Shredder: Learning Noise Distributions to Protect Inference Privacy

Fatemehsadat Mireshghallah Mohammadkazem Taram
Prakash Ramrakhyani* Ali Jalali† Dean Tullsen Hadi Esmaeilzadeh
{fmireshg,mtaram}@eng.ucsd.edu,prakash.ramrakhyani@arm.com,ajjalali@amazon.com,{tullsen,hadi}@eng.ucsd.edu
Alternative Computing Technologies (ACT) Lab
University of California San Diego *Arm, Inc. †Amazon.com, Inc.

Abstract

A wide variety of deep neural applications increasingly rely on the cloud to perform their compute-heavy inference. This common practice requires sending private and privileged data over the network to remote servers, exposing it to the service provider and potentially compromising its privacy. Even if the provider is trusted, the data can still be vulnerable over communication channels or via side-channel attacks in the cloud. To that end, this paper aims to reduce the information content of the communicated data with as little as possible compromise on the inference accuracy by making the sent data noisy. An undisciplined addition of noise can significantly reduce the accuracy of inference, rendering the service unusable. To address this challenge, this paper devises Shredder, an end-to-end framework, that, without altering the topology or the weights of a pre-trained network, learns additive noise distributions that significantly reduce the information content of communicated data while maintaining the inference accuracy. The key idea is finding the additive noise distributions by casting it as a disjoint offline learning process with a loss function that strikes a balance between accuracy and information degradation. The loss function also exposes a knob for a disciplined and controlled asymmetric trade-off between privacy and accuracy. While keeping the DNN intact, Shredder divides inference between the cloud and the edge device, striking a balance between computation and communication. In the separate phase of inference, the edge device takes samples from the Laplace distributions that were collected during the proposed offline learning phase and populates a noise tensor with these sampled elements. Then, the edge device merely adds this populated noise tensor to the intermediate results to be sent to the cloud. As such, Shredder enables accurate inference on noisy intermediate data without the need to update the model or the cloud, or any training process during inference. We also formally prove that Shredder maximizes privacy with minimal impact on DNN accuracy while the tradeoff between privacy and accuracy is controlled through a mathematical knob. Experimentation with six real-world DNNs from text processing and image classification shows that Shredder reduces the mutual information between the input and the communicated data to the cloud by 74.70% compared to the original execution while only sacrificing 1.58% loss in accuracy. On average, Shredder also offers a speedup of 1.79× over Wi-Fi and 2.17× over LTE compared to cloud-only execution when using an off-the-shelf mobile GPU (Tegra X2) on the edge.

CCS Concepts. • **Security and privacy → Privacy protections**; • **Theory of computation → Nonconvex optimization**; • **Computing methodologies → Neural networks**; • **Networks → Cloud computing**; • **Computer systems organization → Embedded systems**.

Keywords. Privacy; neural networks; deep learning; edge computing; cloud computing; inference; noise

ACM Reference Format:
Fatemehsadat Mireshghallah, Mohammadkazem Taram, Prakash Ramrakhyani, Ali Jalali, Dean Tullsen, and Hadi Esmaeilzadeh. 2020. 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 (ASPLOS '20), March 16–20, 2020, Lausanne, Switzerland*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3373376.3378522

ASPLOS '20, March 16–20, 2020, Lausanne, Switzerland

ACM ISBN 978-1-4503-7102-5/20/03…$15.00
https://doi.org/10.1145/3373376.3378522

1 Introduction

Online services that utilize the cloud infrastructure are now ubiquitous and dominate the IT industry [1–3]. The increasing processing demand of learning models [4, 5] has naturally pushed most of the computation to the cloud [6]. Coupled with the advances in machine learning, and especially deep learning, this shift has also enabled online services to offer a more personalized and more natural interface to the users [7]. These services continuously receive raw, and in many cases, personal data that needs to be stored, parsed, and turned into insights and actions. In many cases, such as home automation or personal assistants, there is a rather continuous flow of personal data to the service providers for real-time inference. While this model of cloud computing has enabled

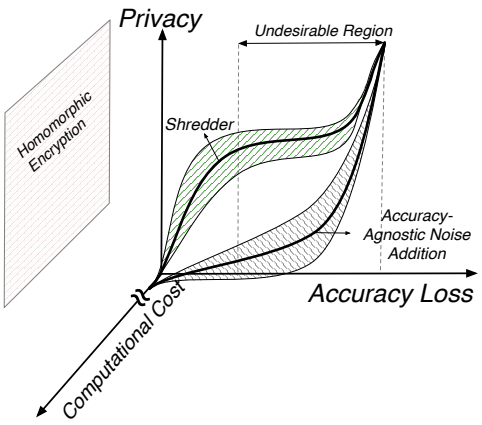


Figure 1. Design space for inference privacy and how Shredder fits.

unprecedented capabilities due to the sheer power of remote warehouse-scale data processing, it can significantly compromise user privacy. When data is processed on the service provider cloud, it can be compromised through side-channel hardware attacks (e.g., Spectre [8] or Meltdown [9]) or deficiency in the software stack [10]. But even in the absence of such attacks, the service provider can share the data with business partners [11] or government agencies [12]. Although the industry has adopted privacy techniques and federated learning [13] for data collection and model training [14, 15], scant attention has been given to the privacy of users who increasingly rely on online services for inference.

As Figure 1 illustrates, researchers have attempted to grapple with this problem by employing cryptographic techniques such as multiparty execution [16, 17] and homomorphic encryption [18–21] in the context of DNNs. However, these approaches suffer from a prohibitive computation and communication cost, with over three orders of magnitude added latency for CIFAR [22] for instance, exacerbating the already complex and compute-intensive neural network models. Worse still, this burdens additional encryption and decryption layers to the already constrained edge devices despite the computational limit being the main incentive of offloading the inference to the cloud.

This paper, as depicted in Figure 1, takes an orthogonal approach, Shredder, and aims to reduce the information content of the remotely communicated data through noise injection without imposing significant computational cost. However, as illustrated, noise injection can lead to a significant loss in accuracy if not administered with care and discipline. Shredder resolves this dilemma by finding the noise distributions through a disjoint offline learning process with a loss function that strikes a balance between information loss and accuracy loss. This loss function also exposes a knob for asymmetrically trading off a modest loss in accuracy for significant improvement in privacy as illustrated in Figure 1. As such, Shredder can use the conventional stochastic gradient descent–used for training machine learning algorithms–to learn the noise distributions.

Our central idea of learning the noise distributions enables Shredder to mathematically incorporate accuracy as well as the measure of privacy, and if available, the information that is expected to remain private in the loss function. The result is a collection of Laplace noise distributions that later during the inference is used by the edge device to scramble the communicated data. This *offline* process of learning the noise distributions does not require retraining the network weight parameters or changing its topological architecture. This non-intrusive approach is particularly appealing as most enterprise DNN models are proprietary, and changing a carefully crafted DNN topology and/or its millions of parameters is undesirable.

For inference, Shredder breaks the DNN between the cloud and the edge device while balancing out computation and communication. In this separate phase of inference, the edge device takes samples from the Laplace distributions that were collected to populates a noise tensor. Then, the edge device adds the noise tensor to the intermediate result that is sent over to the cloud. The number of parameters that Shredder learns in each epoch matches the number of elements in the intermediate result. Hence, Shredder only learns a much smaller collection of parameters–typically <100 KBytes–than the total number of weights–typically >100 MBytes–whose ratio is 0.16%. As such, the same model can be run on the same cloud on intentionally noisy data without the need for retraining the DNN or the added significant cost of supporting computation on encrypted data.

This problem of offloaded inference is different than the classical differential privacy [23] setting where the main concern is the amount of indistinguishability of an algorithm. That is to say, how the output of the algorithm changes if a single user opts out of the input set. In inference privacy, however, the issue is the amount of raw information that is sent out. Nonetheless, since Shredder employs a Laplace mechanism for generating noise, it is commensurate with differential privacy. As such, Shanon's Mutual Information (MI) [24] between the user's raw input and the communicated data to the cloud is used as a measure to quantitively discuss privacy.

We provide a formal formulation and show that Shredder maximizes privacy with minimal impact on DNN accuracy while the tradeoff between privacy and accuracy is controlled through a mathematical knob. This formal proof is backed by empirical analysis that shows Shredder reduces the mutual information between the input and the communicated data by 74.70% compared to the original execution with only 1.58% accuracy loss over six benchmark networks from text processing and image classification. It also offers an average speedup of 1.79× using Wi-Fi for communication, and 2.17× using LTE when an off-the-shelve mobile GPU (Jetson TX2) is used on the edge.

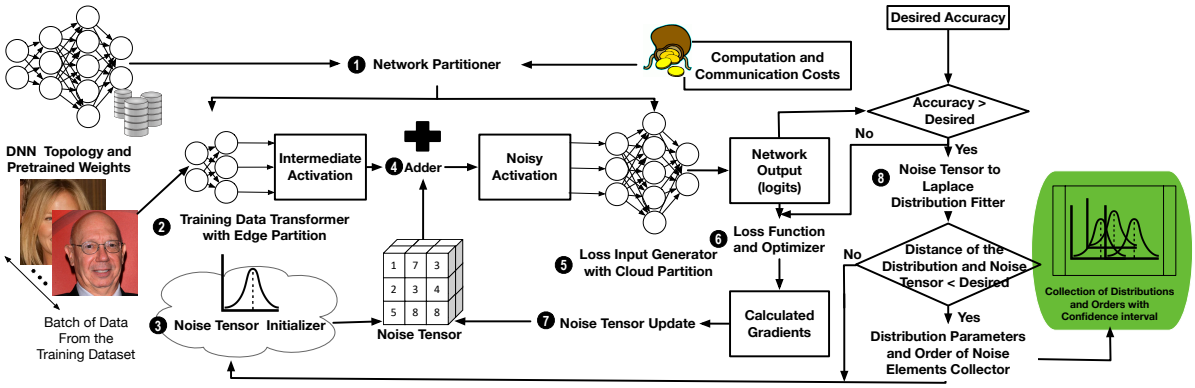


Figure 2. Workflow for offline learning of the noise distribution with Shredder.

