ECAI 2020
G.D. Giacomo et al. (Eds.)
© 2020 The authors and IOS Press.
This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License 4.0 (CC BY-NC 4.0).
doi:10.3233/FAIA200288

NASS: Optimizing Secure Inference via Neural Architecture Search

Song Bian[1] and **Weiwen Jiang**[2] and **Qing Lu**[2] and **Yiyu Shi**[2] and **Takashi Sato**[1]

Abstract. Due to increasing privacy concerns, neural network (NN) based secure inference (SI) schemes that simultaneously hide the client inputs and server models attract major research interests. While existing works focused on developing secure protocols for NN-based SI, in this work, we take a different approach. We propose NASS, an integrated framework to search for tailored NN architectures designed specifically for SI. In particular, we propose to model cryptographic protocols as design elements with associated reward functions. The characterized models are then adopted in a joint optimization with predicted hyperparameters in identifying the best NN architectures that balance prediction accuracy and execution efficiency. In the experiment, it is demonstrated that we can achieve the best of both worlds by using NASS, where the prediction accuracy can be improved from 81.6% to 84.6%, while the inference runtime is reduced by 2x and communication bandwidth by 1.9x on the CIFAR-10 dataset.

1 Introduction

With serious concerns growing over the security risks of property stealing [15] and private information leakage [27] related to machine learning as a service schemes, the study of the security properties of both neural network (NN) training [20] and inference [16] is becoming one of the most important fields of study across the disciplines. In particular, a secure inference (SI) scheme refers to the situation where Bob as a client wants to hide his inputs to Alice, the NN service provider. Meanwhile, Alice also needs to protect her trained network model, as such a trained model is extremely valuable due to the costly dataset preparation and lengthy training processes.

While a number of protocols have been proposed for both secure inference and training on neural networks [16,20,22,25], the general approach of existing works is to find the equivalent NN operations (e.g., matrix-vector product, activation functions) in the secure domain (e.g., using garbled circuits or homomorphic encryption), and instantiate the secure protocols accordingly. In other words, security is not an integral part of the proposed protocol, but rather an added feature with (in many cases, serious) performance penalties.

Recent advances in the secure machine learning field have suggested the possibility of formulating the secure protocols as a design automation problem. For example, in [3], authors proposed a framework that automatically instantiate parameters for homomorphic encryption (HE) schemes. Likewise, a line of research efforts [7,14,16] have explored how to optimize HE parameters and packing capabilities to improve the efficiency of secure computations, especially for neural network based protocols. In all of the existing works, secure primitives are designed to maximize efficiency of a pre-defined neural architecture (in fact, many of the existing works use the same manually designed architecture).

We argue that the existing design techniques based on fixed neural architectures lead to unsatisfactory solutions, as the efficiency of SI (in terms of the inference time and network bandwidth) are significantly affected by the architectures. The performance non-linearity of cryptographic primitives are demonstrated through Fig. 1, where the computational and communication costs of SI are plotted with respect to quantization factors of some neural architecture. From Fig. 1, we can see that for certain quantization intervals (e.g., 14-bit to 15-bit), the inference time is doubled, while for other intervals (e.g., from 2-bit up to 15 bits), the inference time remains unchanged. This non-linear performance curve is primarily due to the underlying primitive (in this case a packed homomorphic encryption (PAHE) scheme) that is constrained by its cryptographic parameters. Our conclusion here is that, in order to obtain better design trade-offs, a joint exploration considering both secure primitives and neural architectures is required to push forward the Pareto frontiers of the efficiency and prediction accuracy of NN-based SI.

In this work, we propose NASS, a novel Neural Architecture Search framework for Secure inference, where the optimization of cryptographic primitives and NN prediction accuracy are integrated. To the best of our knowledge, we are the first to take a *synthetic* approach to improve both the accuracy and the efficiency of SI on NN. In NASS, the process of finding the best SI scheme is formulated as predicting the most rewarding neural architecture, where the reward is derived from the accuracy and efficiency statistics. A system optimizer based on reinforcement learning is used to take feedback from the rewards to generate new architectures, acting as a neural architecture search (NAS) engine. Our main contributions are summarized as follows.

- **Synthesizing Secure Architectures**: To the best of our knowledge, NASS represents the first work to search for neural architectures optimized in secure applications with multiple cryptographic building blocks. In NASS, cryptographic primitives are modeled as design elements, and se-

[1] Kyoto University, Kyoto, Japan, email: paper@easter.kuee.kyoto-u.ac.jp
[2] University of Notre Dame, Indiana, USA, email:{wjiang2, qlu2, yshi4}@nd.edu

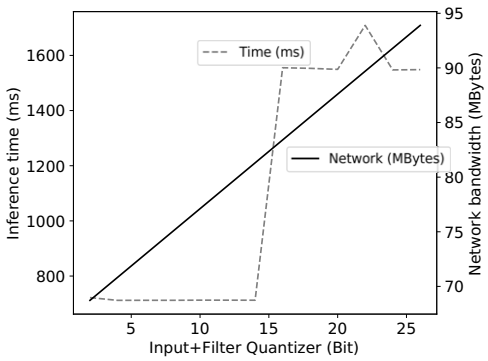

Figure 1. The relationship between neural architectures and the performance of secure inference, using a network with 1 Conv, 1 Relu, and 1 FC as example, where one more bit may double the inference time.

cure computations with these elements become abstract operators that can be automatically synthesized by the optimization engines.

- **Optimizing HE Parameters**: While existing works have already treated the instantiation of HE parameters as a design problem and proposed some solutions [3], we point out that these solutions are not adequate. In particular, we identify an optimization dilemma in learning with errors (LWE) based HE parameter instantiation, and observe that this optimization problem is (computationally) rather difficult to solve, especially for NAS-based optimization with fast turnaround time.
- **A Thorough Architectural Search for SI**: By conducting extensive architectural search, it is demonstrated that the performance of SI can be reduced while improving the prediction accuracy. We achieve a prediction accuracy of 84.6% on the CIFAR-10 dataset, while reducing 2x computational time and 1.9x network bandwidth, compared to the best known SI scheme [16] with a prediction accuracy of only 81.6%.

