COINN: Crypto/ML Codesign for Oblivious Inference via Neural Networks

Siam Umar Hussain*
siamumar@ucsd.edu
UC San Diego
USA

Mojan Javaheripi*
mojan@ucsd.edu
UC San Diego
USA

Mohammad Samragh*
msamragh@ucsd.edu
UC San Diego
USA

Farinaz Koushanfar
farinaz@ucsd.edu
UC San Diego
USA

ABSTRACT

We introduce COINN – an efficient, accurate, and scalable framework for oblivious deep neural network (DNN) inference in the two-party setting. In our system, DNN inference is performed without revealing the client's private inputs to the server or revealing server's proprietary DNN weights to the client. To speedup the secure inference while maintaining a high accuracy, we make three interlinked innovations in the plaintext and ciphertext domains: (i) we develop a new domain-specific low-bit quantization scheme tailored for high-efficiency ciphertext computation, (ii) we construct novel techniques for increasing data re-use in secure matrix-multiplication allowing us to gain significant performance boosts through factored operations, and (iii) we propose customized cryptographic protocols that complement our optimized DNNs in the ciphertext domain. By co-optimization of the aforesaid components, COINN brings an unprecedented level of efficiency to the setting of oblivious DNN inference, achieving an end-to-end runtime speedup of 4.7×–14.4× over the state-of-the-art. We demonstrate the scalability of our proposed methods by optimizing complex DNNs with over 100 layers and performing oblivious inference in the Billion-operation regime for the challenging ImageNet dataset. Our framework is available at https://github.com/ACESLabUCSD/COINN.git.

CCS CONCEPTS

• **Security and privacy** → **Privacy-preserving protocols**; • **Computing methodologies** → **Machine learning**.

KEYWORDS

Privacy-preserving deep neural network inference; secure two-party computation

ACM Reference Format:
Siam Umar Hussain, Mojan Javaheripi, Mohammad Samragh, and Farinaz Koushanfar. 2021. COINN: Crypto/ML Codesign for Oblivious Inference via Neural Networks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3460120.3484797

*Authors contributed equally to this research.

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea.

ACM ISBN 978-1-4503-8454-4/21/11.
https://doi.org/10.1145/3460120.3484797

1 INTRODUCTION

Recent algorithmic and technological breakthroughs in Machine Learning (ML) have led to a surge in cloud-based inference using Deep Neural Networks (DNNs). In this scenario, a server trains and holds the DNN model. Clients then send their data to the server to perform inference using the server's trained DNN. Cloud-based inference, a.k.a. Machine Learning as a Service (MLaaS), is integrated in a wide range of real-world applications such as personal assistants [32], face authentication [40], medical diagnosis [4, 18, 19, 45], and health monitoring [3]. However, plaintext DNN inference either violate the users' privacy by revealing their private data to the server or infringe the server's intellectual property by exposing its proprietary model/data to the client. This paper focuses on the critical subject of oblivious inference, where the server and the client participate in two-party secure computation to run inference without revealing either the model parameters or client's data.

We present COINN, a provably secure cryptographic framework that surpasses the efficiency of all known methods for oblivious inference to date. Our work addresses the tension between three critical requirements for privacy-preserving DNN inference, namely, security, efficiency, and accuracy. Although several prior works have attempted to solve this tri-objective, there still remains a large gap in the accuracy and/or runtime of oblivious inference and plaintext DNN execution. To deliver a balanced tradeoff between the above three criteria, we co-design the DNN and the secure execution protocol and holistically optimize both aspects via our automated design configuration tool. Our key design goals are as follows:

❶ **Compact Communication and Computation:** We optimize the computation bitwidth to reduce the secure execution cost of both linear and nonlinear operations. In doing this optimization, we adapt techniques from Genetic Algorithms [57] to the constraints of secure computation. Moreover, we design efficient cryptographic protocols that reduce the communication cost of secure matrix-multiplication by 5×–9×, and achieve an end-to-end runtime speedup of 4.7×–14.4× over best prior work, namely CrypTFow2 [46], in the LAN setting.