With these encouraging results the paper contributes Shredder: casting accuracy-aware privacy protection as learning noise distributions through the same algorithm that trains the network, albeit, without retraining the network, while incorporating both privacy and accuracy in its loss.

2 Phase I: Learning the Noise Distributions

Shredder is an end-to-end non-invasive solution that consists of two *disjoint* phases: (1) learning the noise distributions offline and (2) noisy inference. As Figure 2 depicts, the first phase takes the DNN topology, its pre-trained weights, a training dataset, the computation and communication costs and acceptable loss in accuracy as inputs. The training dataset is the same as the one used to train the DNN. The output of the phase is a collection of noise tensor distributions coupled with an order for the elements of the tensor for the later phase of inference. This phase also determines which layer is the optimal choice for cutting the DNN to strike the best balance between computation and communication while considering privacy. The accuracy and privacy requirements are worked into the mathematics of the learning process. This section elaborates on the workflow of Shredder in this first phase, while the next section will discuss the second phase of inference.

❶ **Network partitioner.** The first step is to decide where to inject the noise to the DNN. The partitioner decides the layer at which the neural network should be bisected and offloaded to the cloud. This decision is based on the overall computation and communication costs of cutting the network at each layer, and the layer with the lowest cost will be chosen. The deeper the cutting point, the higher the privacy level, given a fixed level of loss in accuracy. This is due to the abstract representation of data in deeper layers of neural networks [25, 26]. We also show this observation in our experiments. Therefore, to maintain acceptable privacy, the partitioner is set to never cut at the input layer and it should include at least one computational layer. Nonetheless, there is a trade-off here, between computation costs and privacy, which will be further discussed in Section 6.2.3. The partitioning happens only once at the beginning of each learning process and the network is cut to two partitions, Edge and Cloud as depicted in Figure 2. The cut is made such that Cloud Partition starts with a convolution (or fully connected layer), and Edge Partition ends in the layer before that–which could be a pooling layer, or activation functions such as ReLU, depending on the DNN topology. The rationale is that the pooling layers reduce the data elements and the ReLU layers suppress some of the values, which naturally reduces the information content that is communicated to the cloud.

❷ **Training Data Transformer with Edge Partition.** To extract the training (inputs, output) pairs for learning the noise distributions, a batch of the training data is fed to Edge Partition, and the intermediate activation is attained. These intermediate activations constitute the input part of the pairs for the noise learning process. The output part of the pair is obtained in ❺. Note that in all of the stages of Shredder, DNN weights are constant and are not altered.

❸ **Noise tensor initializer.** Similar to the network partitioner, the noise tensor initializer is executed only once for each learning process. It initializes the noise tensor by sampling its elements from an initial Laplace distribution. The dimension of the noise tensor exactly matches the dimensions of the intermediate activation.

❹ **Adder.** The adder adds the noise tensor to the intermediate activation, element-wise. This stage imitates how the noise will be injected in the future inference runs.

❺ **Loss Input Generator with Cloud Partition.** To be able to go through the noise learning process, a set of outputs is required to pair with the intermediate activation generated in ❷. To generate the set of outputs, the noisy activations are fed to the Cloud Partition, and the logits (the outputs of the layer before the last classification layer of the neural network, which is usually softmax) are collected. We use logits because even normal stochastic gradient uses this layer to find the parameters. The last layer, typically softmax, uses logits to calculate the probability of each class and picks the one which has the highest probability as the classification label.

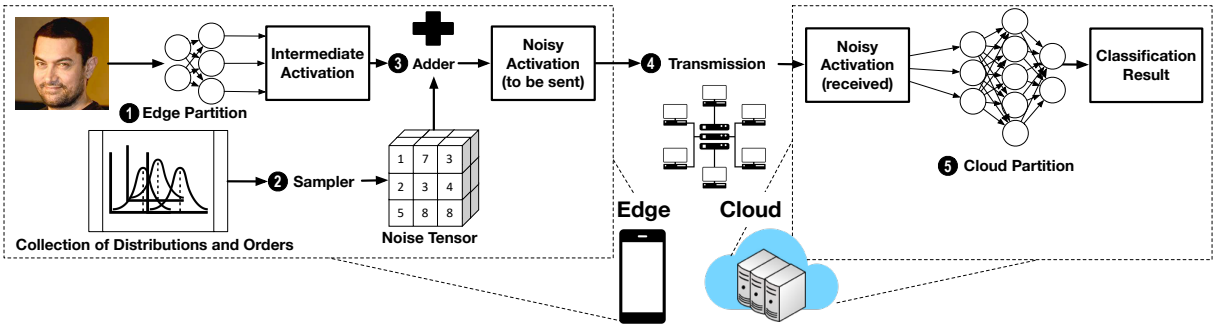


Figure 3. Workflow of noisy inference.

❻ **Loss function and optimizer.** The generated pairs of (input, output) from ❷ and ❺ are used to calculate the loss, and then fed to an optimizer to calculate the gradients and update the noise tensor using stochastic gradient descent. What is important here, is that the gradients are calculated only for the Cloud Partition, and then the adder, but they are not propagated any further.

❼ **Noise tensor update.** With the gradients calculated in the previous step, **only the noise tensor** is updated, and the DNN weights (even those of Cloud Partition) remain *unchanged*. We only calculate the gradients with respect to the weights to be able to update the noise tensor.

❽ **Noise tensor to Laplace distribution fitter.** During the noise learning process, after a given number of iterations of training and updating the noise tensor, the accuracy of the model is measured using a held-out set from the initial training dataset. If the accuracy is within the desired accuracy given by the user, the noise tensor is fitted to a Laplace distribution.

❾ **Distribution parameters and order of noise elements collector.** If the distance between the fitted distribution to the noise tensor and the actual noise tensor is smaller than a pre-determined amount, the parameters of the fitted distribution are saved. Also, the descending order of the elements of the noise tensor is saved to be used later during the future inference phase. The noise tensor elements, themselves, are discarded and only the order is preserved. The order is important to preserve the correlation between the elements. This stage also reports the confidence interval for the accuracy of the model as well.

This workflow is repeated until a certain number of noise distributions are collected. For our experiments, we collect 20 distributions. Now that the noise distributions are learned, the DNN can be used by the edge device for inference. The next section discusses this disjoint process of noisy inference.

3 Phase II: Noisy Inference

Figure 3 shows inference with a sampled noise tensor from the learned noise distributions. This process is very similar to the normal execution of the DNN except that a noise tensor is added to the intermediate activations that are sent to the cloud. Below, we discuss each step.

❶ **Edge Partition.** User's input data that s/he wants to classify is fed to the Edge Partition. The output is the intermediate activations.

❷ **Sampler.** In parallel with ❶, the Sampler takes in the collection of distribution and orders, i.e. the output of the offline learning phase, and uses that to generate different noise tensors for each inference pass. This makes predicting the noise tensor non-trivial for the adversary since for each input data, a different noise is generated **stochastically**. To generate the noise tensor, one distribution is picked randomly from the collection of distributions. Then, samples are drawn from this distribution to populate the noise tensor, which has the same dimensions as the intermediate activation. Then, when the noise tensor is populated, it's elements are rearranged, so as to match the saved order for that distribution. For this, the sampled elements are all sorted, and they are replaced according to the saved order of indices in the learning phase.

❸ **Adder.** This rearranged noise tensor is simply added to the intermediate activations. Note that, we do not rearrange the activations and the cloud part of the inference can continue as usual without the cloud knowing about the noise. The result is the noisy activations.

❹ **Transmission.** At this point, the noisy activation is transmitted over the network from the edge device to the cloud. The cloud is completely oblivious to the addition of noise or its nature.

❺ **Cloud partition.** The noisy activation is fed to the Cloud Partition, and the final classification labels are achieved. At this point, this result can be sent back to the user or be utilized in the cloud.

4 Noise Distribution Learning Formulation

This section delves deeper into the details of Shredder, starting from describing the problem formulation and the threat model, and how the trainable noise tensor fits into the context. In addition, this section describes the loss function and training process that Shredder uses for finding the desired noise tensor. We also describe our privacy model and the notions of

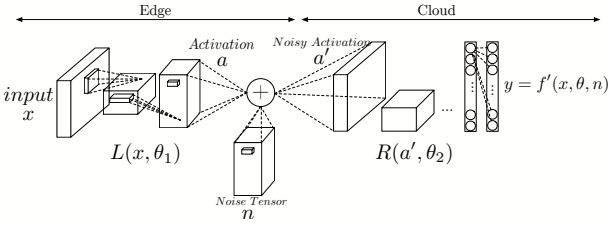


Figure 4. Mathematical modeling of noise injection and learning in Shredder.

privacy that we use. Our threat model is based on the fact that a large number of real-world DNNs cannot be fully executed on edge devices that range from battery-less devices to more powerful cellphones and tablets.

4.1 Threat Model

Given a pre-trained network $f(x,\theta)$ with K layers and pre-trained parameters θ, we choose a cutting point, $layer_c$, where the computation of all the layers $[0..layer_c]$ are made on the edge. We call this the local network, $L(x,\theta_1)$, where θ_1 is a subset of θ from the original model.

The remaining layers, i.e., $[(layer_c+1)..layer_{K-1}]$, are deployed on the cloud. We call this remote network, $R(x,\theta_2)$. This is shown in Figure 4. The f' in this image is the noisy network output (noisy logits) which are the outputs of the last layer of the neural network before going through softmax layer.

We assume the cloud is a deep-learning-as-a-service provider to whom users send their primary data(x) to get the label(y), where $y = f(x,\theta)$. The threat-model assumes the cloud, or anybody with access to the transmitted data, is untrusted in that it may try to extract information other than y from x. We assume the potential adversary(cloud/third-parties) is computationally unbounded. We don't assume limitations on number of queries to the untrusted cloud and its observations, nor the number of edge-devices that query the service. The local DNN(parameters/model) is known to the cloud/adversary. The user provides input x to the local network, and an intermediate activation tensor $a = L(x,\theta_1)$ is produced. Then, a noise tensor n is added to the output of the first part, $a' = a+n$. This a' is then communicated to the cloud where $R(a',\theta_2)$ is computed on noisy data and produces the result $y = f'(x,n,\theta)$ that is sent back to the user.

