Seesaw: Compensating for Nonlinear Reduction with Linear Computations for Private Inference

Fabing Li [1 2] **Yuanhao Zhai** [3] **Shuangyu Cai** [4] **Mingyu Gao** [4 5 6]

Abstract

With increasingly serious data privacy concerns and strict regulations, privacy-preserving machine learning (PPML) has emerged to securely execute machine learning tasks without violating privacy. Unfortunately, the computational cost to securely execute nonlinear computations in PPML remains significant, calling for new model architecture designs with fewer nonlinear operations. We propose Seesaw, a novel neural architecture search method tailored for PPML. Seesaw exploits a previously unexplored opportunity to leverage more linear computations and nonlinear result reuse, in order to compensate for the accuracy loss due to nonlinear reduction. It incorporates specifically designed pruning and search strategies, not only to efficiently handle the much larger design space of both linear and nonlinear operators, but also to achieve a better balance between the model accuracy and the online/offline execution latencies. Compared to the state-of-the-art design for image classification on ImageNet, Seesaw achieves 1.68× lower online latency and 1.55× lower total online + offline latency at 71% iso-accuracy, or 3.65% higher accuracy at iso-latency of 190 seconds, while using much simpler and faster search and training methods.

1. Introduction

Machine learning (ML), particularly deep neural networks, has become a ubiquitous technique in contemporary data-driven applications such as image/video classification and natural language processing (LeCun et al., 2015). The effectiveness of ML hinges on massive training data and extensive computational resources to efficiently process large neural network models. Consequently, ML tasks start to be outsourced to and deployed on cloud computing systems (ai.; clo; azu; int; aws; ope). However, such cloud-based deployment has raised serious concerns regarding the privacy of user data like health/medical records, financial status, and location information, which must now be sent to public cloud platforms and suffer from leakage risks.

In response to such privacy concerns, privacy-preserving machine learning (PPML) has been proposed to securely store and process users' sensitive data. During model training, differential privacy and federated learning techniques could be leveraged to protect individual person's data privacy (Zhang et al., 2018; Wei et al., 2020). More general PPML frameworks for both training and inference now heavily use cryptographic primitives, including homomorphic encryption and multi-party computation, to achieve provable security (Dowlin et al., 2016; Brutzkus et al., 2019; Badawi et al., 2018; Liu et al., 2017; Riazi et al., 2018; Juvekar et al., 2018; Mishra et al., 2020; Ng & Chow, 2021; Chandran et al., 2022; Zhang et al., 2023). However, despite extensive algorithm and system optimizations, their computational cost is still several orders of magnitude higher than the original plaintext models, restricting their practical usage in time-sensitive scenarios like online inference. The high processing overheads are primarily associated with nonlinear activation functions such as ReLU and Sigmoid, which require complex secure multi-party computation protocols with heavy cryptographic computations and frequent communication between the user and the cloud.

Great efforts have been made to alleviate the nonlinear computational cost in PPML, such as developing more efficient protocols for nonlinear operators (Mishra et al., 2020; Ghodsi et al., 2021; Lou et al., 2021), or reducing the number of such operations through pruning and neural architecture search (NAS) (Ghodsi et al., 2020; Cho et al., 2022a; Jha et al., 2021; Cho et al., 2022b; Kundu et al., 2023a; Huang et al., 2022; Kundu et al., 2023b). Nevertheless, almost all prior techniques simply started with an existing network architecture, and only focused on *reducing* the amount of nonlinear operators while struggling to minimize the corresponding negative accuracy impact. This inevitably

[1]Xi'an Jiaotong University, Xi'an, China [2]Institute for Interdisciplinary Information Core Technology, Xi'an, China [3]State University of New York at Buffalo, New York, USA [4]Tsinghua University, Beijing, China [5]Shanghai Artificial Intelligence Lab, Shanghai, China [6]Shanghai Qi Zhi Institute, Shanghai, China. Correspondence to: Mingyu Gao <gaomy@tsinghua.edu.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

causes accuracy degradation when nonlinear operations are reduced, suffering from the fundamental tradeoff between model accuracy and execution latency.

Our contributions. In this work, we aim to break this tradeoff, by exploiting opportunities to *use additional computations and data orchestration to compensate for accuracy loss due to nonlinear reduction*. Specifically, we propose two approaches: (1) *adding more linear operations* to the model to recover its decreased representation capacity; (2) *reusing the results of the remaining nonlinear operators* as much as possible through introducing residual shortcut links to the model topology. Although adding such linear and aggregation computations would increase the execution time in the insecure case, in the PPML scenario the latency impact is mainly limited to the offline pre-processing phase (Garimella et al., 2023), and has negligible online processing overheads compared to the dominant nonlinear cost, therefore exhibiting a unique opportunity.

We propose Seesaw, a one-shot NAS method that leverages the above compensation ideas to automatically search for optimized neural network architectures for PPML. Besides searching for how to selectively enable nonlinear operations under a given budget (Cho et al., 2022b; Kundu et al., 2023a), Seesaw incorporates several novel techniques to address the new challenges arising from the additional linear computations. First, as the increased pre-processing cost cannot simply be ignored (Garimella et al., 2023), we need to carefully control the amount of extra linear computations, in order to *balance the model representation capacity and the online/offline execution latencies*. Seesaw captures the online/offline latency impact into the overall loss function, and applies effective pruning methods to restrict latency while maintaining high accuracy. Second, the overall design space is significantly enlarged with the additional computations, which requires *new search and training strategies to efficiently explore*. Seesaw leverages a novel search strategy that combines (super)model training and network architecture search in one loop to accelerate convergence.

When evaluated on the ImageNet and CIFAR100 datasets under a wide range of nonlinear budgets, Seesaw is able to push the Pareto optimal frontier between the model accuracy and the execution latency compared to the state-of-the-art SENet (Kundu et al., 2023a). Specifically, on ImageNet, at iso-accuracy of 71%, Seesaw achieves lower latency than SENet, by 1.68× for the online phase only, and 1.55× for both online and offline together. At iso-latency of 190 seconds, Seesaw is better than SENet with 3.65% accuracy increase. On CIFAR100, Seesaw also outperforms SENet, with 1.53× online latency reduction at iso-accuracy of 70%, and 0.25% higher accuracy at iso-latency of 8 seconds. We have made Seesaw open source, available at `https://github.com/tsinghua-ideal/Seesaw`.

2. Background

Privacy-preserving machine learning (PPML) addresses the challenges of processing private user data on proprietary ML models, while not revealing any sensitive information to malicious participants during the computation. We focus on PPML inference. More specifically, privacy is protected if (1) the user learns no knowledge of the ML model except for the inference result of her own input data; and (2) the model owner gains no information about the user data.