The rest of this paper is organized as follows. First, in Section 2, basics on PAHE, secure inference, and NAS are discussed. Second, the NASS framework is outlined in Sections 3 and 4, where we detail how security and parameter analyses can be systematically performed, along with the design of reward functions for the integration with the NAS engine. Next, the output architectures of NASS along with performance statistics are demonstrated in Section 5. Finally, our work is summarized in Section 6.

2 Preliminaries

2.1 Cryptographic Building Blocks

In this work, we mainly consider the optimization involving two types of cryptographic primitives, packed additive homomorphic encryption (PAHE) based on the ring learning with error (RLWE) problem [4–6, 10], and garbled circuits (GC) [34]. In what follows, we provide a high-level abstraction of each individual primitive.

PAHE: A PAHE is a cryptosystem, where the encryption (Enc) and decryption (Dec) functions act as group (additive) homomorphisms between the plaintext and ciphertext spaces. Except for the normal Enc and Dec, a PAHE scheme is equipped with the following three abstract operators. We use $[\mathbf{x}]$ to denote the encrypted ciphertext of $\mathbf{x} \in \mathbb{Z}^n$, and $n \in \mathbb{Z}$ here is some lattice dimension.

- Homomorphic addition ($\boxplus$): for $\mathbf{x}, \mathbf{y} \in \mathbb{Z}^n$, $\mathsf{Dec}([\mathbf{x}] \boxplus [\mathbf{y}]) = \mathbf{x} + \mathbf{y}$.
- Homomorphic Hadamard product ($\boxdot$): for $\mathbf{x}, \mathbf{y} \in \mathbb{Z}^n$, $\mathsf{Dec}([\mathbf{x}] \boxdot \mathbf{y}) = \mathbf{x} \circ \mathbf{y}$, where $\circ$ is the element-wise multiplication operator.
- Homomorphic rotation (rot): for $\mathbf{x} \in \mathbb{Z}^n$, let $\mathbf{x} = (x_0, x_1, \cdots, x_{n-1})$, $\mathsf{rot}([\mathbf{x}], k) = (x_k, x_{k+1}, \cdots, x_{n-1}, x_0, \cdots, x_{k-1})$ for $k \in \{0, \cdots, n-1\}$.

GC: GC can be considered as a more general form of HE. In particular, the circuit garbler, Alice, "encrypts" some function f along with her input x to Bob, the circuit evaluator. Bob evaluates $f(x, y)$ using his encrypted input y that is received from Alice obliviously, and obtains the encrypted outputs. Alice and Bob jointly "decrypt" the output of the function $f(x, y)$ and one of the two parties learn the result.

2.2 Homomorphic Evaluation Errors in RLWE-based PAHE

In this work, we omit details on the implementation of RLWE-based PAHE schemes, such as BFV [4, 10], BGV [5], and CKKS [6]. However, for all of the above RLWE-based PAHE schemes (and most RLWE-based cryptosystems), the ciphertext output from the encryption function of the cryptosystem bares some intrinsic errors, which can be thought of an additive components to the ciphertext, i.e.,

$$\overline{\mathbf{c}} = \mathbf{c} + \mathbf{e}, \tag{1}$$

where $\mathbf{c}$ is the "errorless" ciphertext, and $\mathbf{e}$ the error (both are vectors in $\mathbb{Z}^n$ for some lattice dimension n). It is obvious that when we add two ciphertexts, $\overline{\mathbf{c}}_0 \boxplus \overline{\mathbf{c}}_1$, the error is also additively increased (i.e., $\mathbf{e}_{\text{sum}} = \mathbf{e}_0 + \mathbf{e}_1$). Similarly, homomorphic Hadamard product and rotation operations also increases the errors. Therefore, each level of homomorphic evaluation increases the error contained in the ciphertext, and when the size of the error become too large (i.e., too many levels of homomorphic evaluations), *some* ciphertexts will not be correctly deciphered. We emphasize the point that not all ciphertexts become undecipherable as the size of the error is randomly distributed, and this probabilistic behavior can be utilized to improve the efficiency of SI schemes.

2.3 Secure Neural Network Inference

While a number of pioneer works have already established the concept of secure inference and training with neural networks [20, 22, 25], it was not until recently that such protocols carried practical significance. For example, in [20], an inference with a single CIFAR-10 image takes more than 500 seconds to complete. Using the same neural architecture, the performance was improved to less than 13 seconds in one of the most recent arts on SI, Gazelle [16]. Unfortunately, 13 seconds per image inference is obviously still unsatisfactory, especially given the large amount of data exchange in real-world applications. Therefore, we adopt the Gazelle protocol

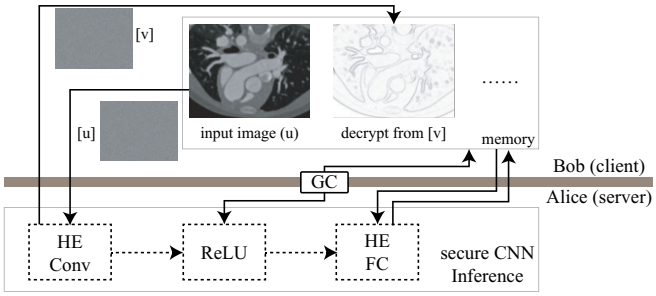

Figure 2. An example of the architecture in Gazelle with one Conv layers, one non-linear layers and one FC layer.

in this work, and take a system-level approach to improve its efficiency.

An overview of the Gazelle protocol is outlined in Fig. 2, where Alice wants to classify some input (e.g., image), and Bob holds the weights. The Gazelle protocol classifies all NN operations into two types of layers: i) linear layers, where the computations are efficiently carried out by PAHE-based cryptographic primitives, and ii) non-linear layers, where interactive protocols such as multiplication triples [2] or GC are employed.