The objective is to find the noise tensor n that minimizes our loss function (Section 4.4). To be able to do this through a gradient-based method of optimization, we must find the $\partial y/\partial n$:

$$\frac{\partial y}{\partial n} = \frac{\partial f'(x,\theta,n)}{\partial n} = \frac{\partial R((a+n),\theta_2)}{\partial n} = \frac{\partial layer_{K-1}}{\partial Layer_{K-2}} \times \ldots \times \frac{\partial Layer_{c+1}}{\partial (a+n)} \times \underbrace{\frac{\partial (a+n)}{\partial n}}_{=\frac{\partial n}{\partial n}=1} \quad (1)$$

Since $L(x,\theta_1)$ is not a function of n, it is not involved in this equation . Gradient of R is also computed through chain rule as shown above. Therefore, the output is differentiable with respect to the noise tensor, which allows for the use of optimization methods like stochastic gradient descent.

4.2 Ex Vivo Notion of Privacy

To measure the privacy, we look at how much information is leaked from input of the network to the data sent across to the cloud. We define information leakage as the mutual information between x and a, i.e., $I(x;a)$, where

$$I(x;a) = \int_x \int_a p_{x,a} log_2 \frac{p_{x,a}}{p_x p_a} dx da. \quad (2)$$

Mutual information has been widely used in the literature for both understanding the behavior of neural networks [26, 27, 27–30], and also to quantify information leakage in anonymity systems in the context of databases [31–33]. We also use the negative of mutual information ($-MI$) as our main and final notion of privacy and call it ex vivo privacy. In our setting, we quantify the information between the user-provided input and the intermediate state that is sent to the cloud.

4.3 In Vivo Notion of Privacy

As the final goal, Shredder reduces the mutual information between x and a'; however, calculating the mutual information at every step of the training is too computationally intensive. Therefore, instead, we introduce an in vivo notion of privacy whose whole purpose is to guide our noise training process towards better privacy, i.e, higher $1/MI$. To this end, we use the reverse of signal to noise ratio ($1/SNR$) as a proxy for our ex vivo notion of privacy. Mutual information is shown to be a function of SNR in noisy channels [34, 35].

4.4 Loss Function

The objective of the optimization is to find the additive noise distribution in such a way that it minimizes $I(x,a')$, the mutual information, and at the same time maintains the accuracy of the primary task. In this point, there are two possible scenarios: a) we do not have private labels, which means Shredder's framework is not aware of what it should be protecting against, so it tries to remove any excess information and b) we have private labels, which means Shredder should apply noise which aims at obfuscating that private label, alongside removing other excess information. In the following subsections we will first explain the loss function insight and intuitions with the formulation for the first case, and then build on that and add an extra term to achieve the loss function for the second case.

4.4.1 No Private Labels Available. .

Although the two objectives mentioned previously (minimizing $I(x,a')$, and maintaining accuracy) seem to be conflicting, it is still a viable optimization, as the results suggest. The high dimensionality of the activations, their sparsity, and the tolerance of the network to perturbations [36, 37] yields such behavior.

The noise tensor that is added is the same size as the activation it is being added to. The number of elements in this tensor would be the number of trainable parameters in our method. Shredder initializes the noise tensor to a Laplace distribution with location parameter $\mu = 0$ (location is equivalent to mean of Gaussian distribution).

We evaluate the privacy of our technique during inference through ex vivo ($-MI$) notion of privacy. However, during training, calculating MI for each batch update would be extremely compute-intensive. For this reason, Shredder uses an in vivo notion of privacy which uses (SNR) as a proxy to MI [35]. In other words, Shredder incorporates SNR in the loss function to guide the optimization towards increasing privacy. We use the formulation $SNR = E[a^2]/\sigma^2(n)$, where $E[a^2]$ is the expected value of the square of activation tensor and $\sigma^2(n)$ is the variance of the noise we add. Given the in vivo notion of privacy above, our loss function would be:

$$-\sum_{c=1}^{M} y_{o,c} log(p_{o,c}) + \alpha \frac{1}{\sigma^2(n)} \tag{3}$$

Where the first term is cross-entropy loss for a classification problem consisting M classes ($y_{o,c}$ indicates whether the observation o belongs to class c and $p_{o,c}$ is the probability given by the network for the observation to belong to class c), and the second term is the inverse of variance of the noise tensor to help it get bigger and thereby, increase in vivo privacy (decrease SNR). α is a coefficient that controls the impact of in vivo privacy in training. Since the numerator in our SNR formulation is constant, which is because it is the expected value of activations and it is constant for noisy and original activations across the training dataset, we do not involve it in the calculations. The standard deviation of a group of finite numbers with the range $R = max - min$ is maximized if they are equally divided between the minimum, min, and the maximum, max. This is in line with our observations that show as we push the magnitude of the noise to be bigger, the in vivo privacy would also get bigger. Since we initialize the noise tensor with $\mu = 0$, some elements are negative and some are positive. The positive ones get bigger, and the negative ones get smaller, therefore, the standard deviation of the noise tensor becomes bigger after each update. That's why we employ a formulation opposite to L2 regularization [38], in order to make the magnitude of noise elements greater. So our loss becomes:

$$-\sum_{c=1}^{M} y_{o,c} log(p_{o,c}) - \alpha \sum_{i=1}^{N} |n_i| \tag{4}$$

This applies updates opposite to L2 regularization term (weight decay and α is similar to the decay factor), instead of making the noise smaller, it makes its magnitude bigger. The α exposes a knob here, balancing the accuracy/privacy tradeoff. In general, as the networks and the number of training parameters get bigger, it is better to make α smaller to prevent the optimizer from making huge updates and overshooting the accuracy.

4.4.2 Private Labels Available. An example of this case is gender classification and identity classification based on images of faces, which we will discuss more in Section 6.2.4. The user may want to classify whether a face is male or female, but s/he does not want the cloud to be able to identify who the person in the image is. In this case, the primary task is gender classification, and the private task is face identification. So, the aim is to minimize the mutual information and maintaining accuracy, similar to Section 4.4.1, but with an extra term, to minimize the accuracy of the private task, i.e. face identification. So, we reformulate our loss function to be:

$$-\sum_{c=1}^{M} y_{o,c} log(p_{o,c}) + \gamma \sum_{k=1}^{T} z_{o,k} log(p_{o,k}) - \alpha \sum_{i=1}^{N} |n_i| \tag{5}$$

Where the first term is cross-entropy loss for the primary classification problem consisting of M classes ($y_{o,c}$ indicates whether the observation o belongs to class c and $p_{o,c}$ is the probability given by the network for the observation to belong to class c), the second term is cross-entropy loss for the private classification problem consisting of T classes ($y_{o,k}$ indicates whether the observation o belongs to class k and $p_{o,k}$ is the probability given by the network for the observation to belong to class k) and the last term is the same term from Equation 4, to increase in vivo privacy (decrease SNR). α is a coefficient that controls the impact of in vivo privacy in training, similar to the previous subsection. γ acts similar to α, exposing a knob to control the effect of private label accuracy on the overall loss function.

4.5 Fitting Noise Tensor to Laplace distribution and Collecting Distribution and Samples

During training, whenever the accuracy of the model over the hold-out set of the training dataset exceeds the level desired by the user, the noise tensor is tested for being collected. We use Scipy's *stats* package to fit each learned noise tensor to a Laplace distribution. It's worth mentioning that this stage is executed offline. Then, the probability density function is calculated for the fitted distribution. Using the density function, the distribution collector calculates the Sum of Squared Errors (SSE) between the fitted distribution's histogram and the learned noise tensor's histogram. If the SSE is less than a threshold, the parameters of this distribution are collected. This threshold can be considered as a tunable parameter and differs from model to model and the different desired levels of accuracy and margin of error. The element orders of the noise tensor are also saved. By the element orders, the sorted indices of the elements of the flattened noise tensor are meant. For instance, if a tensor looks like $[[3.2, 4.8], [7.3, 1.5]]$, it's flattened version would be $[3.2, 4.8, 7.3, 1.5]$, and the sorted which is what the collector saves would be $[2,1,0,3]$.

4.6 Loss Function and Noise Learning Analysis

As Equation 4 shows, our loss function has an extra term, in comparison to the regular cross-entropy loss function. This extra term is intended to help decrease the Signal to Noise ratio (SNR). Figure 5 shows part of the training process on AlexNet, cut from its last convolution layer. The black lines show how a regular noise training process would work, with cross-entropy loss and Adam Optimizer [39]. As Figure 5a shows in black, the in vivo notion of privacy ($1/SNR$) decreases for regular training (privacy agnostic, the one without an extra term for decreasing SNR) as the training moves forward. For Shredder however, the privacy increases and then stabilizes.

This is achieved through tuning of the α in Equation 4. α is decayed by 0.1 at every 500 iterations to stabilize privacy and facilitate the learning process. If it is not decayed, the privacy will keep increasing and the accuracy would increase more slowly, or even start decreasing. The accuracy, however, increases at a higher pace for regular training, compared to Shredder in Figure 5b. It is noteworthy that this experiment was carried out on the training set of ImageNet, and when the training is finished, there is negligible degradation in accuracy for Shredder on the test set, in comparison to the regularly trained model.

4.7 Noise Tensor Sampling During Inference

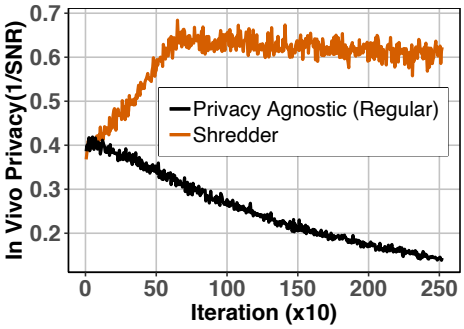
This phase is executed during inference, and it samples noise from the collected distributions in the previous phase, as discussed in Section 2. At this point, one of the distributions from the distribution collection (of 20 distributions) is selected randomly. Then, using samples from the chosen distribution, a flattened tensor (a vector) with the size of the noise is populated. The elements of this vector are then re-ordered to match the saved order of indices for that distribution and then reshaped to match the shape of the intermediate activations and get sent to the adder.

