenario, where each edge device may use the model to process different types of images. The detailed descriptions of the datasets can be found in Table 1.

User-generated dataset. We also created a real-world dataset using images collected from users. The purpose of this dataset is to examine whether our approach is effective in real-world scenarios. To create the dataset, we recruited 27 volunteers (mainly college students) and asked them to take several photos that simulate different edge device scenarios. Specifically, each of them was asked to take three indoor photos in their homes, three outdoor photos in their neighborhoods, and three portraits of themselves. Users had been carefully informed about the purpose of the study and the privacy risk, and they had the freedom to blur any sensitive information in the images (faces, numbers, etc.) and opt-out. The images were stored in secure storage and were only used internally. In the end, we obtained 243 photos (81 photos for each scene). We augmented the dataset by randomly flipping, rotating, and adjusting brightness, to better simulate different camera angles and illumination conditions.

Models. We tested our customization method on three typical and widely-used model architectures: VGG-16 [46], ResNet-18 and ResNet-50 [17]. VGG-16 is a vanilla convolutional neural network with direct connections across Conv layers. Since directly using VGG-16 on our task may lead to overfitting due to its intensive use of fully connected layers, we modify the model by replacing the original fully connected layers with a global average pooling layer. ResNet-18 and ResNet-50 are more complex models with many skip connections between Conv layers. They are organized as several *bottleneck* blocks to reduce the number of parameters. For all models, we used their pre-trained versions (i.e. trained with the ImageNet dataset [7]) before applying to the actual tasks to accelerate training.

PMC implementation. Unless otherwise noted, our PMC implementation in the experiments was based on the following configurations: in the statistics extraction phase, the statistics to describe edge data distribution were computed from 50 randomly-selected channels in a layer between 50% to 75% depth in the models. Other channel selection strategies will be discussed in Section 4.6. When adding noises to the statistics, the privacy budget ϵ was set to 5, which is a relatively strong privacy guarantee according to Apple [2] and Google [62]. Detailed analysis of the trade-off between accuracy and privacy budget will be presented in Section 4.5. In the model customization phase, all the experiments were conducted with the PyTorch [40] framework. The learning rate was set to 0.001 in the fine-tuning process, where a momentum of 0.9 and a weight decay of 0.0005 were adopted to accelerate convergence. We used different models on different datasets: VGG-16 on ImageCLEF_DA, ResNet-18 on Office-31, and ResNet-50 on Office-Home.

Table 2. The accuracy of the customized models generated by PMC and baseline methods on Office31 dataset. Here ADW_{cl} is the abbreviation of $A_{cl} + D_{cl} + W_{cl}$, which represents the cloud data. Each of A_{ed}, D_{ed}, and W_{ed} represents the data in a different edge device. PR is the pruning ratio. When PR is 0, it means the model doesn't need to be compressed, and the original model is compared as the baseline. Otherwise, the baselines are two state-of-the-art model pruning methods. #Params represents the number of parameters in the compressed model.

PR	Method	$ADW_{cl} - A_{ed}$	$ADW_{cl} - D_{ed}$	$ADW_{cl} - W_{ed}$	#Params
0%	*Original*	84.09%	94.78%	95.21%	11.18M
	PMC	**85.51%**	**97.98%**	**98.99%**	11.18M
10%	*L1-norm [25]*	82.24%	96.39%	94.46%	10.23M
	Taylor-based [37]	83.31%	95.58%	95.47%	9.99M
	PMC	**84.94%**	**97.59%**	**97.73%**	10.01M
30%	*L1-norm [25]*	79.90%	91.97%	92.44%	7.91M
	Taylor-based [37]	79.55%	92.98%	93.45%	7.60M
	PMC	**81.68%**	**93.98%**	**94.21%**	7.62M

Baselines. Without PMC, developers would not be able to access user data due to privacy concerns, and thus they can only use model compression techniques to generate models for edge devices. Thus, we used the following two model pruning methods as the baseline methods:

- *l1-norm pruning* [25]. The l1-norm pruning method measures the importance score of each filter by calculating its absolute weight sum. The filters with lower importance scores will be pruned. Note that this method does not require any input data as the score is only related to the filter weights.
- *Taylor-based pruning* [37]. The Taylor-based pruning method measures the importance of each filter based on its influence on the cost function estimated with Taylor expansion. In the process, the cloud data needs to be fed into the model to calculate the feature map that is used to measure the filter importance.

State-of-the-arts. The state-of-the-art methods include mobile personalization and domain transfer techniques. We compare the following two state-of-the-art methods with our approach:

- *AdaBN* [27]. AdaBN achieves personalization/adaptation by changing the statistics of the BN layers in the cloud-trained model only using local unlabeled data. This method does not need to upload any data and can be regarded as a privacy-preserving mobile personalization method.
- *AFN* [56]. AFN is a state-of-the-art deep domain transfer method widely used in the area of domain adaptation [38]. Note that domain transfer methods require both the source domain (cloud) data and target domain (edge) data to implement joint training, thus we can consider AFN as an upper bound on performance when ignoring privacy.

4.2 Accuracy of the Customized Models

The results achieved on the simulated datasets are shown in Table 2, Table 3, and Table 4. We highlight the following key observations from the results:

Comparison with the original model. We first compare the accuracy between the model generated by *PMC* and the original model trained with the cloud data (i.e. the PR=0% rows in the tables). PMC-generated models outperformed the original models in all simulated scenarios by a large margin. Specifically, the improvement is 1.42~3.78% on the Office31 dataset, 3.00~4.33% on the ImageCLEF_DA dataset, and 1.14~2.68% on the Office-Home

Table 3. The accuracy of the customized models generated by PMC and baseline methods on ImageCLEF_DA dataset.