Threat Model: The threat model in Gazelle and this work is that both Bob and Alice are semi-honest, in the sense that both parties follow the described protocol (e.g., encryption and decryption procedures in PAHE, GC), but want to learn as much information as possible from the other party. In particular, Alice wishes to gain knowledge on the trained model from Bob, and Bob is curious about the encrypted inputs from Alice.

2.4 Neural Architecture Search

Recently, Neural Architecture Search (NAS) has been consistently breaking the accuracy records in a variety of machine learning applications, such as image classification [36], image segmentation [19], video action recognition [23], and many more. NAS attracts major attentions mainly because it successfully eliminates the needs of human expertise and labor time in identifying high-accuracy neural architectures.

A typical NAS, such as that in [36], is composed of a controller and a trainer. The controller will iteratively predict (i.e., generate) neural architecture parameters, referred to as child networks. The child networks will be trained from scratch by the trainer on a held-out dataset to obtain the prediction accuracy. Then, the accuracy will be feedback to update the controller. Finally, after the number of child networks predicted by the controller exceed a predefined threshold, the search process will be terminated. The searched architecture with the highest accuracy is identified to be the output of the NAS engine.

Existing works have demonstrated that the automatically searched neural architectures can achieve close accuracy to the best human-invented architectures [36, 37]. In addition, we also identify multi-objective NAS techniques proposed under the context of field-programmable gate array (FPGA) and mobile platforms [9, 12, 13, 21, 29, 32, 33, 35]. However, without proper security performance measures, the identified architectures can have over-complex structures that render them useless in real-world cryptographic applications. In addition, as demonstrated in Fig. 1, cryptographic primitives generally have complicated performance trade-offs, and no existing works have demonstrated that a multi-objective NAS engine is able to learn such complex behaviors. Therefore, the main motivation of the NASS framework is to find accurate and efficient neural architectures for secure inference schemes.

3 NASS Framework

3.1 Problem Formulation and Challenges

In this paper, we aim to identify the most efficient secure neural network inference via neural architecture search. The problem is informally defined as follows: Given a specific dataset and a set of secure inference protocols, our objective is to automatically generate a quantized neural network architecture and the parameters for each of the cryptographic primitives, such that the reward of the resultant neural network after training can be maximized. Here, we define the reward to be a function of the prediction accuracy and performance statistics, including the inference time and network bandwidth.

To solve the above problem, several challenges need to be addressed from both the neural architecture search perspective and the secure protocol perspective. We list two main challenges as follows.

Challenge 1: There are missing links among neural architectural optimizations, quantization optimizations, and cryptographic protocol optimizations, resulting in the non-optimal solutions from existing works. This is based on our observation that all of the above optimizations are tightly cross-coupled; that is, the optimization in one direction (e.g., better prediction accuracy) can have positive or negative impact on the other directions (e.g., larger quantization level and higher secure inference time). Therefore, a framework that can jointly optimize neural architectures, quantizations, and performances of cryptographic primitives, is needed. In this work, we derive the NASS framework (Section 3.2 and 3.3) to fill the gap.

Challenge 2: To the best of our knowledge, there exists no efficient performance estimator for secure inference (SI) involving multiple cryptographic primitives. Since the NAS engine generates a large amount of intermediate architectures iteratively, without an automatic performance estimator, it is impossible to evaluate the performance statistics of such networks adopted in SI. In this paper, we make the contribution of developing efficient estimator engines (Section 4).

3.2 Overview of NASS

Fig. 3 illustrates an overview of the proposed NASS Framework with four components: ① ParmGen, ② Machine Learning (ML) Estimators, ③ Cryptographic Estimators, and ④ Controller. Specifically, component ① parameterizes the architecture and quantization, which identifies a unique neural architecture for the subsequent computations. Upon receiving the input architecture from component ①, Component ② trains and evaluates its accuracy, and component ③ optimizes the cryptographic primitives to estimates its performance. Finally, component ④ will control the optimization flow. All of

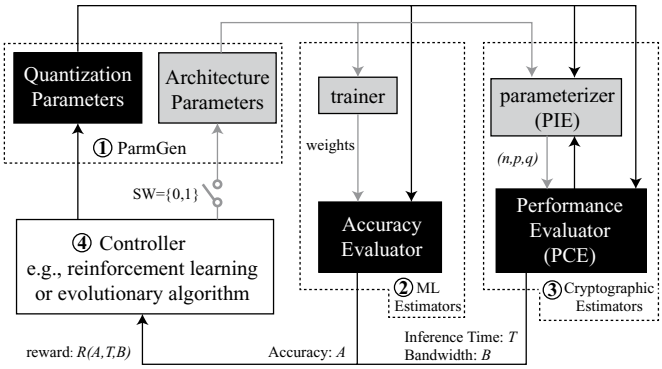

Figure 3. An overview on the proposed NASS framework.

the components collaboratively explore the parameter spaces of neural architecture, quantization, and cryptographic primitives to jointly optimize the accuracy, time, and bandwidth.

The NASS framework works in three steps. First, the controller generates a prediction on a quantized neural architecture (called child network), which will be formulated as LayerParms. Second, the child network will be evaluated by ML Estimators to generate prediction accuracy (A), and optimized in Cryptographic Estimators to provide the inference time (T), and bandwidth (B) feedback. Lastly, a reward signal is generated in terms of A, T, B, to update the controller. Details of each component will be introduced in Section 3.3.

In practice, the lengthy training process dominates the search time. In NASS, we add a switch SW before the architecture parameter (AP) subcomponent to dramatically reduce the number of training processes. This is based on the observation that the quality of quantization parameters for the same architectures can be evaluated using the same trained (floating-point) weights. If the switch is on ($SW = 1$), we will train the architecture from scratch to generate weights and obtain accuracy statistics in terms of quantization. Otherwise ($SW = 0$), we reuse the weights and apply the new quantization parameters to obtain the accuracy.