5 Formal Proof and Guarantees

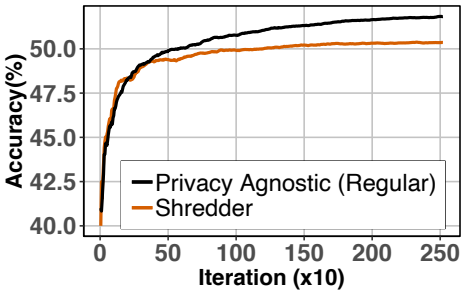
This section provides a formal formulation of the Shredder noise learning process and proves that it maximizes privacy with minimal effects on DNN accuracy.

Neural Network: Consider a deterministic function $f(\mathbf{x};\theta^*) \in \mathbb{R}^m$, where, $\mathbf{x} \in \mathbb{R}^p$ is the flatten input to the function and $\theta^* \in \mathbb{R}^r$ is the parameter that fully determines the function. Further, assume that $f(\cdot)$ is a composition of k functions, i.e. $f = f_1 \circ f_2 \circ \cdots \circ f_k$, where, $f_i(\cdot;\theta^{*(i)})$ is fully determined by $\theta^{*(i)}$ such that $\theta^* = (\theta^{*(1)},\theta^{*(2)},\cdots,\theta^{*(k)})$. In this setting, $f(\cdot)$ represents the trained DNN and f_i's represent network layers. We further define a κ-split $f = f_L^\kappa \circ f_R^\kappa$ where $f_L^\kappa(\cdot;\theta_L^{*\kappa}) = f_1 \circ \cdots \circ f_{*\kappa}$ and $f_R^\kappa(\cdot;\theta_R^{*\kappa}) = f_{\kappa+1} \circ \cdots \circ f_k$ represent local and remote parts of the network, respectively. We assume DNN is trained and the parameter θ^* remains the same throughout the process.

Privacy: We define the privacy P_κ of a κ-split as the negative of the mutual information between the output of the i^{th} layer and the input, i.e., $P_\kappa = -\mathcal{I}(\mathbf{x};f_L^\kappa(\mathbf{x}))$, where, $\mathcal{I}(\cdot;\cdot)$ represents mutual information. The lower mutual information implies higher privacy. Since f_L^κ is a deterministic function, we have



(a) In vivo privacy



(b) Accuracy

Figure 5. (a) In vivo notion of privacy and (b) accuracy per iteration of training on AlexNet, when cutting at the last convolution layer. The black lines show regular training with cross entropy loss function. The orange lines show Shredder's learning, with loss function shown in Equation 4.

$$P_\kappa = -\mathcal{H}(f_L^\kappa(\mathbf{x})) + \mathcal{H}\left(f_L^\kappa(\mathbf{x})\middle|\mathbf{x}\right) = -\mathcal{H}(f_L^\kappa(\mathbf{x})) \tag{6}$$

where $\mathcal{H}(\cdot)$ denotes the entropy function. Moreover, applying Data Processing Inequality, we have $P_1 \le P_2 \le \cdots \le P_k$, that intuitively means that as we pass the input through more layers, the privacy improves.

Noise Injection: Denote a perturbation vector by $\mathbf{w}_\kappa \in \mathbb{R}^{p_\kappa}$ which is independent of $\mathbf{x}$. We intend to perturb the input to f_R^κ by changing $\mathbf{y} = f_L^\kappa(\mathbf{x},\theta_L^\kappa)$ to $\hat{\mathbf{y}} = f_L^\kappa(\mathbf{x},\theta_L^\kappa) + \mathbf{w}_\kappa$, i.e., the output of function $f(\cdot)$ changes from $f(\mathbf{x};\theta^*) = f_R^\kappa(\mathbf{y};\theta_R^\kappa)$ to $\hat{f}(\mathbf{x};\theta) = f_R^\kappa(\mathbf{y}+\mathbf{w}_\kappa;\theta_R^\kappa)$. It is worth reemphasizing that the parameter θ^* remains unchanged. This results in a change to the privacy measure as $\hat{P}_\kappa = -\mathcal{I}(\mathbf{x};f_L^\kappa(\mathbf{x})+\mathbf{w}_\kappa)$. We can provide the following lower bound on the perturbed privacy:

$$\begin{aligned}\hat{P}_\kappa &\ge -\mathcal{I}\left(f_L^\kappa(\mathbf{x});f_L^\kappa(\mathbf{x})+\mathbf{w}_\kappa\right)\\ &= -\mathcal{H}(f_L^\kappa(\mathbf{x})) + \mathcal{H}\left(f_L^\kappa(\mathbf{x})\middle|f_L^\kappa(\mathbf{x})+\mathbf{w}_\kappa\right)\\ &= P_\kappa + \mathcal{H}\left(f_L^\kappa(\mathbf{x})\middle|f_L^\kappa(\mathbf{x})+\mathbf{w}_\kappa\right)\end{aligned} \tag{7}$$

where the last equality is derived from (6). This equation implies that by noise injection process, we can improve the privacy at least by $\mathcal{H}\left(f_L^\kappa(\mathbf{x})\middle|f_L^\kappa(\mathbf{x})+\mathbf{w}_\kappa\right)$.

Optimization Problem: We would like to inject a noise to maximize $\hat{P}_\kappa$ such that the accuracy does not degrade, i.e.,

$$\mathbf{w}_\kappa^* = \underset{\mathbf{w}_\kappa}{\operatorname{argmax}}\ \hat{P}_\kappa \quad \mathbf{s.t.}\ \ \mathcal{L}(\hat{f}) \leq \mathcal{L}(f)+\epsilon \tag{8}$$

where $\mathcal{L}$ is the neural network's loss function. Given (7), we use $\mathcal{H}\left(f_L^\kappa(\mathbf{x})\middle|f_L^\kappa(\mathbf{x})+\mathbf{w}\right)$ as surrogate objective and reformulate the problem in terms of a Lagrange multiplier λ as

$$\mathbf{w}_\kappa^* = \underset{\mathbf{w}_\kappa}{\operatorname{argmin}} -\mathcal{H}\left(f_L^\kappa(\mathbf{x})\middle|f_L^\kappa(\mathbf{x})+\mathbf{w}_\kappa\right)+\lambda\mathcal{L}\left(\mathbf{x},f_R^\kappa(f_L^\kappa(\mathbf{x})+\mathbf{w}_\kappa)\right) \tag{9}$$

Denoting $\mathbf{y}=f_L^\kappa(\mathbf{x})$ and applying the Bayes rule to the conditional entropy, we get $\mathcal{H}\left(\mathbf{y}\middle|\mathbf{y}+\mathbf{w}_\kappa\right)=\mathcal{H}\left(\mathbf{y}+\mathbf{w}_\kappa\middle|\mathbf{y}\right)+\mathcal{H}(\mathbf{y})-\mathcal{H}(\mathbf{y}+\mathbf{w}_\kappa)=\mathcal{H}(\mathbf{w}_\kappa)+\mathcal{H}(\mathbf{y})-\mathcal{H}(\mathbf{y}+\mathbf{w}_\kappa)$, where the last equality follows from the fact that $\mathbf{w}_\kappa$ is independent of $\mathbf{y}$. Since $\mathcal{H}(\mathbf{y})$ is constant with respect to $\mathbf{w}_\kappa$, rewrite (9) as

$$\mathbf{w}_\kappa^* = \underset{\mathbf{w}_\kappa}{\operatorname{argmin}}\mathcal{H}(\mathbf{y}+\mathbf{w}_\kappa)-\mathcal{H}(\mathbf{w}_\kappa)+\lambda\mathcal{L}\left(\mathbf{x},f_R^\kappa(\mathbf{y}+\mathbf{w}_\kappa)\right) \tag{10}$$

Optimization problem (10) has three terms. The term $\mathcal{H}(\mathbf{y}+\mathbf{w}_\kappa)$ controls the amount of information that leaks to the remote part of the DNN and we want to minimize this information. The term $\mathcal{H}(\mathbf{w}_\kappa)$ controls the amount of uncertainty that we are injecting in the form of iid noise. This term is typically proportional to the variance of the noise and we want this term to be maximized (and hence the negative sign). The last term controls the amount of degradation in the accuracy of the DNN and we want to minimize that.

The loss function $\mathcal{L}(\cdot)$ for a q-class classification problem with $\mathbf{z}=f(\mathbf{x};\theta^*)\in\mathbb{R}^q$ can be $\mathcal{L}(\mathbf{x},\mathbf{z})=-\sum_{j=1}^q \mathbf{1}_{\mathbf{x}\in C_j}\log(\mathbf{z}_j)$ where $\mathbf{1}_.$ is the indicator function, C_j represents the j^{th} class and $\mathbf{z}_j$ is the j^{th} entry of $\mathbf{z}$ representing the probability that $\mathbf{x}\in C_j$. Suppose $\mathbf{1}_\mathbf{x}$ is a one-hot-encoded q-vector with j^{th} element being 1 if $\mathbf{x}\in C_j$ and the rest are zero. We then can write the classification loss in vector form as $\mathcal{L}(\mathbf{x},\mathbf{z})=-\mathbf{1}_\mathbf{x}^T\log(\mathbf{z})$. For the remainder of this paper, we target a q-class classification problem.

Consider n iid observations of $\mathbf{x}_1,\cdots,\mathbf{x}_n$ where each entry of $\mathbf{y}_i=f_L^\kappa(\mathbf{x}_i)$ is independently distributed as Laplace distribution $L(\mu_y,b_y)$ and entries of $\mathbf{w}_\kappa$ are iid drawn from $L(0,b_w)$. For Laplace random variables we have $\mathcal{H}(\mathbf{y}+\mathbf{w}_\kappa)\propto\log(b_y+b_w)$ and $\mathcal{H}(\mathbf{w}_\kappa)\propto\log(b_w)$. Hence, we can rewrite (10) as an optimization problem on b_w as follows

$$b_w^* = \underset{b_w}{\operatorname{argmin}}\ \log(b_y+b_w)-\log(b_w)+\lambda_n\sum_{i=1}^n \mathbf{1}_\mathbf{x}^T\log\left(f_R^\kappa(\mathbf{y}_i+\mathbf{w}_\kappa)\right) \tag{11}$$

In order to solve this optimization problem, we start from an initial $\mathbf{w}_\kappa$ and compute the variance of that as b_w. We then take a gradient step on $\mathbf{w}_\kappa$ and then update b_w until we converge. This process gives us an optimal $\mathbf{w}_\kappa^*$ and an optimal b_w^*.