PR	Method	$PCI_{cl} - P_{ed}$	$PCI_{cl} - C_{ed}$	$PCI_{cl} - I_{ed}$	#Params
0%	*Original*	72.00%	91.67%	85.67%	14.72M
	PMC	**75.00%**	**95.00%**	**90.00%**	14.72M
10%	*L1-norm [25]*	65.67%	88.33%	83.00%	12.36M
	Taylor-based [37]	72.00%	90.33%	85.00%	11.61M
	PMC	**72.67%**	**92.00%**	**85.67%**	11.63M
30%	*L1-norm [25]*	42.00%	69.33%	58.67%	8.51M
	Taylor-based [37]	60.33%	83.00%	74.00%	8.57M
	PMC	**65.67%**	**87.00%**	**80.33%**	8.83M

Table 4. The accuracy of the customized models generated by PMC and baseline methods on Office-Home dataset.

PR	Method	$ACPR_{cl} - Ar_{ed}$	$ACPR_{cl} - Cl_{ed}$	$ACPR_{cl} - Pr_{ed}$	$ACPR_{cl} - Rw_{ed}$	#Params
0%	*Original*	75.46%	73.54%	89.18%	84.39%	23.51M
	PMC	**78.14%**	**75.83%**	**90.42%**	**85.53%**	23.51M
10%	*L1-norm [25]*	72.58%	71.48%	86.81%	82.89%	20.78M
	Taylor-based [37]	71.75%	71.82%	87.39%	82.66%	20.35M
	PMC	**75.05%**	**74.91%**	**89.63%**	**84.85%**	20.35M
30%	*L1-norm [25]*	61.24%	68.27%	84.33%	75.89%	14.86M
	Taylor-based [37]	60.00%	64.95%	83.09%	77.96%	14.52M
	PMC	**64.95%**	**71.02%**	**86.13%**	**80.83%**	14.63M

dataset. The average improvement was 2.64%. The stable and significant accuracy improvement demonstrates PMC's effectiveness in adapting the cloud model to the edge data distribution.

Among the three datasets, the ImageCLEF_DA dataset achieved the highest accuracy improvement (3.00~4.33%). This is because that the ImageCLEF_DA dataset collected data from different environments that may have more diverse background features than others, which will harm the accuracy for the edge device in a specific environment. Therefore, adapting data distribution on this dataset brought a more positive effect.

Accuracy achieved with pruning. We also considered the cases where filter pruning was used to generate smaller models for the edge devices. The pruning ratio was set to 10% or 30% for all the methods. For each method, we pruned the model and fine-tuned the pruned model with 20 epochs.

From the results, we can see that *PMC* achieved consistently higher accuracy than baselines for all models, datasets, and pruning ratios, which means that our technique could effectively address the distribution mismatch problem between the cloud data and edge data. Specifically, the performance of VGG-16 on ImageCLEF_DA dataset was more significant. For example, on the 30% pruned version, *PMC* outperformed the *l1-norm* baseline by 17.67~23.67% and the *Taylor-based* baseline by 4~6.33%. We believe the low accuracy of the *l1-norm* method may be due to the unsuitable pruning metric, which is more likely to prune the useful filters for edge data, especially when the dataset (ImageCLEF_DA) has more diverse background features than others.

In some cases, the pruned models may even outperform the original models that have more parameters. For example, almost all of the pruned models generated by *PMC* with the 10% pruning ratio performed better than

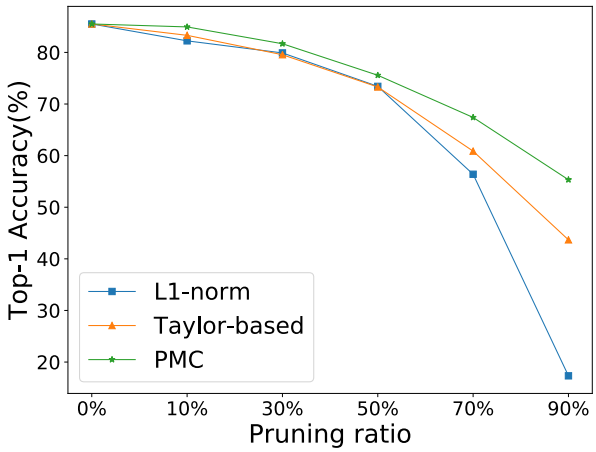

Fig. 3. The accuracy achieved by PMC and other pruning methods with different pruning ratios.

the original cloud-trained model. For example, in the $ADW_{cl} - D_{ed}$ task, the accuracy achieved by our pruned model is higher than the original cloud-trained model by nearly 3%. Other pruning methods could also improve accuracy, although the improvement is not as significant as ours. The reason might be that some filters that were not useful for the edge dataset were pruned, which in turn led to higher accuracy.

Accuracy improvement for different pruning ratios. Another interesting observation is that the accuracy improvement brought by PMC is related to the pruning ratio. For example, in the $PCI_{cl} - I_{ed}$ customization task in Table 3, PMC's accuracy improvement as compared with L1-norm pruning is 21.66% for PR=30% and 2.67% for PR=10%. To better inspect the relation between pruning ratio and accuracy improvement, we tested more pruning ratios on the Office31 $ADW_{cl} - A_{ed}$ customization task. The result is shown in Figure 3. We can notice that the more filters pruned, higher the accuracy improvement PMC achieved as compared with other pruning methods. Especially, when the pruning ratio was very high (90%), the traditional pruning methods performed poorly (accuracy became lower than 50%), while our approach could remain a much higher accuracy (around 60%). This is because a smaller model usually has lower knowledge capacity, thus the training data generated by PMC for the specific edge device becomes more important.

Analysis of the overall performance and the significant test. Based on aforementioned statistics, we observe that the overall accuracy improvement in the unpruned original model is not much significant. However, we would like to highlight that usually the original models need to be pruned before deployment on edge devices due to their resource constraints. Thus, the pruned versions are more suitable to represent the performance of PMC, which is obviously superior to other baselines, especially when pruning ratio is large (e.g. 90%, shown in Figure 3).

