GECKO: Reconciling Privacy, Accuracy and Efficiency in Embedded Deep Learning

Vasisht Duddu, Antoine Boutet, Virat Shejwalkar

GECKO: Reconciling Privacy, Accuracy and Efficiency in Embedded Deep Learning

Vasisht Duddu
Univ Lyon, INSA Lyon, Inria, CITI
vduddu@tutamail.com

Antoine Boutet
Univ Lyon, INSA Lyon, Inria, CITI
antoine.boutet@insa-lyon.fr

Virat Shejwalkar
Univ Massachusetts Amherst
vshejwalkar@cs.umass.edu

ABSTRACT

Embedded systems demand on-device processing of data using Neural Networks (NNs) while conforming to the memory, power and computation constraints, leading to an efficiency and accuracy tradeoff. To bring NNs to edge devices, several optimizations such as model compression through pruning, quantization, and off-the-shelf architectures with efficient design have been extensively adopted. These algorithms when deployed to real world sensitive applications, requires to resist inference attacks to protect privacy of users training data. However, resistance against inference attacks is not accounted for designing NN models for IoT. In this work, we analyse the three-dimensional privacy-accuracy-efficiency tradeoff in NNs for IoT devices and propose Gecko training methodology where we explicitly add resistance to private inferences as a design objective. We optimize the inference-time memory, computation, and power constraints of embedded devices as a criterion for designing NN architecture while also preserving privacy. We choose quantization as design choice for highly efficient and private models. This choice is driven by the observation that compressed models leak more information compared to baseline models while off-the-shelf efficient architectures indicate poor efficiency and privacy tradeoff. We show that models trained using Gecko methodology are comparable to prior defences against black-box membership attacks in terms of accuracy and privacy while providing efficiency.

KEYWORDS

Membership Privacy, Inference Attacks, Efficient Deep Learning, Embedded Computing.

1 INTRODUCTION

The tremendous performance of Machine Learning, especially Deep Learning, has resulted in their deployment to low-powered edge devices and embedded systems. Specifically, Internet of Things (IoT) devices extensively prefer on-device processing to reduce communication latency and overhead, while also preserving the privacy of data from an untrusted data curator [29]. The design of efficient Neural Networks (NNs) requires algorithm-hardware co-design such as model compression, quantization, and designing special architectures with higher efficiency [30]. Such NN architecture design optimizations should conform to efficiency constraints on memory, energy, and computation overhead on embedded devices, and also maintain high prediction accuracy. However, such designs often result in *efficiency-accuracy* trade-off [20].

Additionally, privacy laws, such as HIPAA and GDPR, require on-device processing to maintain the privacy of user's sensitive data (e.g, medical records, location traces, and purchase preferences). In this work we focus on Membership Inference Attack [26], where given a target model and a target record, the adversary determines if the target data record was part of the target model's training data by analyzing the target model's output predictions. For instance, wearable devices, which monitor its user's health, commonly rely on NNs for various health related predictions. Such devices are continuously trained on the private data of a large number of users, and therefore, by mounting membership inference attacks on target device, an adversary can determine if the data of a target user was used to train NNs on the target device. In such cases, it is crucial to design NNs resistant to inference attacks, where the adversary infers unobservable, sensitive information (e.g, user's health status) from the observable information (e.g., model predictions). We refer to the computations that achieve privacy through inference-resistance as the *privacy-preserving computation*. Such privacy preserving computation mechanisms affect the model's predictive accuracy resulting in *privacy-accuracy* trade-off [2, 18, 24].

Considering the trade-offs described above, the three objectives to consider while designing NNs for embedded devices are: (a) high prediction accuracy, (b) efficiency constraints on memory, energy, and computation overhead, and (c) preserving privacy of on-device data. However, designing a model to preserve privacy while satisfying efficiency requirements without a significant cost of the models predictive accuracy is challenging. In this paper, we address this challenge by proposing Gecko a two phase training methodology for designing NNs optimized specifically for performance, accuracy and privacy. We evaluate the privacy leakage of three state of the art hardware software co-design techniques, i.e., namely, model compression, quantization and efficient off-the shelf architectures. We show that model compression (i.e., pruning the network) leaks more information compared to baseline (uncompressed) models indicating a higher privacy risk to the users data while off-the-shelf architectures (MobileNet and SqueezeNet) do not meet all the efficiency requirements but can provide limited privacy leakage. These observations motivate our design choice of quantizing NNs (i.e., reducing the number of bits that represent a number) as part of Gecko training algorithm.

In Phase-I of Gecko, the model parameters and activations are binarized, i.e., constrained to {-1,+1} to reduce the memory, energy consumption, and computation overhead. In addition, to ensure computation efficiency, we replace the

expensive *multiply accumulate* operations between parameter matrices and activation vectors to simple and cheaper XNOR operations. We show that Phase-I of GECKO optimizes the trained model for efficiency *and* privacy but at the cost of a significant drop in accuracy. In Phase-II, we restore this accuracy by transferring knowledge from larger full precision models to the quantized models [10]. Here, the quantized XNOR model uses the output predictions of the full precision state of the art models as labels instead of using the true labels during training. This significantly increases the prediction accuracy of the model while limiting privacy leakage. We show that aggressively quantized NN architectures obtained in Phase-I ensure an efficient and accurate privacy-preserving computation with higher resistance to membership inference attacks.

Finally, we compare the models trained using GECKO with prior state of the art defences against membership inference attacks, namely, Adversarial Regularization [18] and Differential Privacy [2]. We show that our proposed models improve the trade-offs between the efficiency, accuracy, and privacy compared to the baselines approaches. Our work provides the first systematic evaluation of efficiency-accuracy-privacy trade-offs to design a novel training methodology. The code of GECKO is publicly available[1].

The paper is outlined as follows: Section 2 presents background and Section 3 describes GECKO . Baselines and experimental setup are described Section 4, and the evaluation of the proposed algorithm and a comparative analysis with state of the art baselines are given in Section 5. Related work are then presented Section 6 before concluding in Section 7.

2 BACKGROUND

2.1 Embedded Deep Learning

On-device processing is an attractive alternative compared to centralized processing of data from IoT devices. Such on-device processing reduces the overhead of communicating data from the devices to the servers, lowers the privacy and security risk associated with storing sensitive data on untrusted central server and lowers the latency for obtaining results from processing [29].