The switch can be controlled by using a predefined function. In this work, we demonstrate the exploration procedure using the following function:

$$SW = \begin{cases} 1 & Eps \ mod \ SW_N = 0 \\ 0 & Otherwise \end{cases} \tag{2}$$

where Eps is the episode index given to each of the child networks predicted by the controller, SW_N is a scalar to indicate the number of child network with the same architecture but different quantization to be explored.

3.3 NASS Framework Details

① **ParmGen.** The ParmGen block generates layer parameters for the subsequent computations. A neural architecture consists of a set of layers. According to the linearity of function for each layer, there are two types of layers: linear layer (e.g., convolution, fully connection), and non-linear layer (e.g., ReLU, pooling). Each layer can be specified as a set of parameters. Note that different types of layers have different parameters. We denote $\mathsf{LayerParms}_{\mathrm{L}}$ and $\mathsf{LayerParms}_{\mathrm{NL}}$ to represent parameters of linear layers and non-linear layers, respectively.

For *linear layers*, a set of layer parameter is denoted as $\mathsf{LayerParms}_{\mathrm{L}} = \{n_i, n_o, f_w, f_h, l_i, l_f, c_i, c_o\}$, where n_i and n_o represent the dimensions of feature maps (i.e., input data); f_w and f_h represent the dimensions of filters (i.e., weights), l_i and l_f indicate the data and weight quantizers; c_i and c_o stand for the number of input and output channels.

For *non-linear layers*, they do not contain weights, and therefore, there is no parameters for filter quantizations and dimensions. In addition, the number of channels will not be changed, and we only record the input channel number c_i in layer parameter $\mathsf{LayerParms}_{\mathrm{NL}}$. In consequence, we denote parameter sets for non-linear layers as $\mathsf{LayerParms}_{\mathrm{NL}} = \{n_i, n_o, l_i, l_o, c_i\}$.

A neural architecture can be represented by a collection of parameters for all layers, i.e., $\mathsf{NetworkParms} = \{\mathsf{LayerParms}_{\mathrm{L},k}, \mathsf{LayerParms}_{\mathrm{NL},k}\}$ where $k = 0, 1, \cdots$ represents the kth layer.

② **Machine Learning (ML) Estimator.** A machine learning estimator is composed of a trainer and an accuracy evaluator. According to the status of switch before AP in ①, the ML estimator will take different actions. When the switch is on (i.e., $SW = 1$), a new architecture will flow into the ML estimator, and it will be trained from scratch using floating points. Then, the accuracy evaluator will quantize the weights from the trainer according to the given quantization parameters to obtain the accuracy of the quantized neural network. When the switch is off (i.e., $SW = 0$), it indicates that the previous predicted architecture is applied with new quantization parameters. In this case, the ML estimator will not train the architecture. The weights from in the previous iteration are reused with new quantization parameters to obtain the prediction accuracy on the training dataset.

③ **Cryptographic Estimator.** A cryptographic estimator contains two sub-components: the parameter instantiation engine (PIE) and the performance characterization engine (PCE). These engines take input from LayerParms, and collaboratively instantiate parameters for the cryptographic parameters while evaluating their performance. In particular, the outputs of PIE are the cryptographic parameters. For example, for RLWE-based PAHE schemes (e.g., used for homomorphic matrix-vector product in linear layers and multiplication triples for square activation), the cryptographic parameters are (n, q, p). During performance characterization, PCE consults with PAHE and GC libraries to produce characterized scores for a single round of secure inference using the architecture specified in NetworkParms. Details on the implementations of PIE and PCE will be discussed in Section 4. Kindly note that, for some cryptographic protocols (e.g., GC used in implementing ReLU), the cryptographic parameters can be directly determined in terms of $\mathsf{LayerParms}_{\mathrm{NL}}$ without using a PIE.

④ **Controller.** The controller is a core component in the NASS framework. According to the output of the ML estimator (②) and the cryptographic estimator (③), the controller predicts a new NetworkParms which supposedly has higher accuracy, lower latency, and lower bandwidth requirement compared to the architecture predicted in the previous iteration.

The controller can be implemented by different techniques, such as the reinforcement learning or the evolutionary algorithms. However, in both cases, the key element for the controller design is the reward function. In this work, we employ

the reinforcement learning method in the controller whose interactions with the environment are modeled as a Markov decision process (MDP). The reward function is formulated as follows.

$$R(A, T, B) = A + A \cdot \xi(T, B), \tag{3}$$

where A is the prediction accuracy, and $\xi(T, B)$ is the performance score reported by the cryptographic estimators. The detailed definition of ξ can be found in Eq. (8). After calculating the reward, we follow the Monte Carlo policy gradient algorithm [30] to update the controller:

$$\nabla J(\theta) = \frac{1}{m} \sum_{k=1}^{m} \sum_{\tau=1}^{t} \gamma^{t-\tau} \nabla_\theta \log \pi_\theta(a_\tau | a_{(\tau-1):1})(R_k - b) \tag{4}$$

where m is the batch size and t is the total number of steps in each episode. The rewards are discounted at every step by an exponential factor γ and the baseline b is the exponential moving average of the rewards.

4 Estimators for Cryptographic Primitives

While CHET [7] realizes the importance of establishing an abstraction layer for the NN designer to hide specific HE implementation details, they did not think of the cryptographic primitives as *design elements* that carry distinct performance trade-offs (actually, CHET only focused on compiling a single FHE primitive). As observed in Fig. 2, Gazelle instantiate different protocols according to the specific NN layers. Hence, in this section, we describe how to construct estimators that model cryptographic primitives as delay elements with communicational costs, analogous to the FPGA components modeled in the FNAS [11] framework.

4.1 Constructing PIE for PAHE

In this work, we use the widely-adopted BFV [10] cryptosystem as the example PAHE scheme, but our method applies broadly to all RLWE-based PAHE cryptosystems. In BFV, three parameters are required to instantiate the cryptosystem, (n, p, q), where n is the lattice dimension, p the plaintext modulus, and q the ciphertext modulus.