❷ **Inference Accuracy:** COINN improves the accuracy of prior ML-security co-optimization methods, namely [38, 41, 47], by 0.6%–4.7% while achieving 23.1×–36.8× lower secure execution runtime in the LAN setting.

❸ **Scalability:** Our framework scales to DNNs with over 100 layers. COINN achieves 6.1×–7.8× lower runtime in the LAN setting for the largest ever studied image classification task [46] with over 4 billion arithmetic operations.

In what follows, we review the design challenges, survey the prior work, and specify our contributions in detail.

Security-aware Quantization. To reduce the high cost of ciphertext execution, contemporary methods modify the neural network architecture by removing/replacing non-linear operations such as ReLU [22, 41], or binarizing model parameters and activations [47]. While these methods increase the secure execution efficiency, they come at the cost of reduced inference accuracy. Our work approaches the problem from a different perspective. Since the computation and communication overheads of cryptographic protocols are highly dependent on the computation bitwidth, we focus on developing quantization methods that take into account the constraints of ciphertext computation.

Our low-bit quantization reduces the communication and computation cost for not only linear but also nonlinear layers which are the main efficiency bottleneck reported in prior works [23, 31, 35, 41]. A critical design challenge is that off-the-shelf quantization methods used in the ML community comprise operations such as full-precision accumulation, rounding, and scaling; these operations are efficient in plaintext inference but require expensive cryptographic operations in ciphertext. To address this challenge, we devise a novel ciphertext-aware quantization scheme that replaces the costly operations with counterparts that are low-cost in the secure domain while minimally affecting the DNN accuracy.

A major concern in computation on quantized data is the linear growth of the computational bitwidth with increased multiplicative depth. To mitigate this growth, prior work [41] locally truncates the bits which sacrifices the correctness of ciphertext computation by introducing random noise as shown in followup work [46]. Developing cryptographic tools for truncation, as suggested in [17, 46], incurs additional secure execution cost. We address this issue at zero cost by simulating the effect of overflow in our ML quantization library. We further provide training methods compatible with our overflow simulation tofi ne-tune model weights and minimize the effect of overflow on model accuracy in the low-bit regime. This, in turn, eliminates the need for truncation altogether.

Efficient and Secure Linear Arithmetic. Matrix-multiplication comprises the core operation performed in linear layers of contemporary ML models. State-of-the-art oblivious inference frameworks employ either Arithmetic Sharing (AS) [22, 35] or Homomorphic Encryption (HE) [31, 41] for secure matrix-multiplication and Garbled Circuit (GC) for the nonlinear operations. In this work, we choose AS for efficient realization of secure linear layers since the secure conversion cost between AS and GC is $\sim$ 2.5× smaller than the conversion cost between HE and GC[1]. Moreover, prior work [47] demonstrates that current HE-based methods [31, 41] would incur additional overhead to provide circuit privacy.

Our secure AS-based matrix-multiplication is optimized for the amortized setting, where one client-server pair runs multiple inferences on the same trained model. In this setting, which is the common scenario in real-world applications, the matrix-multiplication in each linear layer is computed in a single round, thus reducing the effect of network latency on runtime.

Factored matrix-multiplication. We further optimize the linear layers and introduce repetition into the weight matrices. Our optimization ensures that only a limited set of unique values appear in each layer's weight matrix with minimal loss of inference accuracy. The unique values can then be leveraged to replace individual multiplications with *factored* ones. This, in turn, allows us to substitute the bulk of costly multiplications with cheaper conditional summations. Consider a dot product between two N-dimensional vectors, which requires N multiplications and additions. By ensuring that one vector contains V unique values, only N additions followed by V multiplications and additions are required. To accompany factored multiplication in cipher domain, we introduce an efficient custom protocol based on Oblivious Transfer (OT) that multiplies the factored weights with the activations without revealing either the unique values or their locations.

[1]The $\sim$ 2.5× scale directly compares methods in Gazelle [31] (HE-GC) and ABY [16] (AS-GC). Further explanation is included in Section 5.3.