Different efficiency requirements can be adopted to design NNs for embedded systems. These differences make it difficult to decide which primitive is the best fit for designing a privacy-preserving system for a particular application. The list below presents the most important efficiency requirements accounted by designers of embedded systems using NNs, and privacy is not part of this list:

- *Energy Efficiency.* Energy consumption is a vital constraint for low powered embedded or IoT devices which operate for long duration while maximising their battery lifetime. While executing NNs, every Multiply Accumulate (MAC, a common step that computes the product of two numbers and adds that product to a register in which intermediate results are stored) requires memory access for reading weights, inputs and intermediate output from previous layer and one write to store the computed output. These read-write operations consume significantly higher energy than actually performing the MAC operation in the CPU. Energy efficiency is achieved by reducing the memory access by (a) optimizing hardware to exploit sparsity in MACs and (b) reducing the precision to increase the throughput of data.
- *Computation Efficiency.* The total MAC operations between the parameter matrix and input activation function quantifies the requirement of computation efficiency. The processing rate of MAC operations is constrained by the CPU on embedded device which is reduced by reducing the total number of parameters. Additionally, replacing MACs with cheaper binary arithmetic significantly lowers the computational overhead.
- *Memory Efficiency.* The total size of the model measured in terms of the memory storage for model parameters and additional runtime storage for intermediate outputs should be within the memory constraints of the embedded device. This is achieved in two ways: (a) reducing the precision of the parameters and intermediate outputs and (b) pruning the parameters by increasing sparsity.

2.2 Privacy Threat: Membership Attacks

NNs for embedded systems do not account for privacy threats, however these threats still exist if personal and potentially sensitive data feed the system. For instance, if the adversary learns something specific about a user's data record used in the training dataset, we refer to such information as privacy leakage. This privacy leakage about a user's record can be, for instance, the membership details of the record in the training set of the model, referred to as membership inference attacks. Alternatively, an adversary can learn sensitive attributes about the user's data record which can be used to reconstruct the sensitive training dataset. In this work, we specifically use membership inference attacks to quantify information leakage in machine learning models [26].

Machine Learning algorithms learn a function $f : X \rightarrow Y$ mapping from the input space X to the space of corresponding class labels Y. This is modeled as an optimization where the objective is to find the parameters θ by minimizing the model's loss, $min_\theta L(f(x), y; \theta)$. Machine learning models are more confident while predicting the class of already seen train data record compared to an unseen test data record. Membership inference attacks exploit this difference in the model's confidence to classify a new data record as being a "Member" or "Non-Member" of the model's training data. This is a binary decision problem where the adversary classifies the membership of a given input x using the model's output prediction $f(x; \theta)$ to infer whether a given data record was used in the model's training data or not. Formally, given a user's data record $x \sim P(X, Y)$, where $P(X, Y)$ is the data distribution from which the training data D_{train} was sampled, the adversary estimates $P(x \in D_{train})$ using the model's prediction $f(x; W)$. Empirically, the adversary identifies a

[1] Anonymized for Submission

threshold to estimate whether $x \in D_{train}$ which can also be learnt using a binary classifier. In this work, we use the confidence score attack where the adversary obtains $f(x; W)$ and finds the maximum posterior and infers $x \in D_{train}$ if the maximum is greater than a threshold [22, 34]. The attack is based on the observation that the maximal posterior of a member data record is higher (more confident) than a non-member data record of the training dataset.

In this threat model, we consider a blackbox setting where the adversary is assumed to have no knowledge about the target model. Formally, given a target model $f()$, the adversary only sees the final model prediction $f(x; \theta)$. The adversary does not know the architecture of $f()$ and the model parameters θ. We do not consider whitebox setting where the adversary has the access to both the model output predictions $f(x; \theta)$ as well as the architecture of $f()$ and the model parameters θ. Indeed, this whitebox setting does not necessarily result in any benefit to the adversary in terms of attack accuracy (shown theoretically [21] and empirically [25, 28]. Consequently, blackbox setting is the more practical setting seen typically in Machine Learning as a Service (MLaaS) where the adversary submits an input query to the trained model on the Cloud via an API and receives the corresponding output.

3 GECKO: DESIGN OVERVIEW

In this section we detail Gecko . Gecko is a technique to construct NNs dedicated to IoT with efficiency, accuracy and privacy as main requirements:

- *Privacy.* The model should preserve the privacy of an individual's data record in the training set of the model against inference attacks.
- *Efficiency.* The model should work with low energy, memory and computation capacity for practical deployment to embedded and mobile devices.
- *Accuracy.* The model should be highly accurate.

To fulfill these requirements, Gecko is composed of two phases. In Phase I (Section 3.1), a first model is trained to ensure the efficiency and privacy. Ensuring these requirements is however done at the cost of the accuracy. Consequently, in Phase II (Section 3.2), the first model trained in Phase I is then optimized for improving the accuracy.

3.1 Phase I

We first quantize the model's parameters and intermediate activations. We specifically binarize the values to restrict them to the values of {+1,-1}. This operation (as seen in Section 5.2.3) results in high resistance to inference attacks as well as satisfies the different efficiency requirements. The model achieves computation efficiency by replacing the expensive matrix multiplications with simple boolean arithmetic operations, i.e, XNOR computations. Alternatively, instead of using multiplication and addition circuits in the hardware, we leverage XNOR logic on the inputs followed by a bitcount operation (counting the number of high bits "1" in a binary output sequence). The equation can be represented as follows:

$$\mathbf{x} \cdot \mathbf{w} = N - 2 \times \text{bitcount}(\text{xnor}(\mathbf{x}, \mathbf{w})) \quad (1)$$

In terms of memory efficiency, binarization results in a direct reduction of the model size as well as intermediate output memory requirements by 32x to 64x. Lowering the precision also reduces the number of memory access by 32x to 64x resulting in a significant decrease in the energy consumption.