In addition, we conducted a simple significant test to check whether the difference between the results of two classifier (the original classifier and our customized classifier) over various datasets is non-random, which is widely used in the machine learning field. Here we used Wilcoxon signed-ranks test [6] to implement the significant test. The key idea is to compare the ranks for the positive and the negative differences and use them to check whether the null-hypothesis can be rejected. In our scenarios, we observe that the accuracy of our customized classifier outperforms the original one in all datasets, which means we do not introduce any negative differences. Therefore, according to the principle of Wilcoxon signed-ranks test, the sum of negative rank is 0 and we can always reject the null-hypothesis, which indicates the significance of PMC.

Table 5. The performance of PMC and other state-of-the-art methods on different datasets.

Dataset		AdaBN [27]	AFN [56]	PMC(w/o DP)	PMC
Office-31	$ADW_{cl} - A_{ed}$	84.52%	86.43%	85.44%	85.51%
	$ADW_{cl} - D_{ed}$	95.98%	99.19%	97.59%	97.98%
	$ADW_{cl} - W_{ed}$	96.47%	99.24%	98.99%	98.99%
ImageCLEF_DA	$PCI_{cl} - P_{ed}$	72.66%	76.67%	75.67%	75.00%
	$PCI_{cl} - C_{ed}$	92.33%	95.33%	95.00%	95.00%
	$PCI_{cl} - I_{ed}$	86.00%	91.67%	90.00%	90.00%
Office-Home	$ACPR_{cl} - Ar_{ed}$	75.88%	79.79%	78.14%	78.14%
	$ACPR_{cl} - Cl_{ed}$	73.77%	77.43%	76.06%	75.83%
	$ACPR_{cl} - Pr_{ed}$	89.18%	91.43%	90.98%	90.42%
	$ACPR_{cl} - Rw_{ed}$	84.62%	86.34%	85.53%	85.53%

4.3 Comparison with the State-of-the-art Methods

Here we compare the degree of privacy protection for different methods used in our evaluation. Specifically, AdaBN can ensure user privacy as it only needs to conduct a model inference process on the edge device, without any communication to outside. AFN needs to first upload the raw data to the cloud and then implement joint training with cloud data and uploaded data, which obviously violates user privacy. However, while this method will not be acceptable in our cloud-to-edge scenario, we believe AFN may act as an upper bound on performance with non-privacy protection. Besides, we implement a version of PMC without adding noise for differential privacy to examine the influence of the added noise.

Table 5 shows the performance of PMC and other state-of-the-art methods on different datasets. We can observe that: (1) PMC outperforms AdaBN in all of the datasets by up to 4.00%, which validates the superiority of the proposed approach in the context of privacy protection; (2) PMC can achieve comparable accuracy to AFN, an upper bound domain adaptation method, which means that we do not sacrifice much performance when taking privacy into consideration; (3) There is no significant accuracy gap to whether noises are added for differential privacy. Furthermore, in some datasets (e.g. $ADW_{cl} - D_{ed}$), PMC can exceed the non-noise version, which indicates that the noise sometimes may eliminate bias existed in the statistics.

4.4 Applicability for Alternative Scenarios

We further conducted experiments to simulate some alternative scenario designs, aiming at further examining the applicability of PMC. The designs mainly focus on the following two scenarios:

Customization for edge devices with labeled data. PMC is typically based on the assumption that data are unlabeled on edge devices. In this scenario, we assume some labeling effort can be made to provide labeled data for edge devices. Here we used the VGG-16 and ImageCLEF_DA dataset as an example. We partitioned the dataset into three parts: cloud data (60%), edge training data (20%), and edge testing data (20%). Then several different training schemes were taken to train the model: (1) *Edge train only*: we train the model with the edge training data. (2) *Cloud train+Edge fine-tune*: we first train the model with cloud data and then fine-tune it with edge training data. (3) *Cloud train+PMC*: we train the model with cloud data and use PMC to further adapt it. (4)

Table 6. The accuracy of different training schemes on ImageCLEF_DA dataset.

Method	$PCI_{cl} - P_{ed}$	$PCI_{cl} - C_{ed}$	$PCI_{cl} - I_{ed}$	**Edge Labeling**
Edge train only	20.83%	20.83%	22.50%	Required
Cloud train+Edge fine-tune	70.83%	95.83%	90.83%	Required
Cloud train+PMC	70.83%	96.67%	92.50%	Not required
Cloud train+PMC+Edge fine-tune	72.50%	97.50%	93.33%	Required

Table 7. The accuracy of the original and customized model in the sub-class setting of ImageNet2012.

Sub-class Setting of ImageNet2012	**Selected Classes**		**Unselected Classes**	
	Original	*Customized*	*Original*	*Customized*
1000 classes to 100 classes	77.42%	92.38%	79.82%	42.20%
1000 classes to 50 classes	75.04%	93.96%	79.82%	43.55%
1000 classes to 10 classes	73.60%	97.20%	79.64%	60.52%

Cloud train+PMC+Edge fine-tune: After the processing with PMC, we use the edge training data to fine-tune the model.

Table 6 demonstrates the results of different training schemes. First, we can clearly see that *edge train only* largely damages the training performance due to the overfitting problem [29]. Second, after the *cloud train*, PMC performs better than *edge fine-tune* as well as saving labeling effort, which indicates PMC can effectively utilize the specific knowledge in the cloud data and is more efficient. After customizing the model, developers can further fine-tune it with edge training data if they seek to higher performance.

Customization for edge devices with sub-class data. Here we want to evaluate PMC in another common scenario: sub-class customization. We start with the ImageNet2012 dataset (1000 classes) [7] and assume a dozen of these classes need to be customized. Specifically, we randomly selected 500 samples from each class of ImageNet2012 as the cloud data. For the edge device, we designed three settings: the edge data respectively owns 100, 50, and 10 classes of ImageNet2012 with 50 samples in each class. Note that the edge data should not have any overlapping with the cloud data. Based on the settings, we used PMC to customize the model and evaluated the performance on selected classes and unselected classes.