Currently, there are mainly two approaches to realize PPML. Hardware-based trusted execution environments (TEEs) can protect sensitive data (Kunkel et al., 2019; Hunt et al., 2018; Hynes et al., 2018; Tramer & Boneh, 2019; Kim et al., 2020; Li et al., 2021; 2023), but TEEs are vulnerable to side channels, weakening their security (Chen et al., 2019; Wang et al., 2018). Cryptography-based PPML protects data privacy using modern cryptographic primitives (Gentry, 2009; Damgård et al., 2012; Yao, 1986). They offer theoretically provable, strong security guarantees. Our work optimizes the execution latency of crypto-based PPML solutions while minimizing the accuracy impact.

2.1. Cryptographic Primitives and PPML Protocol

Existing PPML has used various cryptographic primitives to best match different computation patterns in ML applications. **Fully Homomorphic Encryption (FHE)** (Gentry, 2009) allows for applying arbitrary functions composed of addition and multiplication on encrypted data (*e.g.*, user data or model weights). FHE is useful for linear operators (matrix multiplications, convolutions, *etc.*) in PPML, which account for a majority of computations in modern ML models. Previously, CryptoNets (Dowlin et al., 2016), HCNN (Badawi et al., 2018), TAPAS (Sanyal et al., 2018), LoLa (Brutzkus et al., 2019), and Faster CryptoNets (Chou et al., 2018) have leveraged FHE in PPML. Unfortunately, the computational complexity of FHE is quite high and can result in several orders of magnitude slowdown compared to insecure computing.

Another way to support linear computations is **Secret Sharing (SS)** (Damgård et al., 2012). PPML typically assumes two parties, the user and the model owner. SS transforms the data of each party into randomly split *shares*. Each share is hold by one party, and each party only sees its own share but not the full value, ensuring data privacy. Addition of two encrypted values, and multiplication between an encrypted value and a plaintext number, can be done locally with only simple operations. Therefore, the linear operators that involve the encrypted user data and the plaintext weights can be done efficiently. Gazelle (Juvekar et al., 2018) and DELPHI (Mishra et al., 2020) have used SS to replace FHE for higher online processing speed. Nevertheless, FHE is still needed during offline pre-processing to prepare the shares.

The remaining challenge is handling nonlinear operators such as ReLU and MaxPool. **Garbled Circuit (GC)** (Yao, 1986) takes the encrypted boolean representations of the two parties' input data, and securely computes an arbitrary boolean function composed of AND and NOR gates. Most existing PPML systems use GC to compute nonlinear operators (Liu et al., 2017; Mohassel & Zhang, 2017; Rouhani et al., 2018; Juvekar et al., 2018; Mishra et al., 2020). GC processing requires heavy cryptographic computations (*e.g.*, AES encryption) and frequent communication between the two parties. Such significant overheads compared to insecure processing are our main optimization target.

PPML protocol. In this work, we follow the overall flow of the state-of-the-art PPML system, DELPHI (Mishra et al., 2020). The protocol consists of two phases: an offline pre-processing phase, and an online inference phase. During offline pre-processing, we use FHE to generate the secret shares that will be used by the online SS scheme to compute the linear operators. Specifically, for a linear operator $\mathbf{y}_i = \mathbf{W}_i \cdot \mathbf{x}_i$, the user and the model owner each randomly samples a vector, $\mathbf{r}_i$ and $\mathbf{s}_i$, respectively. The user sends $\mathsf{Enc}(\mathbf{r}_i)$ (encryption of $\mathbf{r}_i$) to the model owner, who homomorphically computes $\mathsf{Enc}(\mathbf{W}_i \cdot \mathbf{r}_i - \mathbf{s}_i)$ using FHE. The user receives and decrypts this result to keep $\mathbf{W}_i \cdot \mathbf{r}_i - \mathbf{s}_i$. We also generate the GC boolean function for the nonlinear operators. For example, the user creates a GC function $f(\mathbf{a}) = \text{ReLU}(\mathbf{a} + (\mathbf{W}_i \cdot \mathbf{r}_i - \mathbf{s}_i)) - \mathbf{r}_{i+1}$ for the ReLU operator $\mathbf{x_{i+1}} = \text{ReLU}(\mathbf{y}_i)$, and sends it to the model owner.

In the online inference phase of a linear operator, the two parties start with each holding a share of the input, *i.e.*, $\mathbf{r}_i$ by the user and $\mathbf{x}_i - \mathbf{r}_i$ by the model owner. These shares are either from the results of the previous operator, or the user calculates $\mathbf{x}_i - \mathbf{r}_i$ and provides it to the model owner if this is the first layer. The model owner then evaluates $\mathbf{W}_i \cdot (\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i$ on its share. The user already has $\mathbf{W}_i \cdot \mathbf{r}_i - \mathbf{s}_i$ from the pre-processing phase. We can verify that these two values are exactly the shares of the output, *i.e.*, summed up to $\mathbf{W}_i \cdot \mathbf{x}_i = \mathbf{y}_i$. Thus we have maintained the induction condition.

The online inference uses GC for nonlinear operators, *e.g.*, $\mathbf{x_{i+1}} = \text{ReLU}(\mathbf{y}_i)$. The model owner has the GC function $f(\mathbf{a})$ from the offline phase. It sets $\mathbf{a}$ to its share of $\mathbf{y}_i$, *i.e.*, $\mathbf{a} = \mathbf{W}_i \cdot (\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i$, and evaluates $f(\mathbf{a})$ (involving heavy computation and communication) to obtain $\text{ReLU}(\mathbf{y}_i) - \mathbf{r}_{i+1} = \mathbf{x_{i+1}} - \mathbf{r}_{i+1}$, which is a valid share of the next operator's input. The user holds the other share $\mathbf{r}_{i+1}$.

2.2. Related Work

In the above PPML protocol, SS has made the online computations of linear operators almost as cheap as the original insecure processing, and GC offers general compute capability to support unmodified nonlinear operators to ensure the same accuracy level. However, the use of GC causes the main performance bottleneck (over 300× slower than linear computations in DELPHI (Garimella et al., 2023)). It is therefore necessary to focus on reducing the cost of nonlinear operators to speed up the PPML processing.

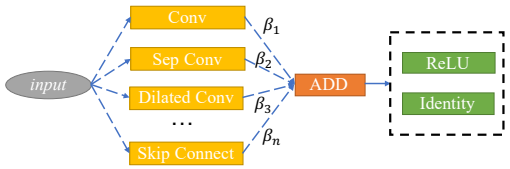
Recently there have been various proposals to address this issue. Some designs change the nonlinear operator computations from ReLU to more crypto-friendly alternatives. DELPHI (Mishra et al., 2020) replaced part of the nonlinear operators with linear approximation to exploit the latency-accuracy tradeoff, using neural architecture search (NAS) techniques. SAFENet (Lou et al., 2021) also used NAS to apply approximation, but at a more fine-grained level to reduce the accuracy impact. Circa (Ghodsi et al., 2021) reconstructed ReLU into a sign test (by GC) plus a multiplication (by SS), in order to reduce the processing cost. Other solutions reduce (*i.e.*, prune) the amount of ReLU operators in existing neural network structures. CryptoNAS (Ghodsi et al., 2020) rearranged the ReLU operators and used a macro-search algorithm, ENAS, to search for a network with fewer nonlinear operators. Sphynx (Cho et al., 2022a) instead used micro-search approaches to design its building blocks more thoroughly for higher accuracy. DeepReDuce (Jha et al., 2021) pruned the model in a more fine-grained manner at the channel level, and further improved accuracy through knowledge distillation. SNL (Cho et al., 2022b) was inspired by the parameterized ReLU and realized pixel-level ReLU pruning. SENet (Kundu et al., 2023a) proposed the concept of ReLU sensitivity, which distinguished the importance of different nonlinear operators and realized automated ReLU pruning.