Algorithm 1 Inference Stage of Binary Neural Network with XNOR Operations where W_k^b are the binarized weights (W_k) and a_k is the activation of the k^{th} layer

for $k = 1$ to L **do**
  $W_k^b \leftarrow \text{Binarize}(W_k)$
  $a_k \leftarrow N - 2 \times \text{bitcount}(\text{xnor}(\mathbf{a_{k-1}^b}, \mathbf{W_k^b}))$
  if $k < L$ **then**
    $a_k^b \leftarrow \text{Binarize}(a_k)$
  end if
end for

The complete inference stage of the Binarized NN with XNOR computation is given in Algorithm 1. The matrix multiplication between the previous layer activation a_{k-1} and the current layer's weights with the bitcount of XNOR operation's output. The function Binarize() is a deterministic thresholding function which maps the input values to the set {-1,+1}. In addition to the above design, we use additional optimizations for XNOR-Net to avoid a significant loss in accuracy. It is well documented that it is difficult to converge a binarized model during training [31] in case of incompatible hyperparameter settings. To this extent, we use the first and last layer of the model as full precision. These additional optimizations have been used previously for XNOR based networks [20, 30] and provides higher accuracy and model convergence at a small cost of memory and energy consumption overhead.

3.2 Phase II

While we optimize for both privacy and efficiency in Phase I (at the cost of significantly degrade the accuracy), we restore in Phase II the accuracy close to the original full precision accuracy by using knowledge distillation [10]. Here, we consider a pre-trained teacher model $f_{teacher}()$ with state of the art accuracy on the classification task and use it to guide the training of the quantized classifier. During training of the quantized model (student), we do not compute the loss between the true label y and predicted label $f_{student}(x)$. We instead estimate the loss between the predicted label $f_{student}(x)$ and the predicted label for the full precision teacher model $f_{teacher}(x)$. The loss function in knowledge distillation $Loss_{KD}(f_{student}, f_{teacher})$ is given as

$$\sum_{k=1}^{m} f_{student}(x_k) log(f_{teacher}(x_k)) + f_{teacher}(x_k) log(f_{student}(x_k)) \quad (2)$$

This ensures that the student model learns to map the prediction boundary of the teacher model and mimics the

prediction behaviour for different inputs. Therefore, the accuracy of the student model increases compared to the original baseline of standalone training without the teacher model.

4 EXPERIMENTAL SETTING

We carried out an extensive evaluation of GECKO. We first describe the datasets and architectures used in this analysis (Section 4.1) before to present the considered comparative baselines (Section 4.2) and metrics (Section 4.3).

4.1 Datasets and Architectures

For evaluating and comparing different efficiency algorithms, we mainly use two standard benchmarking datasets: Fashion-MNIST and CIFAR10. We train the model for 75 epochs for FashionMNIST and 100-150 epochs for CIFAR10.

FashionMNIST. This dataset consists of 60,000 training examples and a test set of 10,000 examples. Each data record is a 28×28 grayscale image which is mapped to one of 10 classes consisting of fashion products such as coat, sneaker, shirt, shoes. For this dataset, we use a modified LeNet architecture with two convolution layers followed by maxpool and dense layers: [Conv 32 (3,3), Conv 64 (3,3), Maxpool (2,2), Dense 128, Dense 10] (Architecture 1). Additionally, we use a fully connected model [512,512,512] (Architecture 2).

CIFAR10. This dataset is a major image classification benchmarking dataset where the data records are composed of 32×32 RGB images where each record is mapped to one of 10 classes of common objects such as airplane, bird, cat, dog. For this dataset, we use standard state of the art architectures: Network in Network (NiN), AlexNet and VGGNet.

4.2 Comparative Baselines

4.2.1 NN models for embedded systems. Model designers use three state of the art approaches for designing efficient NN models for embedded systems: (a) Model Compression via Pruning, (b) quantization of model parameters and activations and (c) designing standard architectures (off the shelf efficient architectures).

- *Model Compression (Pruning).* NNs are overparameterized, i.e, have a large number of redundant weights. Pruning the network refers to removing these redundant weights (setting them as zero) without a degradation of model accuracy. The pruning operation results in a model with sparse parameters for which the hardware designers skip the multiplication and memory storage to improve efficiency. Sparse weights can be stored in a compressed format in the hardware using the compressed sparse row or column format which reduces the overall memory bandwidth [5, 6, 8]. Aggressive pruning, while compressing the model significantly, requires to be re-trained to restore the model's original accuracy. For a sensitivity threshold T, the parameters close to zero are replaced by zero:

$$f(W) = \begin{cases} 0, & \text{if } -T \leq w \leq T \\ w, & \text{otherwise} \end{cases}$$

- *Off-the-Shelf Efficient Architectures.* NNs can be redesigned by changing the hyperparameters (i.e., filter size in convolution, number of layers and their types) to reduce the number of parameters and hence, the memory footprint. One approach is to replace larger convolution filters with multiple smaller filters with less number of parameters but covering the same receptive fields. For instance, one 5x5 filter can be replaced by two 3x3 filters. Alternatively, 1x1 convolutional layers reduce the number of channels in output feature map, lowering the computation and number of parameters. For instance, for an input activation of dimension 1x1x64, 32 1x1 convolutional filters downsamples the activation maps to get an output of 32 channels. Such optimizations enable to design compact network architecture with layers having lower parameters compared to the original model, extensively adopted in MobileNet [23] and SqueezeNet [14].
- *Quantization.* Quantization reduces the precision of the model's parameters and the intermediate activations during execution. Quantization maps parameters and activations to a fixed set of quantization levels [13]. The number of quantized levels determines the precision of the operands ($log_2(\#levels)$). Reducing the precision of the parameters lowers the storage cost of the model in memory. In addition, reducing the precision of activations lowers the computation overhead by replacing MACs with binary arithmetic and reduces the energy consumption by lowering the memory accesses and increasing throughput. Aggressively quantizing the parameters and activations to binary and ternary precision significantly improves the overall efficiency, however, at the cost of accuracy [20]. For instance, Binarized NNs quantize the operands to {-1,+1} values [12] while ternary NNs have values {-w, 0, w} where w can be fixed or learnt during training [16]. These are examples of uniform quantization. Alternatively, weight sharing maps several parameters to a single value reducing the number of unique parameters [6]. This mapping is done using K-Means clustering or a hashing function where a *codebook* maps different parameters to the corresponding shared values.

4.2.2 Privacy defences. We consider two state of the art baselines: Adversarial Regularization and Differential Privacy. These defences have mainly focussed on improving the model's generalization and reduce overfitting which has been considered as the main cause for leakage through membership inference attacks.