Table 1. Platforms used for obtaining the results in Section 6.2. We used a TitanXp GPU as our cloud server, and a Tegra X2 GPU (Jetson TX2) as our edge device.

	Cloud GPU	Edge GPU
Chip	**Titan Xp**	**Tegra X2**
Cores	3,584	256
ChipArea (mm^2)	471	-
TDP	250 W	7.5 W
Memory	12 GB	8 GB
Technology	16 nm	16 nm
Frequency	1,531 MHz	875 MHz

Table 2. Benchmark networks and datasets used for obtaining the results in Section 6.2. They are all real-world application networks, used for image classification. The last network/dataset is widely used for text classification in natural language processing.

Neural Network	No. of Conv layers	No. of FC layers	Pre-trained Accuracy	Suggested Partitioning layer		No. of Classes
				Edge-CPU	Edge-GPU	
LeNet	3	2	99.6%	3	3	10
CIFAR-10	2	3	72.3%	2	2	10
SVHN	7	1	92.5%	4	5	10
AlexNet	5	3	56.6%	2	5	1000
VGG-16 (primary)	13	3	77.80%	2	9	2
VGG-16 (private)	13	3	91.20%	2	9	24
20Newsgroups	3	2	96.5%	3	3	20

Inference: We first draw p_κ (same dimension as the dimension of $\mathbf{w}_\kappa^*$) samples from the Laplace distribution we learned in (11), i.e., $L(0,b_w^*)$. Then, we sort the samples and create a noise vector $\mathbf{w}_\kappa$ such that for all indices i and j, if the i^{th} element of $\mathbf{w}_\kappa^*$ is larger than its j^{th} element, then the same holds for $\mathbf{w}_\kappa$. This process ensures that while we generate random samples, the order of elements are always preserved in $\mathbf{w}_\kappa$.

Guarantee: Since in the process of optimizing (11), we computed the variance from the samples each time, the values of $\mathbf{w}_\kappa^*$ tend to be close to the ordered statistics of the Laplace distribution $L(0,b_w)$. This means that as long as we preserve the order of the values in the noise vector, we statistically stay at the optimal point. This justifies our inference method based on the optimal solution that we get during training.

6 Evaluation

6.1 Methodology

Benchmarked networks with no private labels. We used 6 real-world application networks as our benchmarks, which can be seen in Table 2. LeNet, CIFAR, SVHN, AlexNet, and VGG-16 are all image classification applications, and 20NewsGroups [40] is a widely used news topic classification application for natural language processing. For LeNet, the primary task is digit classification, for CIFAR it is object classification, for SVHN, number street view house number classification,

for Alexnet it is image classification into 1000 classes of ImageNet dataset. VGG-16, the task is gender classification, over a subset of the VGG-Face dataset which has images of celebrities [41]. For the 20Newsgroups, the task is classifying the input text which is extracted from news into 20 topics. Besides the aforementioned expected classifications labels, any other information about the image or text is considered private. For example, for images, what is in the background of the image, if it is day or night, or the identity of the celebrity in the image is considered private.

Benchmarked networks with private labels. For the sake of comparison with the related work [42], we adopt their private labels and primary labels for fairness. For the VGG-16 network, we used VGG-Face dataset [41], as seen in Table 2. The private labels for this dataset are the identity of the celebrities in it. The primary labels are gender classification.

Datasets for experimentation. Each dataset in Table 2 comes with a training dataset and a validation dataset. We use the training dataset for learning noise. During the process of learning, we use a 10% hold-out subset to assess the accuracy. However, we do not use any of the training datasets for the reported accuracy results. Instead, we use the validation dataset that is not seen during the learning phase for assessing the final accuracies.

Communication setup. We have used both private Wi-Fi and AT&T LTE for communication between edge and cloud.

Network partitioning points. Shredder can cut the network from any given layer and apply the noise distribution. However, As explained in Section 2, a cutting point that minimizes the overall communication and computation costs is chosen by Shredder. Our methodology for modeling the computation and communication costs is commensurate with [43]). The partitioning points used in the experiments can be seen in Table 2. These are the indices of convolution layers, and by cutting at the 3rd layer, we mean the noise is applied right before entering the 4th convolution, and from the 4th convolution to the end of the network, the execution would be on the cloud.

Edge device specifications. We used Nvidia Jetson TX2's off-the-shelf GPU board as our edge device with CUDA V10.0.166 alongside a quad-core ARM A57 that runs Ubuntu 18.04.2 LTS (GNU/Linux 4.9.140-tegra aarch64). The specifications of this GPU is in Figure 1. We used the latest PyTorch version 1.2 built from the source.

Cloud specifications. We used Nvidia Titan Xp off-the-shelf GPU with CUDA version 10.0.130 with 12GB of RAM alongside a 12 core Intel Corei9-7920X 2.90GHz CPU that runs Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-55-generic x86_64). We used the latest PyTorch version 1.2 installed with pip3. Table 1 has the specifications of the GPU.

End-to-end cost measurements. For reporting the cost (computation and communication) of Shredder, we have measured the end-to-end execution time of the DNN on the edge and cloud devices. To be more specific, we have measured the time it takes for the network's first section is executed on the Jetson TX2, then the activations are transferred over Wi-Fi to a Titan Xp server and the rest of network is executed there. We have executed 100 times and reported the average.

Baseline. We have selected cloud-only execution as our baseline since it is the widely used approach for edge devices [44, 45].

Offline noise distribution learning phase setup. For this, we use the same hardware and software setup as the cloud.

Privacy measurement setup. Mutual Information (MI) is calculated using the Information Theoretical Estimators Toolbox's [46] Shannon Mutual Information with KL Divergence. In the results reported in upcoming sub-sections, MI is calculated over the shuffled test sets on MNIST [47] dataset for LeNet [48], CIFAR-10 dataset for CIFAR-10 [49], SVHN dataset for SVHN [50], ImageNet [51] dataset for AlexNet [52], a subset of VGG-Face [41] for the VGG-16 dataset and the 20Newsgroups [40] dataset for 20Newsgroup's neural network. These photos were shuffled through and chosen at random. Using mutual information as a notion of privacy means that Shredder targets the average case privacy, but does not guarantee the amount of privacy that is offered to each individual user.

α and γ parameters. For the parameters (knobs) mentioned in Equations 4 and 5, We use -0.01, -0.001, and -0.0001 for α on {LeNet, CIFAR, 20Newsgroups}, {SVHN} and {AlexNet, VGG-16} respectively. For γ, we use 0.01 for VGG-16, since it is the only network with defined private labels.

Optimizer setup for offline noise distribution learning. We used the cross-entropy loss function with Adam Optimizer [39], with a learning rate of 0.01 on LeNet, CIFAR and 20Newsgroups, 0.001 on SVHN and 0.0001 on AlexNet and VGG-16.

Comparison with DPFE setup. We compare Shredder with DPFE[42] over VGG-16 network, for face identification and gender classification on celebrity faces, which is the exact setup used by [42] to evaluate their method. DPFE offers only this benchmark, and we have used the same network partitioning point as well, to provide fairness. We have used celebrity faces from the VGG-Face dataset and partitioned it to validation/training portions to train and verify the classifiers.

6.2 Experimental Results

This section elaborates on our observations in detail and brings empirical evidence for the efficacy of Shredder. We discuss the memory footprint and latency overheads of Shredder, accuracy-privacy trade-off, a comparison with another

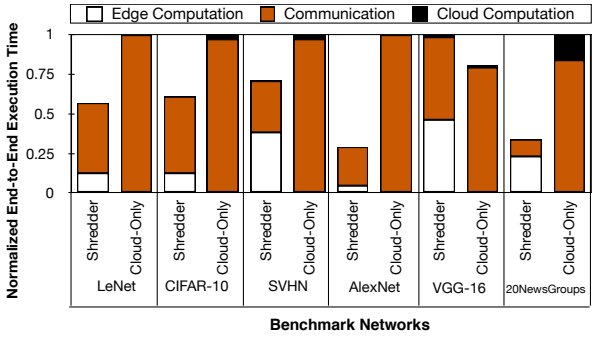


Figure 6. Comparison of Shredder's end-to-end execution time, compared to a baseline of cloud-only execution which is widely used. In this evaluation the partitioning points are those mentioned in Table 2 and Wi-Fi is used for communication.

method, and finally, a network partitioning point privacy and edge computation trade-off analysis. For all the experiments, except the ones in Section 6.2.4, the loss function form of Equation 4 has been used since no assumption on the private labels is given, which is the case in most privacy-related problems. In Section 6.2.4, we assume private labels (Equation 5) and compare Shredder with and without the private labels against related work, DPFE [42].

Table 3 summarizes our experimental results. We have indicated the margin of error for each benchmark here. Since the margin is trivial, we have only brought it in the table, and not in the plots, for the sake of simplicity. It is shown that on the networks, Shredder can achieve on average 74.70% loss in information while inducing 1.58% loss in accuracy. The table also shows that it takes Shredder a short time to train the noise tensor, for instance on AlexNet on ImageNet and 1000 classes, it is 0.1 epoch. It is also evident that the memory overhead of Shredder due to its distribution collector is insignificant.

6.2.1 Overheads of Shredder. As Table 3 shows, Shredder alleviates the complexities of training an entire neural network, by decreasing the number trainable parameters of the network to 0.16%. The collection of distributions and element orders imposes a trivial memory footprint of saving the distribution parameters (two numbers per distribution, which are the *location* and *scale*, are similar to mean and standard deviation) and the orders of tensor elements for each distribution, which is a string of numbers. The length of this string is the same as the size of the noise tensor, and each element is smaller than or equal to the length of the string, since the elements are the sorted indices of the noise tensor, as explained in Section 4.5. There are 20 distributions collected for each benchmark. The average of all these overheads combined is 19.35 KBs.