Besides the online cost of the nonlinear operators, preparing the SS data of the linear operators through FHE at offline also introduces non-negligible overheads. Garimella et al. (2023) alleviated this issue by proposing layer-parallel HE (LPHE) to process these offline computations in parallel.

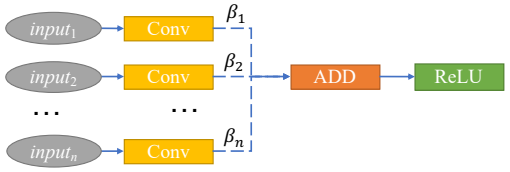
For the model representation capability, Zhang et al. (2021) showed that even the simplest two-layer neural network with a sufficiently large number of parameters could perfectly fit any labeling of training data. Other studies (Arora et al., 2018; Guo et al., 2020) indicated the benefits of over-parameterization for better performance, showing that adding a sufficient number of parameters (linear operators) could greatly improve the network's ability.

3. Design

Previous PPML designs that aimed to reduce the nonlinear cost (Section 2.2) suffered from a common limitation: they merely reduced the ReLU operators without reconsidering the overall network architecture. This inevitably decreases the representation capacity of the model. Since the represen-



(a) Sampling block.



(b) Aggregation block.

Figure 1. Main building blocks of the Seesaw search space. ReLU is always placed after a weighted element-wise ADD to save nonlinear operations and control linear operations.

tation capacity is jointly determined by both the linear and nonlinear operators, our key idea is to *compensate* for the accuracy loss caused by reduced nonlinear operators. We thus propose Seesaw, a one-shot NAS method to automatically search for crypto-friendly model architectures for PPML, with the best accuracy under the given budget for nonlinear (ReLU) and linear operators. We first present the building blocks in Seesaw that allow for additional linear operators and nonlinear output reuse (Section 3.1). Then we describe how to use pruning to control the computation costs of both nonlinear and linear operations (Section 3.2). Finally we summarize the end-to-end search strategy (Section 3.3).

3.1. Design Space

Seesaw uses two ways to compensate for the loss of nonlinear operators. Accordingly, two building blocks are added to its search space, as illustrated in Figure 1.

Figure 1a shows a **sampling block**, which substitutes a traditional Conv-ReLU block by enabling multiple parallel branches with various linear operators (Szegedy et al., 2015; 2016). The branches can be convolutions with different kernel sizes (*e.g.*, 1×1, 3×3, 5×5), depth-wise separable convolutions, dilated convolutions, pooling, or even a direct skip connection. These independent branches enhance the model representation capacity by extracting multiple and different scales of features. The sampling block is designed to significantly increase the expressivity with much more branches than Sphynx (Cho et al., 2022a) and CryptoNAS (Ghodsi et al., 2020) which only used up to four branches. The outputs of all branches have the same shape. They are weighted and combined using an element-wise ADD. The final ReLU may be pruned, *i.e.*, replaced with an Identity operator, to meet the overall ReLU budget.

Figure 1b shows an **aggregation block**, which aggregates the outputs of previous ReLU operators in the model. The goal of such aggregation is to maximally reuse the limited ReLU outputs remained in the pruned model, not only by the immediately next block, but also potentially by other succeeding blocks. Each of these previous ReLU outputs first passes a convolution kernel to reduce the resolution. Then an element-wise ADD operator weighted aggregates them before feeding to the final ReLU activation. Aggregating the ReLU outputs at different positions of the overall supermodel (Figure 2) helps prevent feature loss and overfitting (Szegedy et al., 2015; 2016), and is also another way to introduce extra nonlinearity.

We emphasize two key points in both building blocks. First, both blocks place the (possible) nonlinear ReLU *after an ADD operator*. In contrast to the CONCAT operators used in CryptoNAS (Ghodsi et al., 2020) and Sphynx (Cho et al., 2022a), ADD results in a smaller data size after aggregation, and thus reduces the amounts of nonlinear operations for the following ReLU. Actually, because Seesaw intentionally employs a large number of branches, using CONCAT would lead to significantly higher cost for each ReLU (by a factor equal to the branch count), and thus limit the total number of ReLU operators allowed in the model. We present a detailed comparison in Section 4.2 to demonstrate the benefit. Second, in both blocks, the branches are accumulated according to certain *learnable weights* β. These weights are automatically trained to determine the importance of different branches, so we can prune the non-critical branches to control the amount of extra linear operations that would increase the offline pre-processing cost (Section 3.2). We incorporate the training of these weights into the overall training process rather than separately determining them afterwards, as discussed later in Section 3.3.

Finally, the sampling blocks and the aggregation blocks are used to construct an over-parameterized **supermodel** in Seesaw (Figure 2). Each aggregation block is preceded by several sampling blocks (*i.e.*, m_i). The output of each ReLU is forwarded to all the aggregation blocks after it through residual connections (He et al., 2016). Several prior designs like CryptoNAS (Ghodsi et al., 2020) and Sphynx (Cho et al., 2022a) also used residual connections and mainly followed existing topologies like ResNet (He et al., 2016) and NASNet (Zoph et al., 2018). We emphasize that Seesaw uses much more residual connections beyond the original insecure network model, and for a completely different purpose of reusing ReLU outputs to increase expressivity.

3.2. Pruning Methods

The supermodel in Section 3.1 contains the following parameters: (1) The weight α_i to decide the nonlinear operator (ReLU or Identity) in the i-th sampling block. (2) The

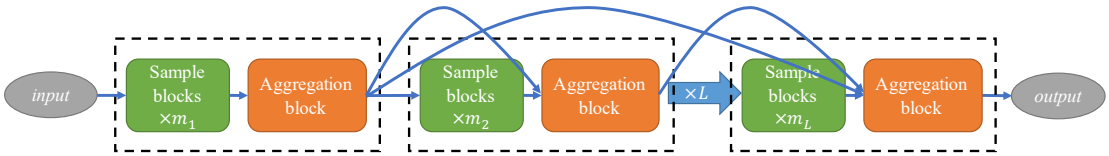


Figure 2. Supermodel architecture of Seesaw. There are L units in total. Each unit contains m_i sampling blocks followed by an aggregation block. The units are densely connected.

weight $\beta_{i,j}$ of the linear operator on the j-th branch in the i-th sampling/aggregation block. In order to limit the amounts of linear/nonlinear operations and thus their corresponding impacts on the execution time, Seesaw applies pruning to the α_i of sampling blocks, and the $\beta_{i,j}$ of sampling blocks (but not aggregation blocks, see Section 4.3).