4.1.1 The Feedback Loop

In the Gazelle protocol, since each linear layer is evaluated independently (decryptions are performed after only one layer of homomorphic evaluation), parameters can be minimized. For example, in our experiments, $n = (2048, 4096, 8192)$, and q ranges from 60 to 180 bits. Therefore, even one bit of loose error margin can easily result in 1.5x to 2x performance penalty on 64-bit machines, due to the requirement of extra integer slots (e.g., from 61-bit q to 62-bit q).

In Gazelle, as long as the dimensions and quantizers are the same, the parameters do not scale with the number of layers in the NN. Therefore, parameter minimization needs to be carried out for every NN layer with varying quantization and filter dimensions. The main difficulty for per-layer parameter minimization lies in the feedback loop between PCE and PIE. The dilemma is that, in order for PIE to instantiate parameters that ensure correct decryption, the error size (explained

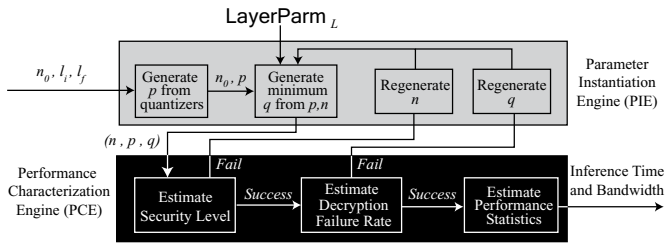

Figure 4. The PIE-PCE co-optimization procedures for characterizing the performance cost of a linear layer.

in Section 2.2) needs to be estimated by PCE. Meanwhile, PCE needs instantiated PAHE parameters from PIE to perform error analysis, thereby forms the loop. Iterating through all possible parameter combinations with error calculations for each NN layer creates significant computational burden in the NASS optimization process. In addition, generating large primes can also be time consuming, as BFV requires additional constraints on the relationship of p, q and n to enable the batching technique [28], which is essential to the efficiency of SI. In particular, both p and q need to satisfy $p \equiv q \equiv 1 (\text{mod } n)$, where q can be a large integer (e.g., 120 bits).

4.1.2 Instantiating the Parameters

An overview of the joint parameter optimization procedure is illustrated in Fig. 4.

① **Initialization**: To start the optimization process, inputs are first fed to PIE. The inputs include n_0, the initial lattice dimension, and (l_i, l_f), the respect quantizers for NN inputs and filters. Here, n_0 is an arbitrary number, and can be set as the smallest n that grants some security levels for extremely small q (e.g., $n_0 = 1024$, which is secure for $q \leq 2^{32}$). l_i and l_f is used to determine the plaintext modulus p. In order to carry out a successful inference, we need that $p \geq l_i + l_f + \lceil \log_2(f_h \cdot f_w) \rceil$ and $p \equiv 1 (\text{mod } n_0)$. After generating the plaintext modulus p, along with n_0 and other parameters in $\mathsf{LayerParms}_\mathsf{L}$ (e.g., the input dimensions n_i, n_o and the filter dimension f_h and f_w), we can calculate a working ciphertext modulus q. Note that this estimation can be loose, but will be tightened in PCE through optimization iterations.

② **Optimization Loop**: Upon receiving parameters from PIE, PCE performs two important evaluations: i) security level estimation, and ii) decryption failure rate estimation. Failure in meeting either of the conditions results in an immediate rejection. First, in i), The security levels are consulted with the community standard established in [1]. When the security standard is not met, we regenerate the lattice dimension n and retry the security analysis. Next, in ii), after obtaining a valid n for the estimated q, a set of ciphertexts are created to see if q is large enough for correct decryption. If the decryption failure rate is too high, we regenerate a larger q and re-evaluate the security of n with respect to the new q. After deriving valid (n, p, q) that passes all the tests, the parameters are fed into a PAHE library to characterize the estimated amount of time and memory consumed by a single layer to calculation.

③ **Output Statistics**: Steps ① and ② described above will be repeated for every layer in the input neural architec-

ture, and all performance statistics are summed up to produce a final score to be used by the overall NASS framework in searching for a better neural architecture for SI.

4.1.3 Generating a Valid Ciphertext Modulus

One last note on the ciphertext modulus q is that, as mentioned in Section 2.1, not all ciphertexts become undecipherable when q is small. The probability that a ciphertext becomes undecipherable is called the decryption failure rate. Observe that different from [3], we do not need an expensive simulation to ensure an asymptotically small (e.g., 2^{-40}) decryption failure probability, since NN-based SI mispredicts much more often than 2^{-40}. In most cases, a 0.1% accuracy degradation is not noticeable for practical CNN applications. Therefore, we can use the standard Monte-Carlo simulation technique to ensure that q is large enough to ensure that $\Pr[\mathsf{Enc}(\mathsf{Dec}(m)) \neq m] < \delta$, where δ ranges from 10^{-3} (1 decryption failure in 1000 inferences) to 10^{-2} (1 in 100), depending on the prediction accuracy requirement.