Figure **??** shows the end-to-end execution time of Shredder, normalized to a baseline of cloud-only execution. The Figure shows that Shredder outperforms the baseline for all benchmarks, except VGG-16 and it has an average speedup of 1.79×. These results are all commensurate with Neurosergeon and show the same trends [43]. This speedup is mainly because of the high communication costs and the large size of the input to the network that should be transmitted to cloud for cloud-only execution, whereas Shredder sends a much smaller intermediate layer. It is important to keep in mind that the computation overhead of the distribution sampler and noise adder is on average 0.03× the communication time of Shredder, and is therefore trivial in comparison to the communication costs.

The reason for VGG-16's slow-down is that for this network, over Wi-Fi, the optimal cutting point is the input layer, and since Shredder does not partition network from input layer to provide more privacy, it partitions from the convolution 13th layer which is the second optimal computation and communication point, and at this point, the activation size is still relatively big, so, it does not help compensate the excess computation time imposed to the edge device by the execution of the previous 13. Therefore, the overall execution time is increased by 23%, in comparison to the cloud-only approach.

As mentioned, communication is measured over Wi-Fi, which is faster than LTE and 3G, so Shredder's speedup would be even higher if one of the slower communication methods were employed. For instance, if LTE is used, the overall speedup would be 2.17× and VGG-16 would have a speedup of 1.03×.

6.2.2 Accuracy-Privacy Trade-Off. There is a trade-off between the amount of noise that we incur to the network and its accuracy. As shown in Figure 1, Shredder attempts to increase privacy while keeping the accuracy intact. Figure 7 shows the level of privacy that can be obtained by losing a given amount of accuracy for LeNet, CIFAR-10, SVHN, and AlexNet. In this Figure, the number of mutual information bits that are lost from the original activation using our method is shown on the Y axis. The cutting point of the networks is their last convolution layer. This can be perceived as the output of the *features* section of the network, if we divide the network into *features* and *classifier* sections.

The *Zero Leakage* line depicts the amount of information that needs to be lost to leak no information at all. In other words, this line points to the original number of mutual information bits in the activation that is sent to the cloud, without applying noise. The black dots show the information loss that Shredder provides, given a certain loss in accuracy. These trends are similar to that of Figure 1, since Shredder tries to strip the activation from its excess information, thereby preserving privacy and only keeping the information that is used for the classification task. This is the sharp (high slope) rise in information loss, seen in sub-figures of Figure 7. Once the excess information is gone, what remains is mostly what is needed for inference. That is why there is a point (the low slope horizontal line in the figures) where adding more noise

Table 3. Summary of the experimental results of Shredder for the benchmark networks.

Benchmark	LeNet	CIFAR	SVHN	Alexnet	VGG-16	20Newsgroups	Average
Original Mutual Information	301.84	236.34	19.2	12661.51	28732.21	27.8	–
Shredded Mutual Information	18.9	90.2	7.1	4439.0	7268.7	7.8	–
Mutual Information Loss	**93.74%**	**61.83%**	**64.58%**	**64.94%**	**74.70%**	**72.95%**	**74.70%**
Accuracy Loss	**1.34%**	**1.42%**	**1.12%**	**1.95%**	**1.68%**	**1.99%**	**1.58%**
Margin of Error	±0.39%	±0.87%	±0.22%	±0.12%	±0.11%	±0.41%	±0.35%
Shredder's Learnable Params over DPFE[42]	0.19%	0.65%	0.04%	0.02%	0.01%	0.05%	0.16%
Number of Epochs of Training	6.3	1.7	1.2	0.1	6.8	7.3	1.99
Shredder's Distribution Collector Memory Footprint (KB)	2.05	8.70	5.00	315.00	918.75	2.01	19.35

(losing more information bits) causes a huge loss in accuracy. The extreme to this case can be seen in 7a, where approaching the *Zero Leakage* line causes about 20% loss in accuracy.

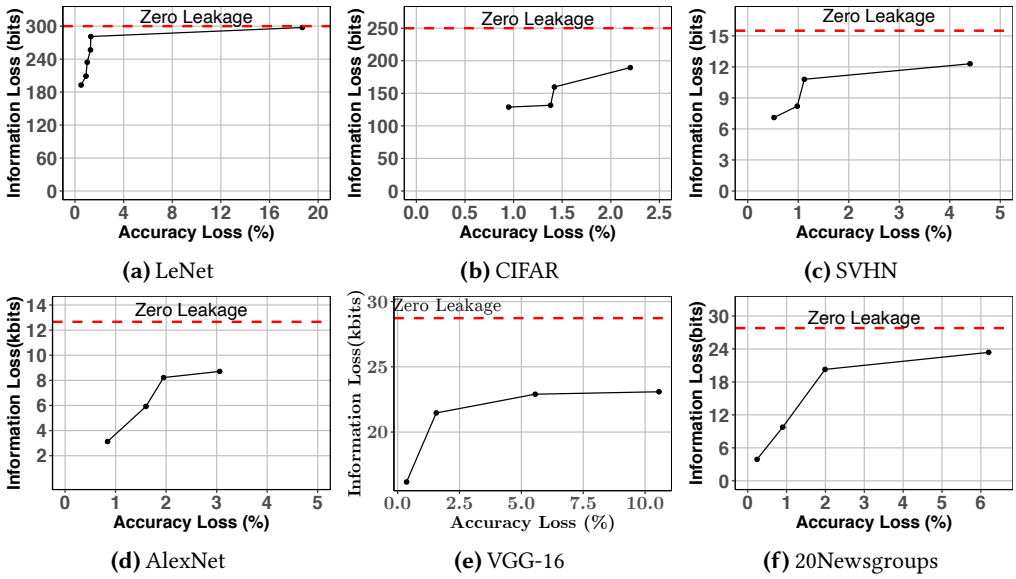
6.2.3 Partitioning Point Trade-offs. Layer selection for network cutting point is mostly an interplay of communication and computation costs of the edge device. As discussed in Section 2, the network partitioner chooses the layer with the lowest end-to-end execution cost. The deeper this layer is, the higher privacy yield for a given accuracy [42], since the network operations like pooling layers, ReLU and convolutions themselves modify the input information and give the Shredder framework a lower mutual information to begin with [25, 26]. That's why the partitioner it set to never choose the input layer as partitioning point (which is what cloud-only execution is similar to) since it compromises privacy. However, if the user wants even higher privacy, they could cut the network deeper and get higher privacy, at the cost of more edge computation. But the relation is not linear, and the returns for privacy start diminishing at some point. Figure 8 shows the highest normalized privacy that can be reached with less than 5% loss in accuracy for different edge device computation time (different partitioning layers) over AlexNet and VGG-16. The numbers on the x axis show the computation time and each point on the plot shows a layer. For AlexNet, after the 3rd convolution layer, and for VGG-16, after the 8th one, the increase in privacy is insignificant as we move forward through the convolution layers. The trend is similar for other networks, for the sake of space, we have chosen VGG-16 and Alexnet as representative of other DNNs [53]. The users can weigh the trade-offs, and **sacrifice costs for more privacy** and cut deeper layers. However, as we show in the experiments, the privacy plateaus at some point and is not ever-increasing. The suggested partitioning points can be seen in Table 2

6.2.4 Comparison with DPFE. Deep Private Feature Extraction (DPFE) [42] is a privacy protection mechanism that aims at obfuscating given private labels, by modifying the network topology and re-training all the model parameters. DPFE partitions the network in two partitions, first partition to be deployed on the edge and the second on the cloud. It also modifies the network architecture by adding an auto-encoder in the middle and then re-training the entire network with its loss function. DPFE's loss function is composed of three terms, first, the cross-entropy loss which aims at maintaining accuracy, second, a term that tries to decrease the distance between intermediate activations with different private labels, and a final term which tries to increase the distance between intermediate activations of inputs with the same private label. After training, for each inference, a randomly generated noise is added to the intermediate results on the fly.

DPFE can only be effective if the user knows what s/he wants to protect against, whereas Shredder offers a repetitively more general approach that tries to obliterate any information that is irrelevant to the primary task. Table 3 has a row that compares the number of trainable parameters for Shredder with DPFE and it can be seen that Shredder's parameters are extremely lower than DPFE's.

To run experiments, the intermediate outputs of the networks are fed to two classifiers, for gender and identity, each of which display an original (before adding noise) accuracy of 91.56% and 77.8%, respectively. Then DPFE and Shredder are applied to the neural networks, and the accuracy results over validation sets are seen in Figure 9.

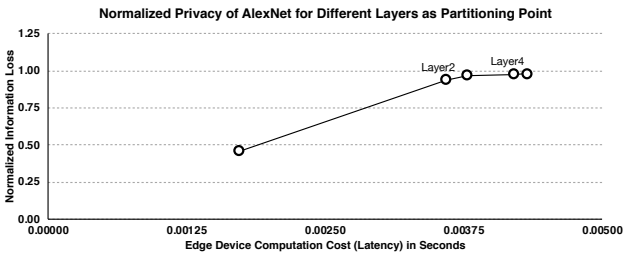
Figure 9a compares DPFE to Shredder with its Equation 4 (without private labels) and Equation 5 (with private labels) in terms of the misclassification rate of the private task, i.e. the identity classification (identity compromise) for given levels of accuracy for the main task, which is the same metric used in [42]. As expected, Shredder without private labels has the lowest misclassification rate, which is due to not having any knowledge of the private data. At higher levels of primary accuracy, Shredder with private labels outperforms DPFE. This can be attributed to Shredder's loss function, especially the last term, which aims at maximizing the amount of noise while keeping the accuracy intact. This could be the reason why Shredder has a higher misclassification rate in a more constrained setting (higher primary accuracy). Whereas DPFE adds the noise on the fly, which can only help increase the misclassification rate to a certain degree. In lower accuracy levels where the state-space is less constrained, DPFE offers higher



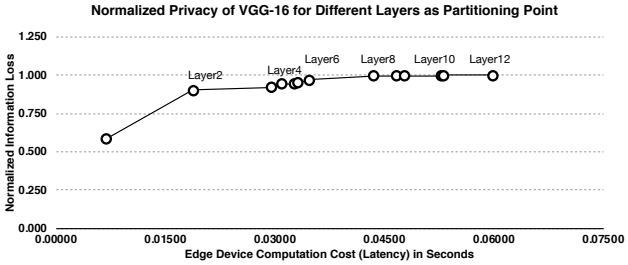
(a) LeNet **(b)** CIFAR **(c)** SVHN

(d) AlexNet **(e)** VGG-16 **(f)** 20Newsgroups

Figure 7. Accuracy-Privacy trade-off in 6 benchmark networks. The zero leakage line shows the original mutual information between input images and activations at the cutting point.