Seesaw prunes the number of nonlinear ReLU operators in the model, by selectively enabling a subset of the sampling blocks to use ReLU, while the others use Identity operators. This is controlled by the weight α_i for the i-th sample block. Using Identity instead of ReLU would reduce the execution latency, but potentially sacrifices the model accuracy.

Seesaw also prunes the branches of linear operators in each sampling block. Unlike traditional NAS that usually keeps only one of the multiple branches (Liu et al., 2018; Cai et al., 2019; Wu et al., 2019), in PPML the linear operators are not the computation bottleneck. Thus Seesaw could retain more branches in each block to better compensate for the accuracy loss due to ReLU pruning. Nevertheless, pruning unimportant branches improves model generalization and prevents overfitting. More importantly, despite the negligible *online* computation cost, linear operations still affect the *offline* pre-processing overheads in PPML. Garimella et al. (2023) recently showed that the offline cost cannot be ignored in real-world PPML inference scenarios where user requests are continuously coming to the model provider's servers. Therefore, Seesaw needs to control its impact on the offline phase to avoid making it a new bottleneck.

Reducing and balancing online and offline latencies. Seesaw develops simple analytical models to estimate the execution latencies of the online and offline phases in the DELPHI framework. The offline latency is proportional to the amounts of both linear operations (where FHE is used to prepare their secret shares, Section 2.1) and nonlinear operations (where GC functions are created). The online latency is dominated by the nonlinear ReLU operations. Specifically, the latencies can be represented as

$$\tau_{\text{offline}} = \mathcal{E}_{\text{offline}}\left(\sum_i \alpha_i H_i W_i C_i, \sum_{i,j} \beta_{i,j}\text{FLOPs}_{i,j}\right) \quad (1)$$

$$\tau_{\text{online}} = \mathcal{E}_{\text{online}}\left(\sum_i \alpha_i H_i W_i C_i\right) \quad (2)$$

where H_i, W_i, C_i are the i-th block output shape, and $\text{FLOPs}_{i,j}$ is the FLOPs of the j-th branch in the i-th block. $\mathcal{E}_{\text{offline}}$ and $\mathcal{E}_{\text{online}}$ are linear models whose coefficients are the empirically measured per-operation latency costs. The α_i weights are always binarized to $\{0, 1\}$, but we omit such binarization for $\beta_{i,j}$ (see below and Section 3.3).

It is worth noting that, while the online latency should be minimized, it is not necessary to minimize the offline latency as long as it does not become the bottleneck, *i.e.*, the offline latency can be hidden by (no larger than) the online latency when the two phases are implemented as a processing pipeline (Garimella et al., 2023). Our search strategy in Section 3.3 leverages this fact when balancing between the model accuracy and online/offline latencies.

Pruning methodology. Similar to ProxylessNAS (Cai et al., 2019), during training, the α_i weights are binarized every epoch to ensure only one between ReLU and Identity is activated while searching. In contrast, we need the concrete values of $\beta_{i,j}$ in order to rank the importance of the branches in the sampling blocks and keep the top ones. We first prune the branches with very small weights (*i.e.*, less than an empirical threshold 0.0001) in each block. Then, if the offline latency still exceeds the online one, we continue to prune more branches in the ascending order of their weights across all sampling blocks, but skip the blocks with only a few (empirically set to 3) branches left. Our approach is able to achieve more stable performance compared to simple methods such as keeping a fixed number of branches in each block, as we will demonstrate in Section 4.3.

3.3. Search Strategy

The goal of Seesaw is to discover new PPML models with better balance between model accuracy and (online/offline) execution latency. As a result, we incorporate the latency cost into the loss function of Seesaw as below.

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda_1 \mathcal{L}_{\text{total}} + \lambda_2 \mathcal{L}_{\text{balance}} \quad (3)$$

where $\mathcal{L}_{\text{CE}}$ is the original cross-entropy loss, $\mathcal{L}_{\text{total}}$ controls the total latency cost of both online and offline processing, and $\mathcal{L}_{\text{balance}}$ helps balance the two phases (Section 3.2). With

the latency target τ_{target}, the latter two are defined as

$$\mathcal{L}_{\text{total}} = \left| \frac{\tau_{\text{offline}} + \tau_{\text{online}} - \tau_{\text{target}}}{\tau_{\text{target}}} \right| \quad (4)$$

$$\mathcal{L}_{\text{balance}} = \left| \frac{\tau_{\text{offline}} - \tau_{\text{online}}}{\tau_{\text{online}}} \right| \quad (5)$$

Both $\mathcal{L}_{\text{total}}$ and $\mathcal{L}_{\text{balance}}$ help guide the search process to choose proper pruning on α_i and $\beta_{i,j}$ in Equations (1) and (2), to strike a balance between latency and accuracy.

For the network architecture search strategy, traditional NAS typically constructs an over-parameterized supermodel encompassing all building blocks and potential branches in the search space. This supermodel contains numerous architecture parameters (*e.g.*, for each branch and for each block) that must be first sampled to generate a specific network to train (Liu et al., 2018; Cai et al., 2019). The search space from which network architectures are sampled is too large, making the training converge slowly. Some approaches try to directly train on the dataset, then optimize via a specific search algorithm, and finally do retraining (Zoph et al., 2018; Tan et al., 2019). This process can still be computationally intensive and time-consuming.

Seesaw uses a novel search strategy, which only includes the existence of nonlinear operators (*i.e.*, α_i) in the search space, and treats the branch weights for the large number of linear operators (*i.e.*, $\beta_{i,j}$) similarly to the model weights and to be updated during training without extra sampling. This greatly reduces the search space, accelerating the convergence when searching the best network architectures.