4.2 PCE: Performance Characterization

4.2.1 Characterizing Linear Layers

The main arithmetic computations in both Conv and FC involve a set of inner products between some plaintext matrix and ciphertext matrix (flatten as vectors) homomorphically. To compute any homomorphic inner product, the pioneering work in [16] proposes to align the weight matrix with the rotating input ciphertext vector to minimize the number of homomorphic operations. In general, the algorithm computes the inner product between $W \in \mathbb{Z}_p^{n_o \times n_i}$, a weight matrix, and $[\mathbf{u}] \in \mathbb{Z}_q^{n_i}$, the encrypted input vector as follows.

$$[\mathbf{t}] = \sum_{i=0}^{n_o-1} \mathbf{w}_i \boxdot \mathsf{rot}([\mathbf{u}], i) \tag{5}$$

$$= \mathbf{w}_0 \boxdot [\mathbf{u}] + \cdots + \mathbf{w}_{n_o-1} \boxdot \mathsf{rot}([\mathbf{u}], n_o - 1), \tag{6}$$

$$[\mathbf{v}] = \sum_{i=1}^{\lg(n/n_o)} \mathsf{rot}\left([\mathbf{t}], \frac{n}{2^i}\right), \tag{7}$$

where $[\mathbf{v}]$ holds the result vector $\mathbf{v} = W\mathbf{u} \in \mathbb{Z}_p^{n_o}$, $\mathbf{w}_i$'s are the diagonally aligned columns of W with dimension $\mathbf{w}_i \in \mathbb{Z}_p^n$, and $\lg(\cdot)$ denotes $\log_2(\cdot)$. In Eq. (6), we first rotate $[\mathbf{u}]$ n_o times, each time multiplying it with the aligned vectors $\mathbf{w}_i \in \{\mathbf{w}_0, \cdots, \mathbf{w}_{n_o-1}\}$. Each multiplication generates an intermediate ciphertext that holds only *one* entry in $\mathbf{v}_i$ with respect to $\mathbf{w}_i$. Summing these ciphertexts gives us a single ciphertext that is packed with n/n_o partial sums in the corresponding inner products, and packed results can be summed up to obtain the final product $[\mathbf{v}]$.

It is noted that the performance non-linearity illustrated in Fig. 1 lies critically in the way homomorphic inner products are computed. Take a toy example where $n_o \times n_i = 10 \times 1024$ and $n = 1024$. The input vector $\mathbf{u} \in \mathbb{Z}^{n_o}$ can be tightly stored into a single ciphertext $[\mathbf{u}] \in \mathbb{Z}^n$. Using the Gazelle algorithm, we rotate the input ciphertext $n_o = 10$ times, and compute 10 homomorphic Hadamard products. However, suppose that the input dimension is somehow 10×1025. Since the lattice dimension n can only be a power of 2, the ciphertext size becomes $\mathbb{Z}^{2048}$ in order to hold an input vector $\mathbf{u} \in \mathbb{Z}^{1025}$. All subsequent homomorphic evaluations require double the amount of computations and bandwidths compared to $n = 1024$. If the amount of accuracy improvement from $n_i = 1024$ to $n_i = 1025$ is marginal (e.g., $\leq 0.01\%$), then this improvement suggestion should be rejected.

The above example represents the precise procedure performed in PCE, where all the information contained in $\mathsf{LayerParms}_{\mathrm{L}}$ jointly determines how packing can be performed to maximize the protocol efficiency. The output of this procedure is the inference time T and network bandwidth B. We use a simple weighted sum to derive the performance score ξ, where

$$\xi(T, B) = \beta \cdot T + (1 - \beta) \cdot B. \tag{8}$$

One important implementation detail is that, instead of performing the entire calculation using an actual PAHE implementation, only basic operations ($\boxplus$, $\boxdot$ and rot) need to be characterized. The actual runtime and bandwidth usage can be scaled from the combinations of the basic operations. In fact, this can be a critical performance improvement, as secure inference is still quite slow for deep neural networks (10 to 100 seconds), and running PCE for full network characterizations can be a performance bottleneck in the optimization process.

4.2.2 PCE for Non-Linear Layers

Running PCE for interactive protocols such as multiplication triples [2] and GC [34] is much simpler than linear layers, as these non-linear functions (e.g., square and ReLU) are performed on a per-element basis with fixed functionality. The performance statistics can be characterized once, and used throughout all layers when properly scaled.

5 Numerical Experiments and Parameter Instantiations

5.1 Experiment Setup

In this work, we compare NASS with the performance statistics of two best-performing recent works on secure inference, namely, Gazelle [16] and XONN [24]. First, we point out that the reported statistics are not entirely reliable in Gazelle. For example, the architecture used for a single CIFAR-10 inference needs more than 120,000 ReLU calls, and Gazelle reports 551 ms of runtime per 10,000 ReLU evaluations. Nevertheless, the total online inference time is less than three seconds. Since the main focus of NASS is to improve the architectural design of NNs, the performance of both Gazelle and the derived architectures in this work are characterized by the proposed performance estimator. We base our experiments on three datasets, MNIST [18], fashion-MNIST [31], and CIFAR-10 [17]. The characterization experiments are conducted with a Intel i3-6100 3.7 Ghz CPU, and the architectural search is peformed using a NVIDIA P100 GPU. The adopted PAHE library is SEAL version 3.3.0 [26], and GC protocols are implemented using ABY [8].

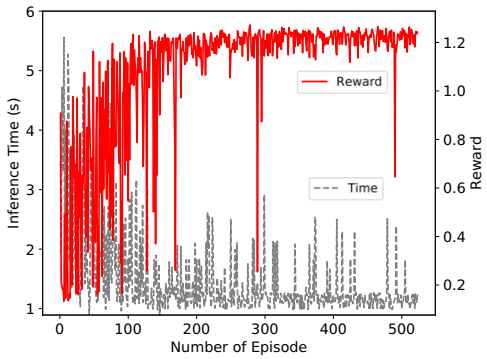

Figure 5. The learning curves show that as the number of episodes increase, both the reward and the inference time tend to converge.

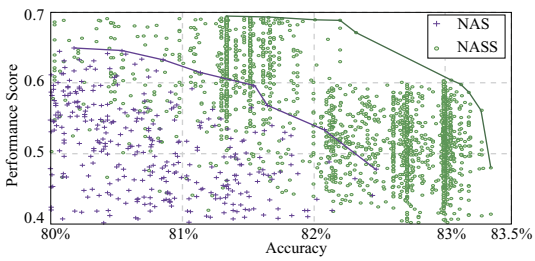

Figure 6. The proposed NASS can significantly push forward Pareto frontier in terms of accuracy and score, compared with the NAS without considering secure inference: each point indicates an architecture with specific quantization parameters.