Table 7 summarizes the results achieved by the original and the customized model. We can observe the following phenomenons: (1) For selected classes, our customized model exceeds the original model by a large margin (over 10%), especially when the number of edge classes decreases, which validates the effectiveness of PMC in the sub-class scenario. (2) The accuracy on unselected classes drops a lot when we conduct customization. This is understandable since the goal of PMC is to adapt the model to be specific on the target selected data. The performance on other unselected data is not important because it has no contribution to the edge-related application.

4.5 Privacy Budget Analysis

PMC adds noise to the extracted statistics for stronger privacy guarantees. The intensity of the noise is controlled by the privacy budget ϵ. The smaller ϵ is, the larger noises are added, which indicates stronger privacy. In most of our experiments, ϵ is set to 5. However, developers may have different privacy budgets in practice [2, 62]. To better understand the trade-off between the model accuracy and the privacy budget, we used different ϵ values

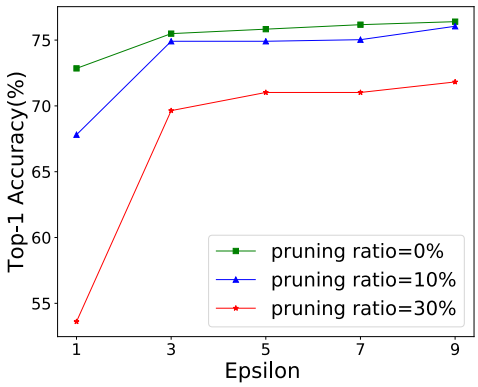

Fig. 4. The trade-off between the customized model's accuracy and the privacy budget (i.e. ϵ).

to generate the noise for the statistics and customize the cloud model. This experiment was conducted on the Office-Home $ACPR_{cl} - Cl_{ed}$ customization task, and the result is shown in Figure 4.

Based on the figure, we can draw following conclusions: First, a larger ϵ (weaker privacy guarantee) usually leads to higher accuracy, which is intuitive since it introduces smaller noises to the edge distribution statistics. Second, the accuracy improvement brought by increasing ϵ is not significant after ϵ is higher than 3, which indicates that the effectiveness of our edge statistics will not be significantly reduced when the noise intensity is below a threshold. Third, the model with a large pruning ratio is more sensitive to the privacy budget. For example, when we increased ϵ from 1 to 3, the accuracy improvement for the 30% pruning ratio was 16.03%, which was much higher than the 2.64% improvement for the model without pruning. These results can serve as a reference for the application developers and edge device owners to decide the privacy budget ϵ when using PMC for model customization. By default, it is recommended to set ϵ to 3~5 since it offers a strong privacy guarantee with very small accuracy loss. In other cases where strict privacy guarantee is not required, developers can use a very high ϵ, or even do not add any noise at all, to achieve the best accuracy.

In addition to the trade-off between privacy and effectiveness (accuracy), we would like to further highlight that PMC is extremely efficient. We do not need to label the data or consume massive computation resource since our approach only requires a model inference process with unlabeled user data to extract the statistics, which makes it more applicable to many resource-constrained scenarios.

4.6 Fine-grained Analysis of Different Channel Selection Strategies

Extracting statistics to represent edge data distribution is a core step in PMC. In our current implementation, statistics are extracted from 50 randomly selected channels in a Conv layer at the depth of 50% to 75%. However, there are other options to select the channels. Thus, we conducted an in-depth analysis of the channel selection strategies, in order to find the most effective channel selection strategy.

In this experiment, we assume that the goal of PMC was to customize a ResNet-18 model from the cloud with data $ACPR_{cl}$ to the edge device with data Ar_{ed} (constructed with the Office-Home dataset). PMC would select several channels according to a certain strategy, extract statistics from the channels on the edge device, and reweight cloud data with the statistics. The effectiveness of the selected channels can be measured as the ratio of samples in Art domain selected from $ACPR_{cl}$ in the cloud.

Our analysis considered the several variations of four channel selection strategies, including *select by type*, *select by layer*, *select by value*, and *number of neurons*. The results are shown in Figure 5 and explained as follows:

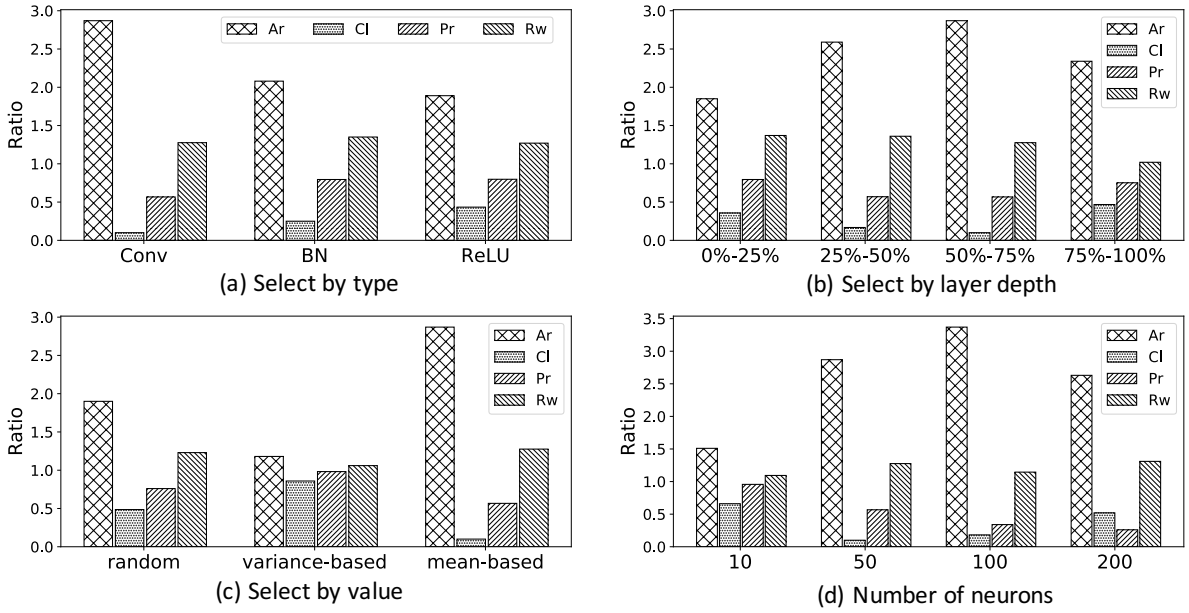

Fig. 5. The effectiveness of different channel selection strategies. The Y-axis is the ratio of samples selected from each domain using the statistics generated with the channels. The experiments were conducted on the Office-Home dataset with the ResNet-18 model. The statistics were computed with the Ar_{ed} subset, and the effectiveness of each channel selection strategy can be measured as the ratio of samples selected from the *Ar* domain. For example, in the first subfigure, selecting channels from Conv layers is better than from other types of layers. Note that the ratios may be greater than 1 because the same sample could be picked several times.

- *Select by type.* The channels can be selected from different types of layers, such as *Conv*, *BN* and *ReLU*. It is obvious that the channels selected from *Conv* layers are the most effective, since they can lead to a much higher ratio of samples in the *Art* domain selected in the cloud.
- *Select by layer depth.* The depth of layers to select channels from also matters. We split the layers into 4 equal ranges and found that the intermediate layers (50%~75% and 25%~50%) are more effective. This is probably because the shallow layers are not good at capturing features from the inputs, while features captured by deep layers are too relevant to the output labels. Instead, the features extracted by intermediate layers are more relevant to the general data distribution.
- *Select by value.* Inside a layer, one could also select channels based on their activation values. For example, we can select channels randomly, based on the variance of activation values, or based on the mean of activation values. We found that selecting channels with higher mean activation values was effective. However, we use the random selection strategy in our implementation because it is easier for the edge devices to employ (no profiling needed) and can also effectively distinguish different data distributions.
- *Number of neurons.* We also analyzed the effectiveness of selecting different numbers of channels for statistics extraction. We tested 10, 50, 100, and 200 channels and found that 50 channels were already enough to precisely describe the edge data distribution. Surprisingly, when the number of channels was larger (more than 200), the effectiveness of statistics decreased. The main reason is that the magnitude of noises added for differential privacy is related to the number of uploaded statistics (see Equation 5). Too

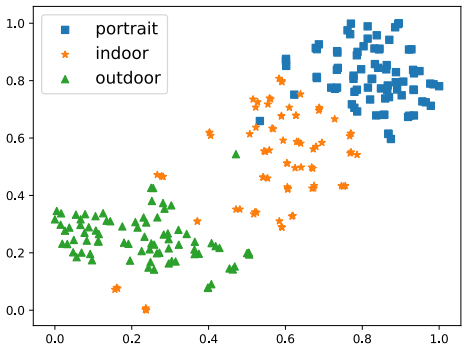

Fig. 6. The t-SNE visualization of statistics extracted for different scenes.

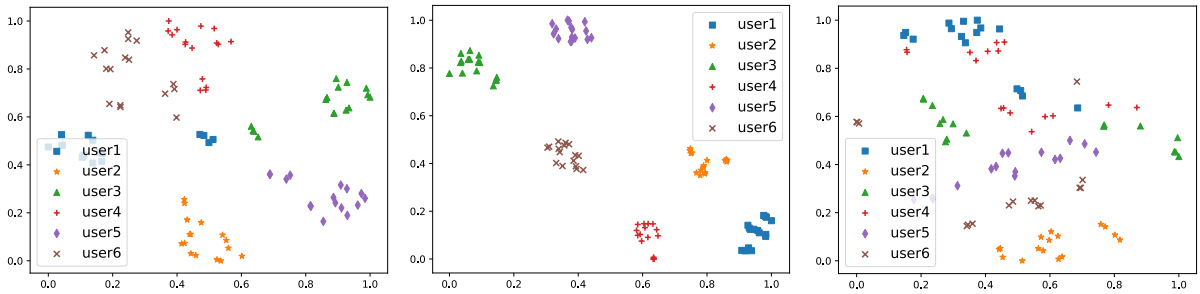

Fig. 7. The t-SNE visualization of statistics extracted for different users. Here we randomly picked 6 users for simplicity.

many channels selected would require large noises and harm PMC effectiveness. This is also one of the main reasons why we want to represent data distribution with a small number of statistics.

In summary, the best practice of channel selection in PMC is to select 50 to 100 channels from an intermediate Conv layer randomly or based on the magnitude of mean activation values. We also tested on other models and datasets and the conclusions were the same.

4.7 Study on User-generated Data

To further investigate the effectiveness of our approach in real-world settings, we conducted a qualitative experiment on the user-generated dataset (described in §4.1). Since the images in the dataset were not labeled and did not have any downstream applications, we could not evaluate the model customization performance as we did for the large-scale public datasets. Instead, we examine whether the statistics extracted from intermediate channels can effectively distinguish different domains (e.g. *different scenes* and *different users*), which is the theoretical basis of PMC. Specifically, we used the ResNet-18 model pretrained with ImageNet dataset [7] to extract statistics, and examined the effectiveness of the statistics by visualizing them with t-SNE [34]. If different domains can be clearly distinguished in the visualization, it means the statistics are effective in representing the data distribution knowledge, indicating PMC can effectively customize models based on such data.

The visualizations are shown in Figure 6 and Figure 7. Specifically, Figure 6 shows that the statistics extracted from each domain are clustered together. It means that PMC can be used to customize models to edge devices in different scenes (indoor, outdoor, portrait). In Figure 7, different users' images can be distinctly distinguished for indoor and portrait scenes, while the outdoor images can not be perfectly clustered. The reason might be that the outdoor images from each user are more arbitrary than in other scenes, and thus do not have a distinguishable data distribution. However, we believe that PMC would be effective in most cloud-to-edge deployment scenarios where each edge device has a unique data distribution.