Algorithm 1 Seesaw network architecture search

Input: training dataset D_T, validation dataset D_V, target latency τ_{target}
Output: optimized network architectures

1: **while** not converged **do**
2: **if** epoch $>$ # warm-up epochs **then**
3: sample (X_V, Y_V) from D_V
4: M = nas_modules.sample()
5: $\mathcal{L} = \text{CE}(\text{M}(X_V), Y_V) + \lambda_1 \mathcal{L}_{\text{total}} + \lambda_2 \mathcal{L}_{\text{balance}}$
6: update(nas_modules, $\mathcal{L}$)
7: **end if**
8: sample (X_T, Y_T) from D_T
9: M = nas_modules.sample()
10: $\mathcal{L}_{\text{CE}} = \text{CE}(\text{M}(X_T), Y_T)$
11: update(M, $\mathcal{L}_{\text{CE}}$)
12: **end while**

Algorithm 1 shows the pseudocode of the Seesaw search algorithm. Seesaw takes as input the training dataset D_T, the validation dataset D_V, and the target latency τ_{target}. It trains the supermodel and searches for the network architecture iteratively in a continuous loop until converged. In each iteration, it samples a network architecture from the search space (*i.e.*, sample α_i values at Line 9), and uses the training dataset to train the network weights as well as the branch weights $\beta_{i,j}$ in the sampled model (Lines 8 to 11). After a certain number of warm-up training epochs, it starts to train the architecture parameters, *i.e.*, the NAS modules (Lines 2 to 7). The NAS modules are sampled to determine α_i, *i.e.*, the existence of each ReLU operator (Line 4). We use the overall loss $\mathcal{L}$ from Equation (3) to update the NAS modules (Lines 5 and 6). The network weight parameters are frozen during this process. The use of the validation set enhances the robustness of the architecture. After convergence, the optimized network architectures can be derived based on the trained supermodel.

4. Evaluation

We compare Seesaw with several previous PPML methods, including DELPHI (Mishra et al., 2020), Sphynx (Cho et al., 2022a), SNL (Cho et al., 2022b), SENet (Kundu et al., 2023a), as well as unmodified baseline models. The baseline models are ResNet-18 and ResNet-34 (He et al., 2016), with CIFAR100 (Krizhevsky & Hinton, 2009) and ImageNet (Deng et al., 2009) as the datasets. We leverage the DELPHI framework to perform real performance experiments, with the optimizations in Garimella et al. (2023) for offline pre-processing. We set 100 epochs for searching and 150 epochs for retraining, with decreasing learning rates from 0.5 to 0.0005. λ_1 and λ_2 are set to 0.001 and 0.1, respectively.

Seesaw does not incur significant search cost compared to existing methods. Specifically, with NVIDIA RTX 3090 GPUs, the complete architecture search process of Seesaw in Algorithm 1 takes approximately 10 GPU hours on CIFAR100, and 45 GPU hours on ImageNet. Such a search cost is moderate, and even more efficient than some previous PPML NAS methods. For example, Sphynx, which also used NAS, required 44 GPU hours on CIFAR100.

4.1. Comparison with State-of-the-Art

Figure 3 shows the comparison on ImageNet between our Seesaw and state-of-the-art PPML methods. We illustrate the accuracy vs. latency tradeoff curves of various methods in two ways: (1) online processing time only, (2) end-to-end offline and online time. The results clearly demonstrate the efficiency of Seesaw, in terms of the Pareto optimal frontiers between the classification accuracy and the runtime latency. The ability to achieve higher accuracy with lower latency makes Seesaw a highly efficient and promising approach for PPML inference.

Specifically, for the online latency comparison (Figure 3 left), if we look at the same accuracy level of 71% for

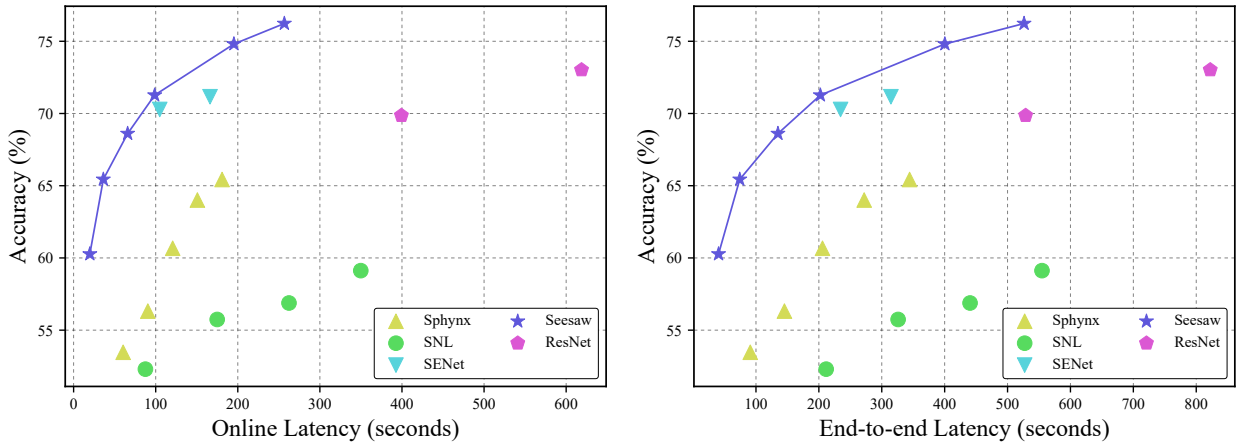


Figure 3. Accuracy vs. online latency (left) and end-to-end online + offline latency (right) results on ImageNet.

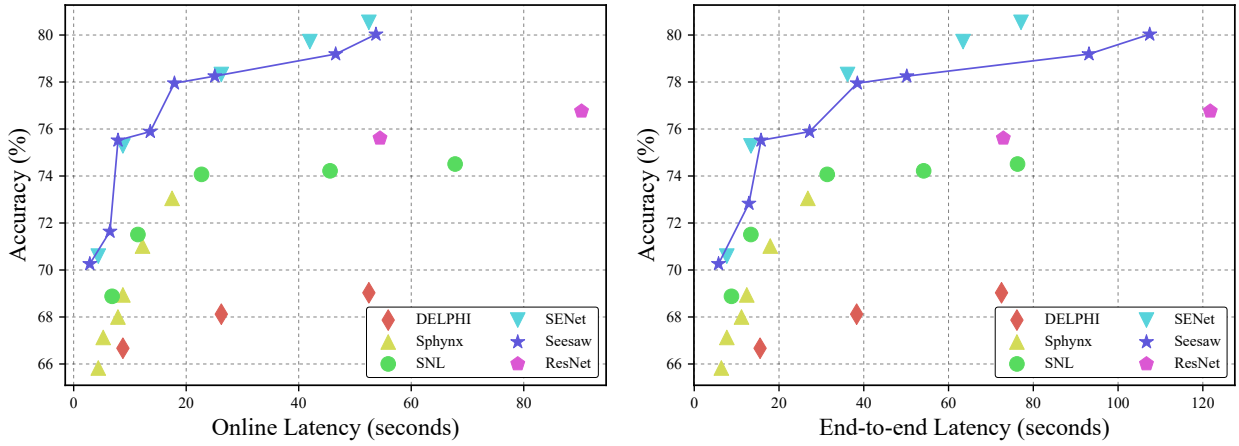


Figure 4. Accuracy vs. online latency (left) and end-to-end online + offline latency (right) results on CIFAR100.

example, Seesaw only needs about 100 seconds, which is $1.68\times$ faster than the next best design SENet. On the other hand, when doing an iso-latency comparison at 190 seconds, Seesaw improves the accuracy to 74.81%, which is 3.65% better than SENet at 170 seconds and 9.38% better than Sphynx at 180 seconds. Even when including the higher offline cost (Figure 3 right), at 71% accuracy Seesaw is still $1.55\times$ faster than SENet. Compared with the original ResNet, Seesaw gets about $4\times$ acceleration with similar accuracies.