- *Adversarial Regularization (AdvReg) [18].* Here, the problem of defending against membership inference attack is modelled as a minimax game between two NNs: classifier network and attacker network. The two networks are trained alternatively with conflicting objectives: first, the attacker network is trained to distinguish between the training data members and non-members followed by training the classifier network to minimize the loss as well as fool the attacker network. Formally, the target classifier outputs a single probability $I(F(x), y) \in [0, 1]$ which indicates the likelihood of x being part of the training data. The classifier

minimizes the loss along with the output of the attacker classifier balanced with a privacy risk hyperparameter λ : $min_\theta l(F(x), y) + \lambda log(I(F(x), y))$.

- *Differential Privacy (DP) [2].* In this work, we specifically consider DP-SGD which adds carefully crafted noise to the gradients during backpropagation in SGD algorithm. The noise is sampled from a Laplacian or Gaussian distribution proportional to the model's sensitivity which is then added to the gradients during backpropagation. This provides provable bound on the information leaked about an individual data record in the dataset and ensures that the presence or absence of a data record does not change the model's output, hence defending against membership inference attacks.

4.3 Metrics

Efficiency. We evaluate efficiency on memory efficiency, computation efficiency and energy efficiency. Memory efficiency is compared based on the reduction in the memory footprint of the model computed from the parameters stored in the memory. Computation efficiency is compared based on the reduction in the MAC operations which influences the execution time. Finally, the energy consumption is compared based on memory accesses from reading inputs and writing results to the memory.

Privacy. We use the inference attack accuracy to estimate the success of membership inference attack. An accuracy above random guess 50% indicates a training data leakage through membership inference attack. This indicates that the adversary is able to identify the membership details of a data record with an accuracy higher than random guess. The success of inference attack accuracy is strongly correlated with the model's extent of overfitting empirically measured as the difference between the train and test accuracy (i.e., generalization error). Higher generalization error (i.e., overfitting) results in higher distinguishability between the test and train resulting in higher membership inference accuracy [26]. Additionally, the accuracy of the model is computed using the model's performance on unseen test data.

5 EVALUATION

We carried out an extensive evaluation of Gecko. We first analyse the three state of the art techniques for efficiency model computation (Section 5.1) and privacy leakage (Section 5.2), before summarizing the comparison (Section 5.3). We then evaluate the proposed training methodology for Phase I (Section 5.4) and Phase II (Section 5.5). Finally, we compare Gecko against defence schemes (Section 5.6).

5.1 Efficiency Requirements

In this section, in the view of the memory-computation-energy efficiency requirements, we compare the three baseline algorithms (i.e., model compression, off-the-shelf architecture, and quantization). Since, significant literature has compared the efficiency empirically [30], we provide a qualitative analysis for the considered baselines.

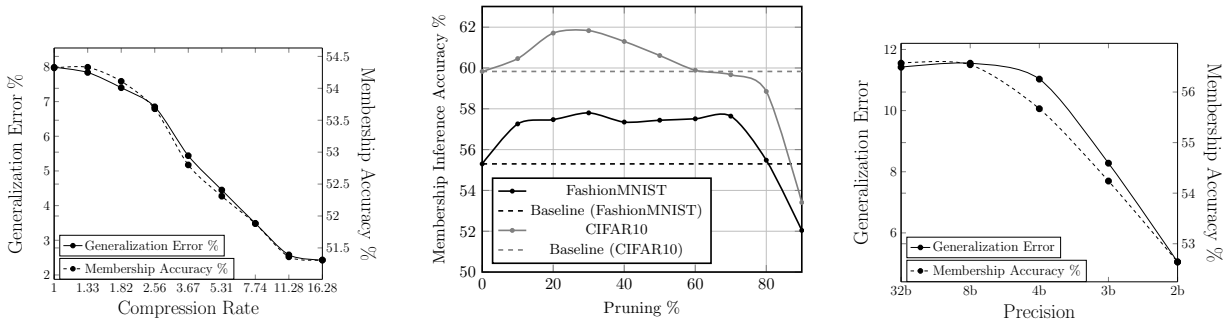
Memory Efficiency. Off the shelf models are designed to specifically reduce the memory footprint. For instance, the memory footprint of Squeezenet and MobileNet is 5MB and 14Mb compared to 250Mb of Alexnet and >500Mb of VGG architectures [14, 23]. Additionally, quantization lowering the model precision from 64 or 32 bit floating point to binary precision results in a direct reduction of 64x or 32x in the overall memory footprint of the model. However, in case of model compression the model parameters which are pruned are simply replaced by a value of "0". Hence, storing even the "0" parameter takes up memory and does not necessarily decrease the overall memory footprint unless the hardware is optimized to skip the storage of all the zero values in the memory. This requires additional logic to check for zero valued parameters in a dictionary.

Computation Efficiency. Design of efficient off-the-shelf architectures replaces the complex matrix-vector multiplications to smaller dimensions. This reduces the overall number of parameters but it has been shown empirically[2] that this does not necessarily reduce the number of multiply accumulate operations [3]. In case of parameter pruning (i.e., model compression), achieving efficiency requires additional hardware optimization. Particularly, instead of actually computing the multiplications with "0" pruned values, the hardware optimization enables the user to skip the computation and replace the output by a "0" directly. For quantized models with binarized parameters and activations the MACs can be replaced by binary XNOR operations, maxpool replaced by OR operation, while the activations can be replaced by checking the sign bit and hence reducing the FLOPS drastically [1]. This results in high computational efficiency and hence, faster inference.

Energy Efficiency. Energy efficiency does not vary much with reduction of number of parameters and data type, but the number of memory accesses play vital role [11]. Specifically, for the case of off-the-shelf architectures, while computation efficiency improves, the energy efficiency is close to large scale state of the art models like AlexNet [14, 30]. Alternatively, for the case of model compression, energy efficiency can be marginally improved by additionally providing hardware optimization [6, 33]. For quantization, however, the energy efficiency is high as the memory access can be drastically reduced by increasing the throughput of data fetched from the memory. Specifically, lowering the precision from 32 bit floating point to binary results in lowering the memory accesses and 32x improvement in energy consumption [12, 20]. While some improvements are seen natively for quantized models (from replacing MACs with XNOR), higher benefits can be achieved via additional hardware optimization [32]. The benchmarking of energy consumption for different optimization and architectures is well explored and out of scope of this work. We refer the readers to [30] for more details.