5 RELATED WORK

This section presents a brief summary of existing literature related to our study.

5.1 Model Compression

Model compression has been proposed to reduce the size of a neural network by low-rank approximation [48, 57], parameter quantization [60, 61], knowledge distillation [18, 22] and network pruning. Among them network pruning has become a promising and widely applicable technique for accelerating and compressing CNNs, where the redundant parts of a CNN can be easily removed without incurring significant performance drop. Specifically, *weight pruning* [14–16, 58] and *filter pruning* [25, 29, 37] are two commonly used pruning techniques. For *weight pruning*, Han *et al.* [16] proposed an iterative method to discard the small weights whose values are below the predefined threshold. For *filter pruning*, Li *et al.* [25] measured the importance of each filter by calculating its absolute weight sum. Molchanov *et al.* [37] proposed a global ranking method based on Taylor expansion that approximated the change in the cost function induced by pruning network parameters.

Our customization method utilizes the *filter pruning* technique. However, we prune the model according to the unique properties of each edge device rather than the whole cloud dataset, which can be regarded as a customized pruning method.

5.2 Domain Adaptation

The goal of domain adaptation [38] is to overcome the shift across the source and target domains, which is usually achieved by learning domain-invariant features. Common methods of domain adaptation can be approximately divided into two categories. The first one is the *discrepancy-based* approach [31, 33, 50], utilizing different metric learning schemes to diminish the domain shift. For example, Long *et al.* [31] learned transferable features by minimizing the Maximum Mean Discrepancy (MMD) of their kernel embeddings. The second category is the *adversarial-based* approach [10, 32, 49], where a domain discriminator is leveraged to encourage the domain confusion by an adversarial objective. Inspired by the idea of adversarial learning, Ganin *et al.* [10] enabled the network to learn domain invariant representations adversarially by adding a domain classifier and back-propagating inverse gradients. Besides, multi-source domain adaptation [4, 20, 59], semi-supervised domain adaptation [13, 43] have also been widely studied. Domain adaptation aims to find domain-invariant features, rather than domain-specific features that are important in cloud-to-edge scenarios.

Moreover, existing domain adaptation techniques assume that the model developer can acquire data from both source and target domain for implementing the adaptation, which is not realistic in cloud-to-edge adaptation scenarios due to privacy concerns.

5.3 Privacy Protection for Edge, Mobile, and Deep Learning

Privacy protection is an important topic in mobile and edge computing, even before the emergence of on-device deep learning applications, such as permission access control [41], local computation [26], differential privacy [8], k-anonymity [24], *etc.* Many of the techniques and concepts are adopted to enhance privacy protection for deep

learning applications today. For example, DeepType [55] proposed to use a on-device input method for next-word prediction. MoRePriv [5] introduced a mobile service that can automatically mine the users' interest based on personal data. Differential privacy has also been used in various other deep learning applications. Abadi *et al.* [1] provided a differentially-private gradient descent method for training CNN, where the gradient is clipped and added noise to protect privacy. Papernot *et al.* [39] focused on private knowledge transfer where the student was trained to predict an output generated by noisy voting among the teachers. Geyer *et al.* [11] introduced differential privacy for federated learning systems, where edge devices collaborate to train a model without exchanging their data. Our approach also uses different privacy techniques, but we address a different problem - how to customize a model for an edge device with minimal privacy sacrifice and accuracy loss, which has different settings and challenges as compared with prior work.

Besides, federated learning [35] and other decentralized algorithms were proposed to cope with the privacy and personalization issue. For example, Wang *et al.* [53] designed an algorithm to determine the frequency of global aggregation so that the available resource was most efficiently used. Meurisch *et al.* [36] introduced a new decentralized and privacy-by-design platform (PrivAI) that enabled personalized AI services while preserving user privacy. However, all of them need the labeled user data and local training in the edge/mobile end, which may be infeasible in practice. Different from these methods, our approach only requires a model inference process with unlabeled user data to extract the statistics, without labeling and massive computation (training) effort.

6 DISCUSSIONS

Trade-off between customization and privacy. In our work, we use differential privacy techniques to provide formal privacy protection. However, there exists a trade-off here as stronger privacy guarantees (i.e., larger noises) will degrade the customization performance to some extent. In the future, we will attempt to explore other methods to ensure user privacy without affecting customization effectiveness.

Generalizability to other CNN models. PMC is only evaluated on VGG and ResNet models in this work. While there are many new CNN architectures proposed in recent years, we believe our method can be generalized to other CNN models, because the key idea of our approach, i.e. the statistics extracted from Conv channels can encode domain-related features, is common across all CNN architectures.

Other deep learning tasks. Our approach is focused on CNNs that are commonly used in computer vision tasks. However, there are other deep learning tasks that are based on other types of neural networks. For example, natural language processing tasks are usually based on recurrent neural networks (RNNs). Our approach currently cannot be applied to these models because they do not have Conv channels. We believe it is interesting to extend our method to these domains by analyzing what types of neurons in RNNs can reflect domain-related knowledge.

Other model compression methods. Besides the filter pruning technique used in this paper, there are many other model compression methods widely studied in the AI and edge computing fields, such as low-rank approximation [48, 57], parameter quantization [60, 61], and knowledge distillation [18, 22]. In the future, we will experiment with other compression methods to examine the performance of our approach.

Evaluation in real-world applications. Our current user study does not involve concrete performance comparison with other baselines. This is because the collected images were not labeled and do not have any downstream applications, which makes model training and customization unrealistic. Indeed, we agree that comparing PMC with other baselines in real-world settings is an important step to further evaluate PMC's benefits in practice. In our future work, we will look for ways to evaluate PMC for real-world applications.