Note that when the ReLU budget is abundant, Seesaw can even exceed the accuracy of the original insecure ResNet models. This is expected because Seesaw uses more linear operators. In the insecure scenario, such accuracy improvements come at the cost of longer inference latencies. However in PPML, the online latency is dominated by ReLU, for which Seesaw has similar or fewer operators.

Figure 4 presents the same comparison on the CIFAR100 dataset. At low ReLU budgets, Seesaw outperforms SENet, *i.e.*, $1.53\times$ faster at iso-accuracy of 70%, and 0.25% higher accuracy at iso-latency of 8 seconds. When the ReLU budget increases, the improvements over SENet become relatively small, and sometimes worse at high ReLU budgets. This is because SENet applies more fine-grained pixel-level ReLU pruning, which reduces the accuracy loss but requires more complex search and training methods. We expect similar pixel-level pruning can also be applied on top of Seesaw and further increase its performance. As a preliminary exploration, we apply pixel-level pruning based on SNL (Cho et al., 2022b) to the models discovered by Seesaw. The results in Table 1 show that with pixel-level pruning, the Seesaw models outperform the baseline SENet models, demonstrating the effectiveness of combining Seesaw with pixel-level pruning. This highlights the robustness and versatility of Seesaw, which can effectively leverage various orthogonal techniques, such as the fine-grained pruning, as post-processing steps to further improve efficiency.

Table 1. Comparison between SENet and Seesaw + SNL on CIFAR100.

ReLU Budget	Method	Accuracy
25K	SENet	70.59%
	Seesaw + SNL	71.42%
50K	SENet	75.28%
	Seesaw + SNL	76.02%

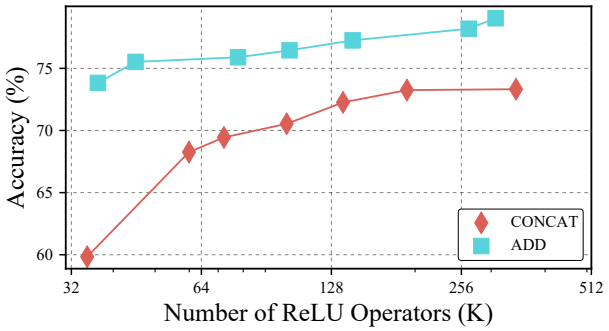


Figure 5. Comparison between the ADD and CONCAT operators on CIFAR100.

Table 2. Comparison between different pruning methods at low and high ReLU budgets on CIFAR100.

ReLU Budget	Pruning Method	End-to-End Latency (seconds)	Accuracy
36K	All	78.57	72.63%
	Fixed-11	20.98	72.53%
	Ours	15.13	72.32%
78K	All	91.66	74.93%
	Fixed-11	58.39	75.25%
	Ours	27.22	75.89%
110K	All	100.11	76.04%
	Fixed-11	39.64	75.23%
	Ours	45.66	77.26%
170K	All	192.08	76.92%
	Fixed-11	95.92	77.88%
	Ours	50.15	78.25%
300K	All	222.07	77.24%
	Fixed-11	132.27	79.33%
	Ours	107.47	80.03%

4.2. Ablation Study: ADD vs. CONCAT

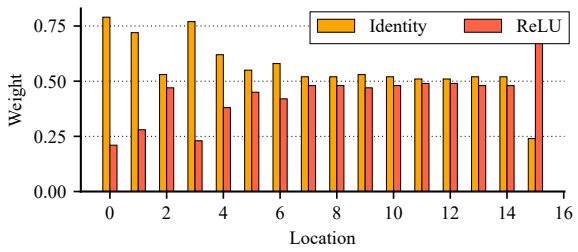
We compare the performance of ADD and CONCAT operators in Figure 5. We design another sampling block that is similar to Figure 1a but uses CONCAT instead of ADD. We then apply the same Seesaw search algorithm to find the best network architectures under different ReLU budgets and train the new models. For a fair comparison, we use the same ReLU budgets for the ADD- and CONCAT-based models. From the figure we see that, the CONCAT-based models can achieve good accuracies, but still not as high as the ADD-based models, exhibiting a noticeable gap of 7.0% on average. The accuracy difference is particularly significant when the ReLU budget is tight. Essentially, using ADD allows for more linear operators and thus higher expressivity *without* consuming extra nonlinear operators. These findings highlight the importance of carefully designing the linear combination mechanism within the sampling blocks of Seesaw.

4.3. Ablation Study: Pruning Methods

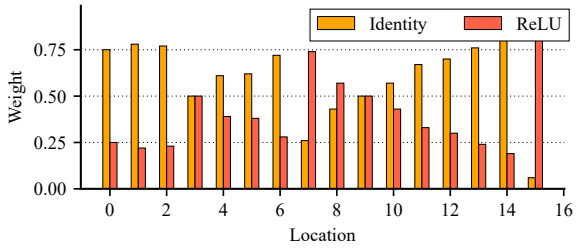
Section 3.2 introduces how to prune linear operator branches in each sampling block. We initially set 27 branches of different linear operators in every sampling block. After learning their weights $\beta_{i,j}$, we prune the branches until the offline latency decreases to the acceptable level. Now we evaluate its effectiveness by comparing with several other pruning methods, *e.g.*, keeping all branches without pruning (**All**), and keeping a fixed number (11) of branches with the highest weights in each block (**Fixed-11**).

Table 2 shows the end-to-end latencies and the classification accuracy numbers of different pruning methods under several ReLU budgets from 36K to 300K. In all cases, **All** only achieves slightly better or even worse accuracies than the two pruned models, despite the $2\times$ to $5\times$ higher latencies. This demonstrates the necessity of pruning, both for avoiding overfitting and for constraining execution time. Between the two pruning methods, specifying a fixed number of branches to keep cannot effectively adapt to diverse ReLU budgets. At a medium budget of 110K ReLU operations, compared with our method in Seesaw, **Fixed-11** is only 15% faster but has a noticeable 2.03% accuracy degradation, meaning that it does not keep enough linear operation branches to recover the accuracy. We also observe similar trends at higher ReLU budgets of 170K and 300K, where **Fixed-11** reduces the latencies but consistently underperforms in terms of accuracy compared to our pruning method. On the other hand, when the budget is low at 36K, **Fixed-11** causes 39% slowdown but only gains a minor 0.21% accuracy increase, which implies too many unnecessary branches are kept in the model.