In summary, compared to different optimization techniques, the quantized architectures show significant benefits for different efficiency requirements over the other alternatives.

[2] https://github.com/albanie/convnet-burden



(a) Impact of Model Pruning on Privacy (FashionMNIST)

(b) Impact of Retraining Pruned Model on Privacy

(c) Mitigating Privacy Leakage via Weight Sharing (FashionMNIST)

Figure 1: Pruning the model lowers the membership inference leakage at the cost of accuracy. Retraining the pruned model to restore accuracy results in a higher membership privacy leakage compared to uncompressed baseline model. This additional leakage can be mitigated by weight sharing at the cost of accuracy.

5.2 Privacy Leakage

In this section, we evaluate the information leakage through membership inference attacks for the three baseline algorithms considered.

5.2.1 Model Compression. We evaluate the privacy leakage on compressing a model by pruning the connections in the model. Here, pruning is achieved by replacing some of the parameters with "0" value. As described in the original paper [7, 8], pruning is followed by retraining the model to restore the model's original accuracy with the pruned connections. We validate the impact on membership privacy on model compression on four datasets but only report results for FashionMNIST and CIFAR10 for space reason[3].

Impact of Pruning Parameters. On pruning the model, the model's test accuracy decreases but also lowers the membership inference accuracy (Figure 1a). As the compression rate increases, the generalization error decreases (owing to a decrease in both train and test accuracy) with a decrease in membership accuracy to close to random guess. This is expected as the parameters are responsible for memorizing the training data information [4, 19, 27].

Impact of Retraining Pruned Model. Interestingly, on retraining the pruned model, we observe that the membership inference accuracy is much higher than the original unpruned baseline model (Figure 1b). This indicates that model compression in turn increases the overall privacy leakage. This can be attributed to the lower number of parameters forced to learn the same amount of information stored previously in the unpruned model with larger number of parameters. In other words, the same amount of information is now captured by less number of parameters resulting in higher memorization of information per parameter. As the model is compressed (i.e., pruned), the number of parameters decreases which results in increase in information per parameter. However, on aggressive pruning, the train and test accuracy also decreases resulting in a decrease in the information per parameter, which is empirically indicated by a decrease in membership inference accuracy from 75% of pruning.

In summary, model compression results in a higher membership privacy leakage compared to the baseline uncompressed model making it a poor candidate for applications with sensitive data.

Mitigating the Privacy Risks in Pruned Models. We describe a potential approach to mitigate the privacy risk of the compressed models without requiring to modify the model's training. The post-hoc approach utilizes the weight sharing (i.e., a class of quantization techniques) for the compressed model. This is however, accompanied by a decrease in the model's prediction accuracy indicating a privacy-utility trade-off. As shown Figure 1c, reducing the precision from 32 bits to 2 bits results in a decrease in inference accuracy from 56.57% to 52.64% for FashionMNIST. This decrease in inference attack accuracy is caused by a decrease in generalization error due to decrease in prediction (both train and test) accuracy of the model. For the experiments, we use the compressed model with highest privacy leakage (by sweeping sensitivity threshold values) to evaluate the effectiveness of weight sharing on the worst case condition. This pipeline approach of pruning followed by retraining followed by weight sharing, not only maintains the algorithm's objective for efficiency but is used as a post-hoc approach to reduces the overall inference risk [6, 7].

5.2.2 Off-the-Shelf Efficient Architectures. In this section, we evaluate two popular state of the art architectures, SqueezeNet and MobileNet, trained on CIFAR10 dataset used for low powered systems. This evaluation is done only on CIFAR10 dataset as these state of the art architectures are not adapted for the FashionMNIST dataset. As seen in Table 1, the SqueezeNet and MobileNet models shows lower membership inference accuracy of 53.07% and 55.57% compared to larger models which have higher privacy leakage.

[3]the two other datasets (capturing preferences of online customers and capturing location check-ins [26]) depict similar results

CIFAR10				
Architecture	**Memory Footprint**	**Train Accuracy**	**Test Accuracy**	**Inference Accuracy**
SqueezeNet	5 MB	88.21%	81.92%	53.07%
MobileNetV2	14 MB	97.50%	87.24%	55.57%
AlexNet	240 MB	97.86%	80.34%	60.40%
VGG11	507 MB	99.13%	86.43%	58.04%
VGG16	528 MB	99.58%	88.95%	58.70%
VGG19	549 MB	99.09%	88.18%	57.85%

Table 1: Model complexity influences the membership inference leakage.

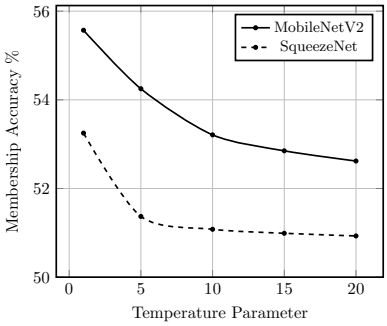


Figure 2: The privacy leakage of off-the-shelf models is reduced by increasing the softmax temperature (CIFAR10 dataset).

Further, the membership inference accuracy of SqueezeNet and MobileNet can be further reduced close to random guess by increasing the temperature parameter of the softmax function applied to the output. Increasing the temperature parameter reduces the granularity of the model's output and is given by $F_i(x) = e^{\frac{z_i(x)}{T}} \; / \sum_j e^{\frac{z_j(x)}{T}}$ where $z(x)$ computes output of the model before the softmax layer. For the case of SqueezeNet, we are able to reduce the inference accuracy to 50.93% from 53.07% while for MobileNet we can reduce the inference accuracy to 52.62% from 55.57% as seen in Figure 2. This reduction in membership inference accuracy is without any cost of the prediction test accuracy of the model.

5.2.3 Quantization. In this section, we evaluate the technique of reducing the precision of both model's parameters and intermediate activations. Further, we consider the extreme case of binarizing the parameters and activations allowing to evaluate on the most optimized case. We evaluate on FashionMNIST dataset for two architectures with convolutional and fully connected layers as seen in Table 2.

In both architectures, we see that computation on binarized parameters and activations reduces the inference risk by a small value. However, on replacing the MAC operations with XNOR operations, we observe that the inference risk decreases close to random guess, however, at the cost of prediction test accuracy. The CIFAR10 results corresponding to the XNOR operations and its privacy comparison with full precision counterpart is indicated in Table 4.