7 CONCLUSION

In this paper, we have addressed the fundamental conflict between model customization and data privacy. We have demonstrated our solution through PMC, a novel framework to customize a CNN model from cloud to

edge with formal privacy guarantees. The key observation to motivate the framework is that the statistics extracted from intermediate neurons can be used to represent domain-related knowledge. Inspired by this, we present a set of techniques to achieve privacy-preserving model customization. Extensive simulation-based experiments demonstrate the effectiveness of the PMC framework on various network architectures. Finally, a study on real-world data from 27 users confirms the effectiveness of the extracted statistics in describing edge data distributions, indicating that our method will also be effective in real-world settings.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable feedback. This work was partly supported by the National Key Research and Development Program (2016YFB1000105) and the National Natural Science Foundation of China (61772042).

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 308–318.
[2] Apple. [n.d.]. Differential Privacy Overview. https://www.apple.com/privacy/docs/Differential_Privacy_Overview.pdf.
[3] Apple. [n.d.]. Private, on-device technologies to browse and edit photos and videos on iOS and iPadOS. https://www.apple.com/ios/photos/pdf/Photos_Tech_Brief_Sept_2019.pdf.
[4] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. 2010. A theory of learning from different domains. *Machine learning* 79, 1-2 (2010), 151–175.
[5] Drew Davidson, Matt Fredrikson, and Benjamin Livshits. 2014. Morepriv: Mobile os support for application personalization and privacy. In *Proceedings of the 30th Annual Computer Security Applications Conference*. 236–245.
[6] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research* 7, Jan (2006), 1–30.
[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.
[8] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*. Springer, 265–284.
[9] Thomas Forgione, Axel Carlier, Géraldine Morin, Wei Tsang Ooi, Vincent Charvillat, and Praveen Kumar Yadav. 2018. An Implementation of a DASH Client for Browsing Networked Virtual Environment. In *Proceedings of the 26th ACM international conference on Multimedia*. 1263–1264.
[10] Yaroslav Ganin and Victor Lempitsky. 2014. Unsupervised domain adaptation by backpropagation. *arXiv preprint arXiv:1409.7495* (2014).
[11] Robin C Geyer, Tassilo Klein, and Moin Nabi. 2017. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557* (2017).
[12] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 580–587.
[13] Yves Grandvalet and Yoshua Bengio. 2005. Semi-supervised learning by entropy minimization. In *Advances in neural information processing systems*. 529–536.
[14] Yiwen Guo, Anbang Yao, and Yurong Chen. 2016. Dynamic network surgery for efficient dnns. In *Advances in neural information processing systems*. 1379–1387.
[15] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).
[16] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*. 1135–1143.
[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[18] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).
[19] Jacob Hochstetler, Rahul Padidela, Qi Chen, Qing Yang, and Song Fu. 2018. Embedded deep learning for vehicular edge computing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 341–343.
[20] Judy Hoffman, Mehryar Mohri, and Ningshan Zhang. 2018. Algorithms and theory for multiple-source adaptation. In *Advances in Neural Information Processing Systems*. 8246–8256.

[21] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.
[22] Jangho Kim, SeongUk Park, and Nojun Kwak. 2018. Paraphrasing complex network: Network compression via factor transfer. In *Advances in Neural Information Processing Systems*. 2760–2769.
[23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[24] Kristen LeFevre, David J DeWitt, and Raghu Ramakrishnan. 2006. Mondrian multidimensional k-anonymity. In *22nd International conference on data engineering (ICDE'06)*. IEEE, 25–25.
[25] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710* (2016).
[26] Yuanchun Li, Fanglin Chen, Toby Jia-Jun Li, Yao Guo, Gang Huang, Matthew Fredrikson, Yuvraj Agarwal, and Jason I Hong. 2017. Privacystreams: Enabling transparency in personal data processing for mobile apps. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 3 (2017), 1–26.
[27] Yanghao Li, Naiyan Wang, Jianping Shi, Jiaying Liu, and Xiaodi Hou. 2016. Revisiting batch normalization for practical domain adaptation. *arXiv preprint arXiv:1603.04779* (2016).
[28] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*. 2980–2988.
[29] Bingyan Liu, Yao Guo, and Xiangqun Chen. 2019. WealthAdapt: A General Network Adaptation Framework for Small Data Tasks. In *Proceedings of the 27th ACM International Conference on Multimedia*. 2179–2187.
[30] Peng Liu, Bozhao Qi, and Suman Banerjee. 2018. Edgeeye: An edge service framework for real-time intelligent video analytics. In *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking*. 1–6.
[31] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I Jordan. 2015. Learning transferable features with deep adaptation networks. *arXiv preprint arXiv:1502.02791* (2015).
[32] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. 2018. Conditional adversarial domain adaptation. In *Advances in Neural Information Processing Systems*. 1640–1650.
[33] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. 2017. Deep transfer learning with joint adaptation networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2208–2217.
[34] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.
[35] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*. 1273–1282.
[36] Christian Meurisch, Bekir Bayrak, and Max Mühlhäuser. 2020. Privacy-preserving AI Services Through Data Decentralization. In *Proceedings of The Web Conference 2020*. 190–200.
[37] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440* (2016).
[38] Sinno Jialin Pan and Qiang Yang. 2009. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2009), 1345–1359.
[39] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Úlfar Erlingsson. 2018. Scalable private learning with pate. *arXiv preprint arXiv:1802.08908* (2018).
[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*. 8024–8035.
[41] Franziska Roesner, Tadayoshi Kohno, Alexander Moshchuk, Bryan Parno, Helen J Wang, and Crispin Cowan. 2012. User-driven access control: Rethinking permission granting in modern operating systems. In *2012 IEEE Symposium on Security and Privacy*. IEEE, 224–238.
[42] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. 2010. Adapting visual category models to new domains. In *European conference on computer vision*. Springer, 213–226.
[43] Kuniaki Saito, Donghyun Kim, Stan Sclaroff, Trevor Darrell, and Kate Saenko. 2019. Semi-supervised domain adaptation via minimax entropy. In *Proceedings of the IEEE International Conference on Computer Vision*. 8050–8058.
[44] Rathindra Sarathy and Krishnamurty Muralidhar. 2011. Evaluating Laplace noise addition to satisfy differential privacy for numeric data. *Trans. Data Privacy* 4, 1 (2011), 1–17.
[45] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. 2017. Membership Inference Attacks Against Machine Learning Models. In *2017 IEEE Symposium on Security and Privacy (SP)*. 3–18.
[46] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[47] Masashi Sugiyama, Taiji Suzuki, Shinichi Nakajima, Hisashi Kashima, Paul von Bünau, and Motoaki Kawanabe. 2008. Direct importance estimation for covariate shift adaptation. *Annals of the Institute of Statistical Mathematics* 60, 4 (2008), 699–746.
[48] Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. 2015. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067* (2015).
[49] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. 2017. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7167–7176.
[50] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. 2014. Deep domain confusion: Maximizing for domain invariance. *arXiv preprint arXiv:1412.3474* (2014).
[51] Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. 2017. Deep hashing network for unsupervised domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5018–5027.
[52] Vedran Vukotić, Christian Raymond, and Guillaume Gravier. 2016. Multimodal and crossmodal representation learning from textual and visual features with bidirectional deep neural networks for video hyperlinking. In *Proceedings of the 2016 ACM workshop on Vision and Language Integration Meets Multimedia Fusion*. 37–44.
[53] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. 2018. When edge meets learning: Adaptive control for resource-constrained distributed machine learning. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 63–71.
[54] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. 2019. A first look at deep learning apps on smartphones. In *The World Wide Web Conference*. 2125–2136.
[55] Mengwei Xu, Feng Qian, Qiaozhu Mei, Kang Huang, and Xuanzhe Liu. 2018. Deeptype: On-device deep learning for input personalization service with minimal privacy concern. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 4 (2018), 1–26.
[56] Ruijia Xu, Guanbin Li, Jihan Yang, and Liang Lin. 2019. Larger Norm More Transferable: An Adaptive Feature Norm Approach for Unsupervised Domain Adaptation. In *Proceedings of the IEEE International Conference on Computer Vision*. 1426–1435.
[57] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. 2015. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence* 38, 10 (2015), 1943–1955.
[58] Ziqi Zhang, Yuanchun Li, Yao Guo, Xiangqun Chen, and Yunxin Liu. 2020. Dynamic Slicing for Deep Neural Networks. *Proceedings of the 2020 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (2020).
[59] Han Zhao, Shanghang Zhang, Guanhang Wu, José MF Moura, Joao P Costeira, and Geoffrey J Gordon. 2018. Adversarial multiple source domain adaptation. In *Advances in Neural Information Processing Systems*. 8559–8570.
[60] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044* (2017).
[61] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. 2016. Trained ternary quantization. *arXiv preprint arXiv:1612.01064* (2016).
[62] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *Proceedings of the 21st ACM Conference on Computer and Communications Security*. Scottsdale, Arizona. https://arxiv.org/abs/1407.6981