The above results demonstrate that our pruning approach in Seesaw could adapt effectively to diverse ReLU budgets, pruning just enough branches to meet the latency goals while maintaining the best accuracy level. In contrast, keeping a fixed number of branches would lead to either accuracy degradation or unnecessary computational overheads.



(a) At 36K ReLU budget.



(b) At 110K ReLU budget.

Figure 6. Nonlinear operator weights of sampling blocks at different locations on CIFAR100. The one with a larger weight between ReLU and Identity will be used.

4.4. Network Architecture Analysis

Finally, we illustrate the distribution of the ReLU operators in the optimized network architectures discovered by Seesaw. Figure 6 shows the corresponding pre-binarization weight values for ReLU and Identity at different sampling blocks in two networks with different ReLU budgets of 36K and 110K. The sampling blocks at the latter stage of the network tend to have higher ReLU weights and would keep the ReLU operators. This observation aligns with the ReLU sensitivity study in SENet (Kundu et al., 2023a). For example, on the premise of retaining the ReLU of aggregation blocks, Figure 6a with a small 36K ReLU budget only keeps the last nonlinear operator. However, Seesaw can also retain some earlier nonlinear operators if the ReLU budget allows, in order to boost the accuracy. For example, Figure 6b with a 110K budget also preserves the ReLU operators in the middle at positions 7 and 8.

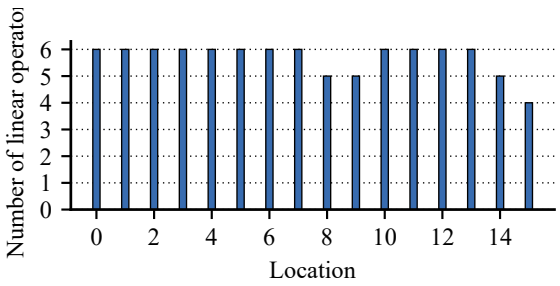


Figure 7. Numbers of linear operation branches of sampling blocks at different locations on CIFAR100, under a ReLU budget of 110K.

In contrast, for the distribution of the linear operations, Figure 7 shows that the number of branches in a sampling block is likely higher towards the early part of the network, reflecting that more linear operators are kept at the beginning of the network in order to compensate the accuracy loss due to the reduced ReLU amounts in the latter stages.

Combining the above two trends, we get an interesting observation. *An optimized PPML network architecture needs to preserve sufficient nonlinearity in the latter blocks of the model, while at the earlier stage, it can instead use more linear computations to increase the representation capacity.* The two patterns compensate very well, once again validating the design principle of Seesaw.

5. Conclusions

In this paper, we present Seesaw, a neural network structure search scheme that is tailored to private machine learning inference. Seesaw compensates for the negative accuracy impact of reducing expensive nonlinear operators through adding more linear computations and reusing existing nonlinear results. It incorporates novel pruning and search approaches to efficiently determine the optimized amounts of extra computations and data reuse. Our evaluation shows that Seesaw achieves higher accuracy with fewer nonlinear operations compared to previous proposals. Seesaw paves the way for deploying accurate and efficient privacy-preserving machine learning models in real-world scenarios, where computational constraints and privacy concerns are both critical.

Acknowledgments

The authors thank the anonymous reviewers for their valuable suggestions, and the Tsinghua IDEAL group members for constructive discussion. This work was supported by the National Natural Science Foundation of China (62072262) and Shanghai Artificial Intelligence Lab.

Impact Statement

This paper presents work whose goal is to advance the field of machine learning. There could be many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Alibaba Cloud. https://ai.aliyun.com/. Accessed: August 2021.

Deep Learning on AWS. https://aws.amazon.com/deep-learning/. Accessed: August 2021.

Machine Learning Service, Microsoft Azure. https://azure.microsoft.com/en-us/services/machine-learning/. Accessed: August 2021.

Deep Learning VM, Google Cloud. https://cloud.google.com/deep-learning-vm/. Accessed: August 2021.

Baidu AI Cloud. https://intl.cloud.baidu.com/. Accessed: August 2021.

ChatGPT. https://openai.com. Accessed: August 2023.

Arora, S., Cohen, N., and Hazan, E. On the Optimization of Deep Networks: Implicit Acceleration by Overparameterization. In *35th International Conference on Machine Learning (ICML)*, pp. 244–253, 2018.

Badawi, A. A., Chao, J., Lin, J., Mun, C. F., Jie, S. J., Tan, B. H. M., Nan, X., Aung, K. M. M., and Chandrasekhar, V. R. Towards the AlexNet Moment for Homomorphic Encryption: HCNN, the First Homomorphic CNN on Encrypted Data with GPUs. *arXiv preprint arXiv:1811.00778*, 2018.

Brutzkus, A., Gilad-Bachrach, R., and Elisha, O. Low Latency Privacy Preserving Inference. In *36th International Conference on Machine Learning (ICML)*, pp. 812–821, 2019.

Cai, H., Zhu, L., and Han, S. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *7th International Conference on Learning Representations (ICLR)*, 2019.

Chandran, N., Gupta, D., Obbattu, S. L. B., and Shah, A. SIMC: ML Inference Secure Against Malicious Clients at Semi-Honest Cost. In *31st USENIX Security Symposium*, pp. 1361–1378, 2022.

Chen, G., Chen, S., Xiao, Y., Zhang, Y., Lin, Z., and Lai, T. H. SgxPectre: Stealing Intel Secrets from SGX Enclaves via Speculative Execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 142–157, 2019.

Cho, M., Ghodsi, Z., Reagen, B., Garg, S., and Hegde, C. Sphynx: A Deep Neural Network Design for Private Inference. *IEEE Security & Privacy*, 20(5):22–34, 2022a.

Cho, M., Joshi, A., Reagen, B., Garg, S., and Hegde, C. Selective Network Linearization for Efficient Private Inference. In *39th International Conference on Machine Learning (ICML)*, pp. 3947–3961, 2022b.

Chou, E., Beal, J., Levy, D., Yeung, S., Haque, A., and Fei-Fei, L. Faster CryptoNets: Leveraging Sparsity for Real-World Encrypted Inference. *arXiv preprint arXiv:1811.09953*, 2018.

Damgård, I., Pastro, V., Smart, N., and Zakarias, S. Multiparty Computation from Somewhat Homomorphic Encryption. In *32nd Annual Cryptology Conference (CRYPTO)*, pp. 643–662, 2012.

Deng, J., Dong, W., Socher, R., Li, L. J., and Li, F. F. ImageNet: A Large-Scale Hierarchical Image Database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255, 2009.

Dowlin, N., Gilad-Bachrach, R., Laine, K., Lauter, K., Naehrig, M., and Wernsing, J. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *33rd International Conference on Machine Learning (ICML)*, pp. 201–210, 2016.

Garimella, K., Ghodsi, Z., Jha, N. K., Garg, S., and Reagen, B. Characterizing and Optimizing End-to-End Systems for Private Inference. In *28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Volume 3*, pp. 89–104, 2023.