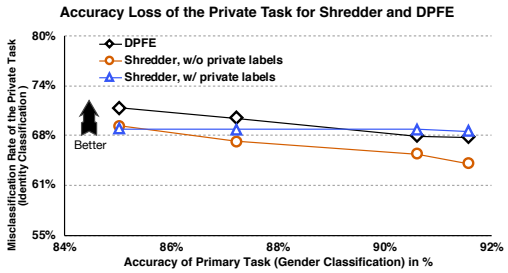
(a) Normalized privacy over last layer



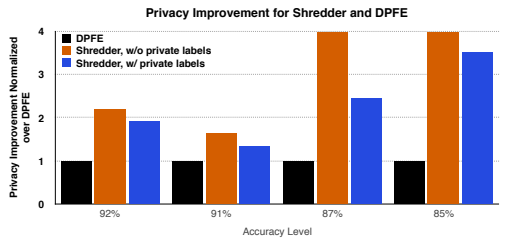
(b) Normalized privacy over last layer

Figure 8. Normalized privacy over last layer for different levels of computation on the edge device (different partitioning points) for (a) AlexNet on ImageNet and (b) VGG-16 on VGG-Face dataset. After a point (3rd convolution layer for AlexNet and 8th convolution layer for VGG-16) the improvements in privacy start to diminish.

misclassification, which can be attributed to it's a higher number of trainable parameters which can give it more wiggle room.



(a) Misclasification rate of private labels



(b) Normalized privacy improvement over DPFE

Figure 9. Misclasification rate of private labels (identity) and privacy improvement comparison for different accuracy levels of the primary task (gender classification) over VGG-16 on VGG-Face dataset, for Shredder with both loss functions and DPFE.

Figure 9b shows the privacy improvement of Shredder with both loss functions, over DPFE. It can be inferred from the Figure that Shredder without the private labels performs better since it has a more general approach which scrambles more overall information. However, Shredder with the private labels and also DPFE, take an approach which is directed at a specific goal, which impedes them from providing privacy

for aspects other than the private task. This is seen more in DPFE than Shredder with private labels since the latter still tries to maximize noise standard deviation.

7 Related Work

The literature abounds with a variety of attempts to provide greater protection to users' private data in a neural processing system [19, 21, 54–56]. These efforts span different levels of the system, from training to inference. The majority of these studies [55, 56]; however, have focused on preserving the privacy of contributing users to statistical databases or training models. [57], for instance, introduces a privacy-preserving protocol for federated learning. These techniques tackle the inherent conflict of extracting useful information from a database while protecting private or sensitive data of the individuals from being extracted or leaked [58]. As Table 4 illustrates, the landscape of research in privacy for neural networks can be categorized into the efforts that focus on training or inference. These categories can be further grouped according to whether or not they require retraining the DNN weights or modifying the model itself (i.e., intrusive). Shredder falls in the category of the techniques that are non-intrusive and target the inference phase.

The other technique in this same category, MiniONN [21], uses homomorphic encryption that imposes non-trivial computation overheads making it less suitable for inference on edge. Below, we discuss the most related works, which typically require obtrusive changes to the model, training, or add prohibitively large computation overheads.

Adding noise for privacy. The idea of noise injection for privacy goes back at least to the very first differential privacy papers [23, 59] where they randomize the result of a query to a database by adding noise drawn from a Laplace distribution. More recently, Wang et al. [60] proposes data nullification and noise injection for private inference.

However, unlike Shredder, they retrain the network. Osia et al. [42, 61] design a private feature extraction architecture that uses principal component analysis (PCA) to reduce the amount of information. Leroux et al. [62] use an autoencoder to obfuscate the data before sending it to the cloud, but the obfuscation they use is readily reversible, as they state. We, on the other hand, cast finding the noise as differentiable noise tensor while considering accuracy in the loss function of the optimization that finds the noise.

Trusted execution environments. Several research propose running machine learning algorithms in in trusted execution environments such as Intel SGX [63] and ARM TrustZone [64] to address the same remote inference privacy [65–68] as well as integrity [65]. However, the privacy model in that research requires users to send their data to an enclave running on a remote servers. In contrast to Shredder, this model still allows the remote server to have access to the raw data and as the new breaches in hardware [8, 9, 69–71] show, the access can lead to comprised privacy.

Differential privacy. As a mathematical framework, differential privacy [23, 58, 59] was initially proposed to quantify privacy of users in the context of privacy-preserving data-mining or statistical databases. To this end, it measures the degree to which the algorithm behaves similarly if an individual record is in or out of the database/training set. This definition gives a robust mathematical guarantee to the question of – given a private training set (or, database entry) as input, how safe is the trained model (or, aggregate database) to publish [72]. Naturally, differential privacy has also been employed in training of deep neural networks [55, 56] where the datasets may be crowdsourced and contain sensitive information. The research on differential privacy is largely in *centralized models*, where users trust a curator who has access to the whole pool of private data [58]. In a more practical model, called local differential privacy, the system does not require users to even trust the curator to inspect their data, even for the purpose of preserving privacy [14, 15, 73, 74]. In this setting, which the system is just collecting data and not performing inference, the data is still scrambled on the edge devices. This scrambled data is then remotely aggregated and just provides an average trend across multiple sources. The existing differential privacy models are in fact solving a fundamentally different problem than Shredder. They are concerned with data collection while Shredder aims to improve privacy during a real-time cloud-enabled inference.

Encryption and cryptographic techniques. Secure multiparty computation (SMC) [16, 17] and homomorphic encryption [19, 21, 75] have also been used as attempts to deal with the privacy on offloaded computation on the cloud [19, 21, 22, 75, 76, 76, 77]. Secure multiparty computation refers to a group of protocols that enable multiple parties to jointly compute a function while each party solely has access to its own part of the input [17, 77]. To establish trust and isolation, SMC relies on compute-heavy encryption or obfuscation techniques. To adopt SMC to the privacy problem, recent works [77] assume a two-party secure computation in which the cloud holds a neural network and the client holds an input to the network, typically an image and the communication happens in the encrypted domain. Homomorphic encryption, which can be used to implement SMC, is also used for privacy protection in neural networks. This cryptographic technique allows (all or a subset of) operations to be performed on the encrypted data without the need for decryption. These works [19, 21] suggest the client/edge device encrypts the data (on top of the communication encryption, e.g., SSL) before sending it to the cloud; which it then, performs operations on the encrypted data and returns the output. Nevertheless, this approach suffers from a prohibitive computational and communication cost, exacerbating the complexity and compute-intensivity of neural networks especially on resource-constrained edge

Table 4. Privacy Protecting methods in DNNs.

	Non-Intrusive	Intrusive
Inference	***Shredder*** MiniONN [21]	CryptoNets [19], GAZELLE [75] Arden [60], DPFE [42, 61]
Training/DB	Rappor [15] Apple [14]	With Differential Privacy [55, 56], SecureML [77]

devices. Shredder, in contrast, avoids the significant cost of encryption or homomorphic data processing.

8 Conclusion

Privacy is a fundamental human right recognized in the United Nations (UN) Declaration of Human Rights. However, the systems and computational infrastructure in use seem to have been designed for offering functionality without foundational consideration for privacy. Such a gap is more concerning today since cloud-based deep learning service make their way to the households as home automation devices or shape our social, political, and economical interactions. A single paper is not an answer to this brewing epidemic, but it represents an initial effort to build a mathematically-sound private systems that offer reliable degrees of utility. As such, this paper examines the use of noise to reduce the information content of the communicated data to the cloud while still maintaining high levels of accuracy. By casting the noise injection as a learning process that uses differentiation to find the distribution of the noise, we devise Shredder, which strikes an asymmetric balance between accuracy and privacy with formal mathematical guarantees. Experimentation with multiple real-life DNNs showed that Shredder can significantly reduce the information content of the communicated data with only 1.58% accuracy loss. These results pave a promising path forward.

9 Acknowledgment

We thank the anonymous reviewers for their insightful comments. This work was in part supported by National Science Foundation (NSF) awards CNS#1703812, ECCS#1609823, CCF#1553192, Air Force Office of Scientific Research (AFOSR) Young Investigator Program (YIP) award #FA9550-17-1-0274, National Institute of Health (NIH) award #R01EB028350, and Air Force Research Laboratory (AFRL) and Defense Advanced Research Project Agency (DARPA) under agreement number #FA8650-20-2-7009 and #HR0011-18-C-0020. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Arm, Amazon, NSF, AFSOR, NIH, AFRL, DARPA or the U.S. Government.