A PROOF OF THEOREM 2

Proving Theorem 2 is equivalent to proving that $\widehat{\Delta f}$ given by Equation 5 is an upper bound of f's sensitivity Δf, i.e.

$$\max_{p \in P} \|f(D) - f(D - D_p)\|_1 \leq \max_{p \in P} \frac{|D_p|}{|D|} * \sum_{c_i \in C} \{\Delta z_{c_i}(D) + \Delta z_{c_i}(D)^2\} \tag{10}$$

Let $\textcircled{1}$ denote $\|f(D) - f(D - D_p)\|_1$ and D' denote $D - D_p$ in Equation 10. According to Equation 2, $f(D) = \{\mu(D)\} \bigcup \{\sigma(D)\}$, thus

$$\begin{aligned} \textcircled{1} &= \|\mu(D) - \mu(D')\|_1 + \|\sigma(D) - \sigma(D - D')\|_1 \\ &= \sum_{c_i \in C} |\mu_{c_i}(D) - \mu_{c_i}(D')| + |\sigma_{c_i}(D) - \sigma_{c_i}(D')| \end{aligned} \tag{11}$$

According to the definition, $\mu_c(D) = \frac{\sum_{x \in D} z_c(x)}{|D|}$, thus

$$\begin{aligned} |\mu_c(D) - \mu_c(D')| =& |\frac{\sum_{x \in D} z_c(x)}{|D|} - \frac{\sum_{x \in D'} z_c(x)}{|D'|}| \\ \leq& |\frac{\sum_{x \in D'} z_c(x) + |D_p| z_c^{max}}{|D|} - \frac{\sum_{x \in D'} z_c(x) + |D_p| z_c^{min}}{|D|}| \\ =& \frac{|D_p|}{|D|}(z_c^{max} - z_c^{min}) \\ =& \frac{|D_p|}{|D|}\Delta z_c(D) \end{aligned} \tag{12}$$

Similarly,

$$\begin{aligned} |\sigma_c(D) - \sigma_c(D')| =& |\frac{\sum_{x \in D} (z_c(x) - \mu_c)^2}{|D|} - \frac{\sum_{x \in D'} (z_c(x) - \mu_c)^2}{|D'|}| \\ \leq& |\frac{\sum_{x \in D'} (z_c(x) - \mu_c)^2 + |D_p|(z_c^{max} - z_c^{min})^2}{|D|} \\ & - \frac{\sum_{x \in D'} (z_c(x) - \mu_c)^2}{|D|}| \\ =& \frac{|D_p|}{|D|}(z_c^{max} - z_c^{min})^2 \\ =& \frac{|D_p|}{|D|}\Delta z_c(D)^2 \end{aligned} \tag{13}$$

Substituting Equation 12 and Equation 13 into Equation 11, then we have:

$$\textcircled{1} \leq \frac{|D_p|}{|D|} \sum_{c_i \in C} (z_c(D) + z_c(D)^2) \tag{14}$$

By substituting Equation 14 into Equation 10, then we can reach Equation 10. ■