5.2 Architectural Optimization and the Pareto Frontier

First, in Fig. 5, we use an example NASS run using the CIFAR-10 dataset to show the effectiveness of the proposed framework. We trained the NASS controller through a 500 episodes window, where the neural architecture is fixed to have four convolution layers. As explained in Section 3.2, each episode generates a child network, and the reward of the child network is calculated in Eq. (3) using the prediction accuracy and cryptographic performance scores. The results in Fig. 5 indicate that both the rewards (where accuracy dominates the calculation, as described in Eq. (3)) and the secure inference times converge to their optimized states as learning episodes proceed. Furthermore, in Fig. 6, the best-performing data points are gathered to plot the Pareto frontiers generated by NASS. Here, the vertical axis is the estimated performance score, and the horizontal axis denotes the prediction accuracy on the CIFAR-10 dataset. Two observations are made here. First, the proposed NASS engine is able to learn the extremely non-linear design space of CNN-based SI, and second, the NASS framework pushes forward the Pareto frontier for SI compared to existing works on both SI and NAS.

The predicted architectures laying on the Pareto frontier of the tested datasets are summarized in Table 1. Two types of architectures are selected here. Architectures with a suffix of -Acc are the child networks that have better accuracy but (relatively) worse performance, and -Per the reverse. The insight here is that, for smaller (i.e., easier) datasets such as MNIST, the search is almost exhaustive, where the best architecture achieves highest prediction accuracy and cryptographic performance. Nevertheless, for more complex datasets, the differences become increasingly large, where distinctive trade-offs between neural architectures emerge. We emphasize that depending on the application, all neural architectures on the Pareto frontier are legitimate candidates. However, as also shown in Fig. 6, in many cases, significant performance degradation only results in marginal accuracy improvement, and vise versa.

Table 1. Selected Architectures from NASS

Architecture	Accuracy	Total Time	Bandwidth	No. Episode (Search Time)
MNIST-Acc	98.6%	0.79 s	17 MB	1000 (17 hrs.)
MNIST-Per	98.6%	0.79 s	17 MB	1000 (17 hrs.)
Fashion-Acc	90.6%	1.67 s	50 MB	2000 (32 hrs.)
Fashion-Per	90.4%	0.72 s	22 MB	2000 (32 hrs.)
CIFAR-Acc	84.6%	8.0 s	944 MB	1200 (60 hrs.)
CIFAR-Per	82.6%	5.1 s	582 MB	1200 (60 hrs.)

5.3 Comparison to Existing Works

Table 2. Comparison Between Gazelle and Architecture from NASS

Gazelle			Best Searched by NASS		
Layer	Dimension	Quant.	Layer	Dimension	Quant.
CR	$(64 \times 3 \times 3)$	23	CR	$(24 \times 5 \times 3)$	$(8, 8)$
CR	$(64 \times 3 \times 3)$	23	CR	$(48 \times 3 \times 5)$	$(6, 7)$
PL	(2×2)	23	PL	(2×2)	$(8, 8)$
CR	$(64 \times 3 \times 3)$	23	CR	$(48 \times 5 \times 7)$	$(7, 6)$
CR	$(64 \times 3 \times 3)$	23	CR	$(36 \times 3 \times 3)$	$(6, 5)$
PL	(2×2)	23	PL	(2×2)	$(8, 8)$
CR	$(64 \times 3 \times 3)$	23	CR	$(24 \times 7 \times 1)$	$(4, 6)$
CR	$(64 \times 3 \times 3)$	23			
FC	(1024×10)	23	FC	(1024×10)	$(16, 16)$
Accuracy: 81.6%			Accuracy: 84.6%		
Bandwidth: 1.815 GBytes			Bandwidth: 977 MB		
PAHE Time: 3.22 s GC Time: 13.2 s Total Time: 16.4 s			PAHE Time: 1.62 s GC Time: 6.38 s Total Time: 8.0 s		

We selected the CIFAR-Acc from Table 1 to compare the NASS against the baseline architecture proposed in [20]. The architectures are summarized in Table 2. Here, CR depicts a convolution layer plus a ReLU layer, and PL is an average pooling layer. Dimension indicates the filter dimension, and the input dimension is $(3 \times 32 \times 32)$ in the CIFAR-10 dataset. The important observation here is that, by using architectural search, we do not need to trade accuracy for performance. The generated neural architecture requires only 5 convolution layers rather than 6 (as used in the baseline architecture), while improving the prediction accuracy from 81.6% to 84.6%. The inference time and network bandwidth are reduced by 2x and 1.9x, respectively. The reduction rate can be increased to more than 3x when the same level of accuracy suffices, as demonstrated by the CIFAR-Per architecture in Table 1. Finally, we note that the very recent work [24] that achieves a prediction accuracy of 85% requires more than 30 seconds to carry out the inference, which translates to 4x time reduction when compared to CIFAR-Acc.

6 Conclusion

In this work, NASS is proposed to optimize neural network architectures used in secure inference schemes. Models of cryptographic primitives are created to automatically generate computational and communicational profiles. Rewards are generated based on the calculated profiles and fed to a NAS optimizer to search in the architectural space of convolutional neural networks. Experiments show that security-centric designs result in better inference speed and bandwidth footprint compared to manually tuned neural architectures, while achieving better prediction accuracy.

Acknowledgment

This work was partially supported by JSPS KAKENHI Grant No. 17H01713, 17J06952, Grant-in-aid for JSPS Fellow (DC1), National Science Foundation under Grant CNS-1822099, and Edgecortix Inc.

REFERENCES

[1] Martin Albrecht et al., 'Homomorphic encryption security standard', Technical report, HomomorphicEncryption.org, Toronto, Canada, (November 2018).

[2] Donald Beaver, 'Efficient multiparty protocols using circuit randomization', in *Annual International Cryptology Conference*, pp. 420–432. Springer, (1991).