References

[1] M. A. Cusumano, "Cloud computing and saas as new computing platforms.," *Commun. ACM*, vol. 53, no. 4, pp. 27–29, 2010.
[2] C. Cachin, I. Keidar, A. Shraer, *et al.*, "Trusting the cloud," *Acm Sigact News*, vol. 40, no. 2, pp. 81–86, 2009.
[3] H. R. Motahari-Nezhad, B. Stephenson, and S. Singhal, "Outsourcing business to cloud computing services: Opportunities and challenges," *IEEE Internet Computing*, vol. 10, no. 4, pp. 1–17, 2009.
[4] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.
[5] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.
[6] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Toward dark silicon in servers," *IEEE Micro*, vol. 31, pp. 6–15, 2011.
[7] J. Hauswald, M. Laurenzano, Y. Zhang, C. Li, A. Rovinski, A. Khurana, R. G. Dreslinski, T. N. Mudge, V. Petrucci, L. Tang, and J. Mars, "Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers," in *ASPLOS*, 2015.
[8] P. Kocher, J. Horn, A. Fogh, , D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.
[9] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.
[10] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, (New York, NY, USA), pp. 199–212, ACM, 2009.
[11] NBCNews), "Facebook data harvesting scandal widens to 87 million people." online accessed May 2019 https://www.nbcnews.com/tech/tech-news/facebook-data-harvesting-scandal-widens-87-million-people-n862771.
[12] Axonium), "23andme scandal highlights data privacy concerns shared by axonium and mr koh boon hwee.." online accessed May 2019 https://medium.com/@Axonium_org/23andme-scandal-highlights-data-privacy-concerns-shared-by-axonium-and-mr-koh-boon-hwee-dd2e241f1ef2.
[13] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.
[14] Differential Privacy Team, "Learning with privacy at scale," tech. rep., Apple, 2017. online accessed May 2019 https://machinelearning.apple.com/docs/learning-with-privacy-at-scale/appledifferentialprivacysystem.pdf.
[15] Úlfar Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Proceedings of the 21st ACM Conference on Computer and Communications Security*, (Scottsdale, Arizona), 2014.
[16] A. C. Yao, "How to generate and exchange secrets," in *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pp. 162–167, Oct 1986.
[17] R. Bahmani, M. Barbosa, F. Brasser, B. Portela, A.-R. Sadeghi, G. Scerri, and B. Warinschi, "Secure multiparty computation from sgx," *IACR Cryptology ePrint Archive*, vol. 2016, p. 1057, 2016.
[18] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *In Proc. STOC*, pp. 169–178, 2009.
[19] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 201–210, JMLR.org, 2016.

[20] J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig, "Improved security for a ring-based fully homomorphic encryption scheme," in *Proceedings of the 14th IMA International Conference on Cryptography and Coding - Volume 8308*, IMACC 2013, (Berlin, Heidelberg), pp. 45–64, Springer-Verlag, 2013.

[21] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, (New York, NY, USA), pp. 619–631, ACM, 2017.

[22] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "Xonn: Xnor-based oblivious deep neural network inference." Cryptology ePrint Archive, Report 2019/171, 2019. https://eprint.iacr.org/2019/171.

[23] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, (Berlin, Heidelberg), pp. 265–284, Springer-Verlag, 2006.

[24] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[25] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *NIPS*, 2014.

[26] R. Shwartz-Ziv and N. Tishby, "Opening the black box of deep neural networks via information," 2017.

[27] Z. Goldfeld, E. van den Berg, K. Greenewald, I. Melnyk, N. Nguyen, B. Kingsbury, and Y. Polyanskiy, "Estimating information flow in neural networks," 2018.

[28] A. M. Saxe, Y. Bansal, J. Dapello, M. Advani, A. Kolchinsky, B. D. Tracey, and D. D. Cox, "On the information bottleneck theory of deep learning," in *International Conference on Learning Representations*, 2018.

[29] N. Tishby, F. C. Pereira, and W. Bialek, "The information bottleneck method," *arXiv preprint physics/0004057*, 2000.

[30] N. Tishby and N. Zaslavsky, "Deep learning and the information bottleneck principle," *2015 IEEE Information Theory Workshop (ITW)*, Apr 2015.

[31] W. Wang, L. Ying, and J. Zhang, "On the relation between identifiability, differential privacy, and mutual-information privacy," *IEEE Transactions on Information Theory*, vol. 62, pp. 5018–5029, Sep. 2016.

[32] J. Liao, L. Sankar, V. Y. F. Tan, and F. du Pin Calmon, "Hypothesis testing under mutual information privacy constraints in the high privacy regime," *IEEE Transactions on Information Forensics and Security*, vol. 13, pp. 1058–1071, April 2018.

[33] L. Sankar, S. R. Rajagopalan, and H. V. Poor, "Utility-privacy tradeoffs in databases: An information-theoretic approach," *IEEE Transactions on Information Forensics and Security*, vol. 8, pp. 838–852, June 2013.

[34] D. Guo, S. Shamai, and S. Verdú, "Additive non-gaussian noise channels: Mutual information and conditional mean estimation," in *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.*, pp. 719–723, IEEE, 2005.

[35] Dongning Guo, S. Shamai, and S. Verdu, "Mutual information and minimum mean-square error in gaussian channels," *IEEE Transactions on Information Theory*, vol. 51, pp. 1261–1282, April 2005.

[36] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2016.

[37] O. Temam, "A defect-tolerant accelerator for emerging high-performance applications," *SIGARCH Comput. Archit. News*, vol. 40, pp. 356–367, June 2012.

[38] T. van Laarhoven, "L2 regularization versus batch and weight normalization," *CoRR*, vol. abs/1706.05350, 2017.

[39] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[40] "The 20 newsgroups data set." online accessed July 2019 http://qwone.com/~jason/20Newsgroups/.

[41] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *British Machine Vision Conference*, 2015.

[42] S. A. Osia, A. Taheri, A. S. Shamsabadi, M. Katevas, H. Haddadi, and H. R. R. Rabiee, "Deep private-feature extraction," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2018.

[43] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *SIGARCH Comput. Archit. News*, vol. 45, pp. 615–629, Apr. 2017.

[44] "Apple moves to third-generation siri back-end, built on opensource mesos platform." Online accessed August 2019, https://9to5mac.com/2015/04/27/siri-backend-mesos/.

[45] "Apple moves to third-generation siri back-end, built on opensource mesos platform." Online accessed August 2019, https://9to5mac.com/2015/04/27/siri-backend-mesos/.

[46] Z. Szabó, "Information theoretical estimators toolbox," *Journal of Machine Learning Research*, vol. 15, pp. 283–287, 2014.

[47] N. Yann LeCun (Courant Institute and N. Y. Corinna Cortes (Google Labs, "The mnist dataset of handwritten digits." online accessed May 2019 http://www.pymvpa.org/datadb/mnist.html.

[48] Y. LeCun, "Gradient-based learning applied to document recognition," 1998.

[49] A. Krizhevsky, "Convolutional deep belief networks on cifar-10," 2010.

[50] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. D. Shet, "Multi-digit number recognition from street view imagery using deep convolutional neural networks," *CoRR*, vol. abs/1312.6082, 2014.

[51] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[52] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, pp. 84–90, 2012.

[53] A. Coates, B. Huval, T. Wang, D. J. Wu, B. Catanzaro, and A. Y. Ng, "Deep learning with cots hpc systems," in *ICML*, 2013.

[54] Q. Xiao, K. Li, D. Zhang, and W. Xu, "Security risks in deep learning implementations," *CoRR*, vol. abs/1711.11008, 2017.

[55] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 909–910, Sep. 2015.

[56] M. Abadi, A. Chu, I. J. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," 2016.

[57] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, (New York, NY, USA), pp. 1175–1191, ACM, 2017.

[58] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, pp. 211–407, Aug. 2014.

[59] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, "Our data, ourselves: Privacy via distributed noise generation," in *Proceedings of the 24th Annual International Conference on The Theory and Applications of Cryptographic Techniques*, EUROCRYPT'06, (Berlin, Heidelberg), pp. 486–503, Springer-Verlag, 2006.

[60] J. Wang, J. Zhang, W. Bao, X. Zhu, B. Cao, and P. S. Yu, "Not just privacy: Improving performance of private deep learning in mobile cloud," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, (New York, NY, USA), pp. 2407–2416, ACM, 2018.

[61] S. A. Osia, A. S. Shamsabadi, S. Sajadmanesh, A. Taheri, K. Katevas, H. R. Rabiee, N. D. Lane, and H. Haddadi, "A hybrid deep learning architecture for privacy-preserving mobile analytics," 2017.

[62] S. Leroux, T. Verbelen, P. Simoens, and B. Dhoedt, "Privacy aware offloading of deep neural networks," *CoRR*, vol. abs/1805.12024, 2018.

[63] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and

software model for isolated execution," in *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, (New York, NY, USA), pp. 10:1–10:1, ACM, 2013.

[64] T. Alves and D. Felton, "Trustzone: Integrated hardware and software security," 01 2004.

[65] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," in *International Conference on Learning Representations*, 2019.

[66] T. Hunt, C. Song, R. Shokri, V. Shmatikov, and E. Witchel, "Chiron: Privacy-preserving machine learning as a service," 2018.

[67] L. Hanzlik, Y. Zhang, K. Grosse, A. Salem, M. Augustin, M. Backes, and M. Fritz, "Mlcapsule: Guarded offline deployment of machine learning as a service," 2018.

[68] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors," in *25th USENIX Security Symposium (USENIX Security 16)*, (Austin, TX), pp. 619–636, USENIX Association, 2016.

[69] M. Schwarz, M. Lipp, D. Moghimi, J. Van Bulck, J. Stecklina, T. Prescher, and D. Gruss, "ZombieLoad: Cross-privilege-boundary data sampling," *arXiv:1905.05726*, 2019.

[70] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *Proceedings of the 27th USENIX Security Symposium*, USENIX Association, August 2018.

[71] O. Weisse, J. Van Bulck, M. Minkin, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, R. Strackx, T. F. Wenisch, and Y. Yarom, "Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution," *Technical report*, 2018.

[72] J. F. Reed, A. J. Aviv, D. Wagner, A. Haeberlen, B. C. Pierce, and J. M. Smith, "Differential privacy for collaborative security," in *EUROSEC*, 2010.

[73] A. Bittau, U. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld, "Prochlo: Strong privacy for analytics in the crowd," in *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, (New York, NY, USA), pp. 441–459, ACM, 2017.

[74] B. Ding, J. Kulkarni, and S. Yekhanin, "Collecting telemetry data privately," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, (USA), pp. 3574–3583, Curran Associates Inc., 2017.

[75] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "Gazelle: A low latency framework for secure neural network inference," in *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC'18, (Berkeley, CA, USA), pp. 1651–1668, USENIX Association, 2018.

[76] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network," *IACR Cryptology ePrint Archive*, vol. 2017, p. 35, 2017.

[77] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 19–38, May 2017.