FashionMNIST				
Architecture	**Memory Accuracy**	**Train Footprint**	**Test Accuracy**	**Inference Accuracy**
Architecture 1				
Full	38.39 MB	100%	92.35%	57.46%
BinaryNet	1.62 MB	88.68%	86.9%	55.45%
XNOR-Net	1.62 MB	87.19%	85.68%	51.05%
Architecture 2				
Full	29.83 MB	99.34%	89.88%	54.86%
BinaryNet	0.93 MB	97.61%	89.60%	54.30%
XNOR-Net	0.93 MB	92.67%	86.68%	51.74%

Table 2: Reducing the model precision decreases the inference attack but at the cost of test accuracy.

In summary, we observe that quantization, specifically binarization of parameters and activation along with XNOR computation, provides strong resistance against inference attacks compared to model compression and off-the-shelf architectures.

5.3 Summary of Comparison

We summarize the properties satisfied by each of the approach used for designing NN models for embedded systems in terms of privacy, computation, memory and energy efficiency in Table 3. Here, we mark the attributes which are satisfied with ●, requires additional hardware optimization as ◐ and does not satisfy the property with a ○.

Requirements	Compression	Quantization	Off-the-shelf
Computation Efficiency	◐	●	○
Memory Efficiency	◐	●	●
Energy Efficiency	◐	●	○
Privacy	○	●	◐

Table 3: Only quantization satisfies all requirements.

In order to design NNs for embedded devices, quantization (using binarization with XNOR computation) is an attractive design choice which not only satisfies the computation, memory and energy efficiency but also provides high resistance against inference attacks. On the other hand, model compression without any weight sharing modifications, leaks more training data membership details making it significantly more vulnerable to membership inference attacks. Additionally, it requires hardware support and optimization to achieve better efficiency. Off-the-shelf architectures, while provide decent privacy, does not satisfy all aspects of efficiency. Consequently, Gecko adopts quantization as a NN design to provide a good three dimensional trade-off between privacy-efficiency-accuracy.

5.4 Evaluating Phase I

In Phase I of Gecko, we quantize the model and replace the MACs with cheap XNOR operations. We observe (Table 4) that the inference attack accuracy decreases significantly for all the three architecture close to random guess (∼50%). Specifically, the inference accuracy decreases from 56.69% to 51.76% for NiN, 60.40% to 51.40% for AlexNet and 58.70% to

CIFAR10				
Architecture		Train Accuracy	Test Accuracy	Inference Accuracy
NiN	Full Precision	98.16%	86.16%	56.69%
	Binary Precision	81.93%	78.74%	51.76%
AlexNet	Full Precision	97.86%	80.34%	60.40%
	Binary Precision	68.62%	66.8%	51.40%
VGG13	Full Precision	99.58%	88.95%	58.70%
	Binary Precision	79.67%	74.64%	52.65%

Table 4: Reducing the precision of models lowers the membership privacy leakage at the cost of accuracy.

52.65% for VGGNet. However, since Phase I only optimizes the network for privacy and efficiency, the resultant model shows poor utility (accuracy). We observe a significant loss in test accuracy for all the three models: around 8% accuracy drop from 86.16% to 78.74% for NiN; 14% accuracy drop from 80.34% to 66.8% for AlexNet; 14% for VGG model from 88.95% to 74.64%. In order to restore the accuracy, we use knowledge distillation as described in Phase II of the Gecko training methodology.

The privacy provided by quantized NN is due to the decrease in overfitting, empirically measured using the difference between the train and test accuracy. The leakage in inference accuracy is attributed to the higher overfitting in models as well as memorization of the training data information in the form of the parameters, which are specifically tuned to achieve high performance on the train data [4, 19, 27]. This is attributed to the reduction in learning capacity of the model on quantizing the parameters which lowers the sensitive training data information memorized by the parameters on lowering the precision. Further, the quantization acts as a noise to strongly regularize the model [12]. At the same time, this optimization provides high degree of efficiency to be executed on low powered embedded devices.

5.5 Evaluating Phase II

The objective of Phase II is to enhance the accuracy of the model trained in Phase I (i.e., quantized model with XNOR computations which depicts high inference attack resistance and efficiency). In Phase II, we use the teacher-student model (described in Section 3) to train the quantized student model being guided using the output predictions of the full precision teacher model. Here, Phase II is heterogeneous, i.e, we are flexible to choose any full precision teacher model which can provide high accuracy on the considered dataset (Table 5). Here, we consider pre-trained state of the art architectures[4]: DenseNet169 and ResNet50, along with the full precision versions of NiN, Alexnet and VGGNet. The standalone test accuracy of the DenseNet169 and ResNet50 architectures are 92.84% and 92.12% respectively with inference accuracy around to 55% while the full precision accuracies for NiN, AlexNet and VGGNet are given in Table 4.

The first set of experiments combine the same full precision model architectures with the quantized model versions,

[4]https://github.com/huyvnphan/PyTorch_CIFAR10

Teacher	Student	Train Accuracy	Test Accuracy	Inference Accuracy
Standalone Models				
Binary NiN	None	81.93%	78.74%	51.76%
Binary AlexNet	None	68.62%	66.8%	51.40%
Binary VGG13	None	79.67%	74.64%	52.65%
Homogeneous Architecture Distillation				
NiN	Binary NiN	90.49%	83.52%	53.90%
AlexNet	Binary AlexNet	76.79%	73.5%	51.85%
VGG13	Binary VGG13	89.45%	81.58%	54.98%
Heterogeneous Architecture Distillation				
DenseNet169	NiN	92.84%	83.71%	54.95%
DenseNet169	AlexNet	81.87%	76.23%	53.51%
DenseNet169	VGG13	93.45%	85.8%	54.17%
ResNet50	NiN	91.74%	83.77%	54.53%
ResNet50	AlexNet	80.12%	74.92%	53.12%
ResNet50	VGG13	94.23%	86.52%	54.46%

Table 5: Phase II of Gecko improves the accuracy of the private-efficient model from Phase I.