[3] Song Bian et al., 'Darl: Dynamic parameter adjustment for lwe-based secure inference', in *Proc. of DATE*, pp. 1739–1744. IEEE, (2019).

[4] Zvika Brakerski, 'Fully homomorphic encryption without modulus switching from classical GapSVP', in *Advances in Cryptology–CRYPTO 2012*, 868–886, Springer, (2012).

[5] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan, '(Leveled) fully homomorphic encryption without bootstrapping', *ACM Transactions on Computation Theory (TOCT)*, **6**(3), 13, (2014).

[6] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song, 'Homomorphic encryption for arithmetic of approximate numbers', in *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 409–437. Springer, (2017).

[7] Roshan Dathathri et al., 'CHET: an optimizing compiler for fully-homomorphic neural-network inferencing', in *Proc. of PLDI*, pp. 142–156. ACM, (2019).

[8] Daniel Demmler et al., 'Aby-a framework for efficient mixed-protocol secure two-party computation.', in *Proc. of NDSS*, (2015).

[9] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter, 'Efficient multi-objective neural architecture search via lamarckian evolution', in *International Conference on Learning Representations*, (2019).

[10] Junfeng Fan and Frederik Vercauteren, 'Somewhat practical fully homomorphic encryption.', *IACR Cryptology ePrint Archive*, **2012**, 144, (2012).

[11] Weiwen Jiang et al., 'Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search', *in Proc. of DAC*, (2019).

[12] Weiwen Jiang, Qiuwen Lou, Zheyu Yan, Lei Yang, Jingtong Hu, Xiaobo Sharon Hu, and Yiyu Shi, 'Device-circuit-architecture co-exploration for computing-in-memory neural accelerators', *arXiv preprint arXiv:1911.00139*, (2019).

[13] Weiwen Jiang, Lei Yang, Edwin Sha, Qingfeng Zhuge, Shouzhen Gu, Yiyu Shi, and Jingtong Hu, 'Hardware/software co-exploration of neural architectures', *arXiv preprint arXiv:1907.04650*, (2019).

[14] Xiaoqian Jiang et al., 'Secure outsourced matrix computation and application to neural networks', in *Proc. of ACM SIGSAC Conference on Computer and Communications Security*, pp. 1209–1222. ACM, (2018).

[15] Mika Juuti et al., 'Prada: protecting against dnn model stealing attacks', in *Proc. of EuroS&P*, pp. 512–527. IEEE, (2019).

[16] Chiraag Juvekar et al., 'Gazelle: A low latency framework for secure neural network inference', *arXiv preprint arXiv:1801.05507*, (2018).

[17] Alex Krizhevsky and Geoffrey Hinton, 'Learning multiple layers of features from tiny images', Technical report, Citeseer, (2009).

[18] Yann LeCun, Corinna Cortes, and CJ Burges, 'MNIST handwritten digit database', *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, **2**, (2010).

[19] Chenxi Liu et al., 'Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation', in *Proc. of CVPR*, pp. 82–92, (2019).

[20] Jian Liu et al., 'Oblivious neural network predictions via MinioNN transformations', in *Proc. of ACM SIGSAC Conference on Computer and Communications Security*, pp. 619–631. ACM, (2017).

[21] Qing Lu, Weiwen Jiang, Xiaowei Xu, Yiyu Shi, and Jingtong Hu, 'On neural architecture search for resource-constrained hardware platforms', *arXiv preprint arXiv:1911.00105*, (2019).

[22] Payman Mohassel et al., 'Secureml: A system for scalable privacy-preserving machine learning', in *Proc. of Security and Privacy (SP)*, pp. 19–38. IEEE, (2017).

[23] Wei Peng et al., 'Video action recognition via neural architecture searching', in *Proc. of ICIP*, pp. 11–15. IEEE, (2019).

[24] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin E Lauter, and Farinaz Koushanfar, 'Xonn: Xnor-based oblivious deep neural network inference.', *IACR Cryptology ePrint Archive*, **2019**, 171, (2019).

[25] Bita Darvish Rouhani et al., 'Deepsecure: Scalable provably-secure deep learning', in *Proc. of DAC*, pp. 1–6. IEEE, (2018).

[26] Microsoft SEAL (release 3.3). `https://github.com/Microsoft/SEAL`, 2019. Microsoft Research, Redmond, WA.

[27] Reza Shokri et al., 'Membership inference attacks against machine learning models', in *Proc. of Security and Privacy (SP)*, pp. 3–18. IEEE, (2017).

[28] Nigel P Smart and Frederik Vercauteren, 'Fully homomorphic encryption with relatively small key and ciphertext sizes', in *International Workshop on Public Key Cryptography*, pp. 420–443. Springer, (2010).

[29] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le, 'Mnasnet: Platform-aware neural architecture search for mobile', in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828, (2019).

[30] Ronald J Williams, 'Simple statistical gradient-following algorithms for connectionist reinforcement learning', *Machine learning*, **8**(3-4), 229–256, (1992).

[31] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[32] Lei Yang et al., 'Co-exploring neural architecture and network-on-chip design for real-time artificial intelligence', *in Proc. of ASP-DAC*, (2020).

[33] Lei Yang, Weichen Liu, Nan Guan, and Nikil Dutt, 'Optimal application mapping and scheduling for network-on-chips with computation in stt-ram based router', *IEEE Transactions on Computers*, **68**(8), 1174–1189, (2018).

[34] Andrew C Yao, 'Protocols for secure computations', in *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pp. 160–164. IEEE, (1982).

[35] Xinyi Zhang, Weiwen Jiang, Yiyu Shi, and Jingtong Hu, 'When neural architecture search meets hardware implementation: from hardware awareness to co-design', in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 25–30. IEEE, (2019).

[36] Barret Zoph et al., 'Neural architecture search with reinforcement learning', *arXiv preprint arXiv:1611.01578*, (2016).

[37] Barret Zoph et al., 'Learning transferable architectures for scalable image recognition', *arXiv preprint arXiv:1707.07012*, **2**(6), (2017).