Gentry, C. Fully Homomorphic Encryption Using Ideal Lattices. In *41st Annual ACM Symposium on Theory of Computing (STOC)*, pp. 169–178, 2009.

Ghodsi, Z., Veldanda, A. K., Reagen, B., and Garg, S. CryptoNAS: Private Inference on a ReLU budget. *Advances in Neural Information Processing Systems (NeurIPS)*, 33: 16961–16971, 2020.

Ghodsi, Z., Jha, N. K., Reagen, B., and Garg, S. Circa: Stochastic ReLUs for Private Deep Learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 34: 2241–2252, 2021.

Guo, S., Alvarez, J. M., and Salzmann, M. ExpandNets: Linear Over-Parameterization to Train Compact Convolutional Networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:1298–1310, 2020.

He, K., Zhang, X., Ren, S., and Sun, J. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Huang, Z., Lu, W.-j., Hong, C., and Ding, J. Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference. In *31st USENIX Security Symposium*, pp. 809–826, 2022.

Hunt, T., Song, C., Shokri, R., Shmatikov, V., and Witchel, E. Chiron: Privacy-Preserving Machine Learning as a Service. *arXiv preprint arXiv:1803.05961*, 2018.

Hynes, N., Cheng, R., and Song, D. Efficient Deep Learning on Multi-Source Private Data. *arXiv preprint arXiv:1807.06689*, 2018.

Jha, N. K., Ghodsi, Z., Garg, S., and Reagen, B. DeepReDuce: ReLU Reduction for Fast Private Inference. In *38th International Conference on Machine Learning (ICML)*, pp. 4839–4849, 2021.

Juvekar, C., Vaikuntanathan, V., and Chandrakasan, A. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium*, pp. 1651–1669, 2018.

Kim, K., Kim, C. H., Rhee, J. J., Yu, X., Chen, H., Tian, D., and Lee, B. Vessels: Efficient and Scalable Deep Learning Prediction on Trusted Processors. In *ACM Symposium on Cloud Computing (SoCC)*, pp. 462–476, 2020.

Krizhevsky, A. and Hinton, G. Learning Multiple Layers of Features from Tiny Images. Technical Report, University of Toronto, 2009.

Kundu, S., Lu, S., Zhang, Y., Liu, J., and Beerel, P. A. Learning to Linearize Deep Neural Networks for Secure and Efficient Private Inference. *arXiv preprint arXiv:2301.09254*, 2023a.

Kundu, S., Zhang, Y., Chen, D., and Beerel, P. A. Making Models Shallow Again: Jointly Learning to Reduce Non-Linearity and Depth for Latency-Efficient Private Inference. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 4684–4688, 2023b.

Kunkel, R., Quoc, D. L., Gregor, F., Arnautov, S., Bhatotia, P., and Fetzer, C. TensorSCONE: A Secure TensorFlow Framework using Intel SGX. *arXiv preprint arXiv:1902.04413*, 2019.

LeCun, Y., Bengio, Y., and Hinton, G. Deep Learning. *Nature*, 521(7553):436–444, 2015.

Li, F., Li, X., and Gao, M. Secure MLaaS with Temper: Trusted and Efficient Model Partitioning and Enclave Reuse. In *39th Annual Computer Security Applications Conference (ACSAC)*, pp. 621–635, 2023.

Li, Y., Zeng, D., Gu, L., Chen, Q., Guo, S., Zomaya, A., and Guo, M. Lasagna: Accelerating Secure Deep Learning Inference in SGX-Enabled Edge Cloud. In *ACM Symposium on Cloud Computing (SoCC)*, pp. 533–545, 2021.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable Architecture Search. *arXiv preprint arXiv:1806.09055*, 2018.

Liu, J., Juuti, M., Lu, Y., and Asokan, N. Oblivious Neural Network Predictions via MiniONN Transformations. In *2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 619–631, 2017.

Lou, Q., Shen, Y., Jin, H., and Jiang, L. SAFENet: A Secure, Accurate and Fast Neural Network Inference. In *9th International Conference on Learning Representations (ICLR)*, 2021.

Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., and Popa, R. A. DELPHI: A Cryptographic Inference Service for Neural Networks. In *29th USENIX Security Symposium*, pp. 2505–2522, 2020.

Mohassel, P. and Zhang, Y. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *2017 IEEE Symposium on Security and Privacy (S&P)*, pp. 19–38. IEEE, 2017.

Ng, L. K. and Chow, S. S. GForce: GPU-Friendly Oblivious and Rapid Neural Network Inference. In *30th USENIX Security Symposium*, pp. 2147–2164, 2021.

Riazi, M. S., Weinert, C., Tkachenko, O., Songhori, E. M., Schneider, T., and Koushanfar, F. Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications. In *2018 ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, pp. 707–721, 2018.

Rouhani, B. D., Riazi, M. S., and Koushanfar, F. DeepSecure: Scalable Provably-Secure Deep Learning. In *55th Annual Design Automation Conference (DAC)*, pp. 1–6, 2018.

Sanyal, A., Kusner, M., Gascon, A., and Kanade, V. TAPAS: Tricks to Accelerate (Encrypted) Prediction as a Service. In *35th International Conference on Machine Learning (ICML)*, pp. 4490–4499, 2018.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going Deeper with Convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the Inception Architecture for Computer Vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, 2016.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2820–2828, 2019.

Tramer, F. and Boneh, D. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. In *7th International Conference on Learning Representations (ICLR)*, 2019.

Wang, J., Cheng, Y., Li, Q., and Jiang, Y. Interface-Based Side Channel Attack Against Intel SGX. *arXiv preprint arXiv:1811.05378*, 2018.

Wei, K., Li, J., Ding, M., Ma, C., Yang, H. H., Farokhi, F., Jin, S., Quek, T. Q. S., and Vincent Poor, H. Federated Learning With Differential Privacy: Algorithms and Performance Analysis. *IEEE Transactions on Information Forensics and Security*, 15:3454–3469, 2020.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10734–10742, 2019.

Yao, A. C.-C. How to Generate and Exchange Secrets. In *27th Annual Symposium on Foundations of Computer Science (SFCS)*, pp. 162–167, 1986.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding Deep Learning (Still) Requires Rethinking Generalization. *Communications of the ACM*, 64(3):107–115, 2021.

Zhang, T., He, Z., and Lee, R. B. Privacy-Preserving Machine Learning Through Data Obfuscation. *arXiv preprint arXiv:1807.01860*, 2018.

Zhang, Y., Chen, D., Kundu, S., Liu, H., Peng, R., and Beerel, P. A. C2PI: An Efficient Crypto-Clear Two-Party Neural Network Private Inference. *arXiv preprint arXiv:2304.13266*, 2023.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning Transferable Architectures for Scalable Image Recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8697–8710, 2018.