i.e, full precision NiN with Binarized NiN (i.e., homogeneous knowledge distillation). Here, we see that there is 5% increase in test accuracy (from 78.74% reported Table 4 to 83.52%) for NiN with an increase of 2% in inference attack. Similarly, there is an increase of 7% test accuracy for AlexNet with a very minimal privacy leakage increase of 0.45%; and increase of 7% test accuracy at the cost of 2% inference attack accuracy for VGGNet. For heterogeneous knowledge distillation, i.e, combining other architectures (DenseNet169 and ResNet50) with the quantized models from Phase I, we see that the increase in test accuracy is only minimally higher than the homogeneous models for NiN and AlexNet but a significantly higher increase in the inference attack accuracy. However, in case of VGGNet, we observe an increase of 4% additional test accuracy compared to homogeneous knowledge distillation with a minimal decrease in the inference test accuracy. In Phase II, increase in test accuracy is accompanied with a small but acceptable increase in the inference attack accuracy indicating a privacy-utility trade-off. Hence, the choice of using homogeneous or heterogeneous knowledge distillation is specific to the architecture and the privacy-utility requirements of the application. Compared to the full precision counterparts, we observe that the distilled models show an accuracy degradation of only 3% for NiN(86.66% to 83.77%), 4% for AlexNet (80.34% to 76.23%) and 2% for VGGNet (88.95% to 86.52%).

The Gecko framework results in models which make the output confidence of the train and test data records similar reducing the inference attack accuracy (Figure 3). Further, the knowledge distillation enables to lower the loss of the model compared to the model trained in Phase I resulting in higher test accuracy as shown Figure 3 (c). However, this loss function is still higher than the full precision version indicating the test accuracy degradation of the proposed framework and a privacy-utility trade-off.

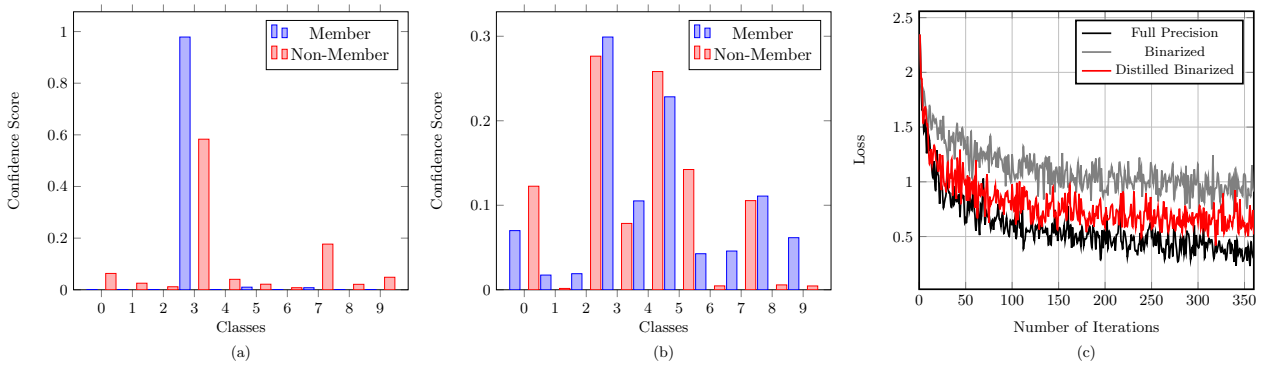


Figure 3: (a) Confidence scores are distinguishable between train and test data records in undefended models making them vulnerable to membership inference attacks, (b) Gecko models have indistinguishable confidence scores, (c) Loss functions in Phase I (Binarized) are higher than Phase II (Distilled Binarized) indicating the improvement in accuracy.

5.6 Gecko versus Prior Defences

The privacy defences proposed in literature can be categorized into (a) regularization based train-time defences and (b) post-training inference time defence. Adversarial Regularization, Differential Privacy and other standard regularization techniques such as L2 and Dropout modify the training of the neural network. Our Gecko training framework is also part of category (a) where we modify the training of the machine learning model in order to provide acceptable levels of privacy and accuracy. We do not consider post-training defences (e.g., MemGuard [15] which adds carefully crafted noise to the target model's output observations to ensure the misclassification of the adversary's attack classifier network) in the comparison as they can be used in addition to Gecko training framework.

The comparison of models trained using Gecko is shown Figure 4 (FP stands for Full Privacy and reports results without protection). Models trained using Gecko are comparable in test accuracy and resisting membership inference leakage to Adversarial Regularization and Differential Privacy. The inference accuracy for NiN is 52.90% (Gecko) compared to 54.09% (DP) and 51.92% (AdvReg) and test accuracy of 83.52% (Gecko) compared to 85.11% (DP) and 83.66% (AdvReg). For AlexNet, the inference accuracy is 51.85% (Gecko) compared to 52.81% (DP) and 51.83% (AdvReg) and test accuracy of 73.5% (Gecko) compared to 79.27% (DP) and 71.02% (AdvReg). For VGGNet, the inference accuracy is 53.17% (Gecko) compared to 52.90% (DP) and 53.33% (AdvReg) and test accuracy of 85.8% (Gecko) compared to 84.91% (DP) and 85.19% (AdvReg). In addition, our proposed models additionally provide efficiency guarantees enabling them to be used for embedded systems.

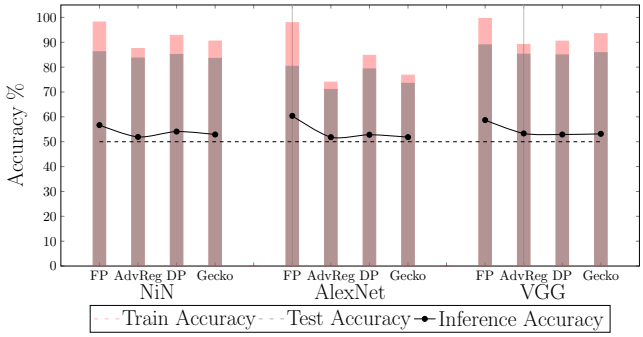


Figure 4: Gecko models are comparable to prior state of the art privacy defences in terms of test accuracy and inference accuracy while additionally ensuring efficiency.

6 RELATED WORK

Data privacy in Machine Learning addresses different inference attacks such as membership inference in a blackbox setting [22, 26] or in the context of whitebox setting [19]. Further, generative model have been shown to be vulnerable to membership inference attacks [9] and distributed setting such as in federated learning have also been exploited [17, 19]. These privacy leakage in machine learning models have been mainly attributed to the memorization of training data by the models [4, 27]. In order to mitigate against inference attacks several defences have been explored such as Differential Privacy [2], simple and adversarial regularization [18, 22] which aim to generalize the model and alternatively, adding noise to the predictions to increase error [15]. Alternatively, confidential computing aims to privately and efficiently compute machine learning models using secure multiparty computation [1]. Interestingly, post-training approaches assuming the adversary uses shadow model attack (e.g., MemGuard [15]), can exploit Gecko by using the models trained by this framework before to add noise.

Hardware-software co-design is crucial to accelerate the performance of NNs for embedded systems. Hardware accelerators reuse weights and intermediate computation enable

significant performance improvement [5]. Algorithmic optimizations have explored model compression through pruning [8] and reducing the precision of the model parameters and activations to binary [12], ternary [16] and generic quantization [13]. Binarization enables to replace multiplication with simple boolean logic improving the overall performance [20]. Alternatively, hardware optimizations have enabled to design NN accelerators for low precision NNs for further efficiency [32]. Further, specialized architectures designed for low memory footprint have also been extensively used for low powered devices such as mobile phones and micro-controllers [14, 23]. However, all these optimization designs do not accounted for the resistance against inference attacks. In this paper, we quantify the privacy leakage for different optimization and design algorithms for NNs and propose a training framework to reduce it.

7 CONCLUSIONS

On device processing of sensitive data using NNs on embedded systems requires a careful analysis of privacy, efficiency and accuracy of the algorithms which is currently lacking in literature. In this work, we propose a two phase GECKO training framework to design private, efficient and accurate NNs for execution on low powered embedded devices. We quantify the privacy leakage using membership inference attacks where the adversary aims to infer whether a given data record was used in the model's training data. We first provide a comprehensive privacy and efficiency analysis of state of the art algorithms for improving efficiency: model compression (pruning), quantization and efficient off-the-shelf architectures. We show that model compression leaks more information compared to the original (uncompressed) model while off-the-shelf architectures do not provide the best efficiency guarantees. Based on these observations, we use quantization as a design choice which shows high resistance against inference attacks while satisfying all the efficiency requirements. While Phase I of GECKO optimizes for privacy and efficiency, in Phase II, we improve the accuracy of the resultant model using knowledge transfer from full precision models. Our extensive evaluations of state of the art architectures on CIFAR10 dataset indicates that models trained using the proposed framework provides high resistance against membership inference attacks (comparable to other state of the art defences) but keeping high efficiency.

REFERENCES

[1] 2019. XONN: XNOR-based Oblivious Deep Neural Network Inference. In *USENIX Security*.

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. In *CCS*. 308–318.

[3] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. 2016. An Analysis of Deep Neural Network Models for Practical Applications. (05 2016).

[4] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. 2019. The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks. In *USENIX Security*. 267–284.

[5] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *ISCA*. 243254.

[6] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *ICLR*.

[7] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, Bryan Catanzaro, and William J. Dally. 2017. DSD: Dense-Sparse-Dense Training for Deep Neural Networks. *ICLR* (2017).

[8] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning Both Weights and Connections for Efficient Neural Networks. In *NIPS*. 1135–1143.

[9] Jamie Hayes, Luca Melis, George Danezis, and Emiliano De Cristofaro. 2019. LOGAN: Membership Inference Attacks Against Generative Models. *PETS* 1 (2019), 133 – 152.

[10] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. In *NIPS Deep Learning and Representation Learning Workshop*.

[11] M. Horowitz. 2014. Computing's energy problem (and what we can do about it). In *ISSCC*. 10–14.

[12] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized Neural Networks. In *NIPS*. 4107–4115.

[13] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. *J. Mach. Learn. Res.* 18 (2017), 6869–6898.

[14] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *CoRR* abs/1602.07360 (2016).

[15] Jinyuan Jia, Ahmed Salem, Michael Backes, Yang Zhang, and Neil Zhenqiang Gong. 2019. MemGuard: Defending against Black-Box Membership Inference Attacks via Adversarial Examples. In *CCS*. 259274.

[16] Fengfu Li and Bin Liu. 2017. Ternary Weight Networks. (2017).

[17] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting unintended feature leakage in collaborative learning. In *SP*.

[18] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2018. Machine Learning with Membership Privacy using Adversarial Regularization. In *CCS*. 634–646.

[19] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2019. Comprehensive Privacy Analysis of Deep Learning: Stand-alone and Federated Learning under Passive and Active White-box Inference Attacks. *SP* (2019).

[20] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks.

[21] Alexandre Sablayrolles, Matthijs Douze, Cordelia Schmid, Yann Ollivier, and Herve Jegou. 2019. White-box vs Black-box: Bayes Optimal Strategies for Membership Inference *(PMLR)*, Vol. 97. 5558–5567.

[22] Ahmed Salem, Yang Zhang, Mathias Humbert, Mario Fritz, and Michael Backes. 2018. ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models. *NDSS* (2018).

[23] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *CVPR*. 4510–4520.

[24] Virat Shejwalkar and Amir Houmansadr. 2019. Reconciling Utility and Membership Privacy via Knowledge Distillation. *arXiv:1906.06589* (2019).

[25] Virat Shejwalkar and Amir Houmansadr. 2019. Reconciling Utility and Membership Privacy via Knowledge Distillation. *CoRR* abs/1906.06589 (2019).

[26] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *SP*.

[27] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. 2017. Machine Learning Models That Remember Too Much. In *CCS*. 587601.

[28] Liwei Song and Prateek Mittal. 2020. Systematic Evaluation of Privacy Risks of Machine Learning Models. arXiv:2003.10595 [cs.CR]

[29] V. Sze. 2017. Designing Hardware for Machine Learning: The Important Role Played by Circuit Designers. *IEEE Solid-State Circuits Magazine* (2017), 46–54.

[30] V. Sze, Y. Chen, T. Yang, and J. S. Emer. 2017. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proc. IEEE* (2017), 2295–2329.
[31] Wei Tang, Gang Hua, and Liang Wang. 2017. How to Train a Compact Binary Neural Network with High Accuracy?
[32] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Heng Wai Leong, Magnus Jahre, and Kees A. Vissers. 2017. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. In *FPGA*.
[33] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2017. Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning. (2017).
[34] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha. 2018. Privacy Risk in Machine Learning: Analyzing the Connection to Overfitting. In *CSF*. 268–282.