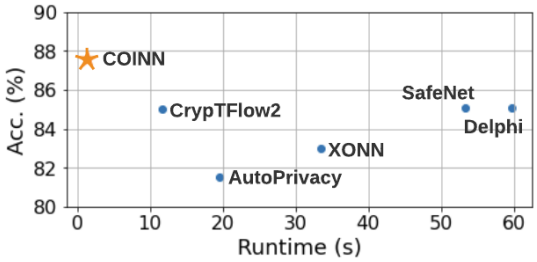

Figure 1: Accuracy and secure inference runtime of a 7-layer DNN on CIFAR-10 dataset using prior work: Delphi [41], SafeNet [38], XONN [47], AutoPrivacy [39], and CrypTFlow2 [46]. The ★ symbol represents COINN.

Automated Parameter Configuration. To fully exploit the efficiency gains from quantization and factored multiplication while minimally affecting the inference accuracy, COINN automatically determines the best parameters for quantization and factorization across all DNN layers. By this cross-layer heterogeneous parameter selection, our system achieves a prominent advantage over previous work that use homogeneous and superfluous bitwidths [2, 35], as shown in Figure 1. Thefi rst challenge infi nding the best set of per-layer parameters is simultaneous optimization of two of our objectives that are conflicting – accuracy and efficiency. To account for this tradeoff, we leverage a score function that captures both model accuracy and secure execution cost and assigns a quantitative measure of quality to each design configuration. The second challenge is the excessively large number of possible parameter configurations (search-space) that grows exponentially with model layers. We develop a highly scalable parameter optimizer based on genetic algorithms [57] to effectively traverse the large search-space. The score function is then used to guide our genetic algorithm tofi nd the most optimal DNN for secure inference.

COINN API. Our framework includes a high-level API that facilitates end-to-end deployment of user-defined DNNs for secure execution. Our API ensures that a user can employ COINN as a black box without knowing the details of underlying cryptographic protocols and DNN optimizations. The user provides the desired DNN model described in the well-known deep learning library PyTorch along with the trained model parameters. The custom-designed libraries of COINN for quantization and factored multiplication are then invoked through our automated design configurator to deliver the optimized DNN. Our framework also provides a seamless PyTorch interface to the secure inference engine developed in C++.

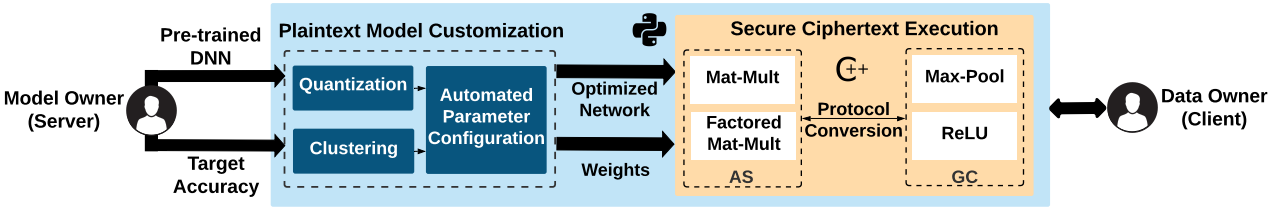

Figure 2: Overview of COINN. The plaintext model customization is only performed once per DNN and provides the optimized network for COINN secure inference.

2 PRELIMINARIES

2.1 Notations

Throughout this manuscript, we represent scalars with lowercase x, vectors with bold lowercase $\mathbf{x}$, 2-dimensional matrices with uppercase X, and higher order tensors with bold uppercase letters $\mathbf{X}$. Element selection is denoted by brackets $\mathbf{x}[i]$ and $x\langle i\rangle$ denotes the i-th bit of scalar x. $\mathbf{0}$ denotes a vector/matrix/tensor with all the entries set to 0. We denote the computational security parameter with κ and set it to 128 following common standard [31, 35, 46].

2.2 Deep Neural Networks

Contemporary DNNs comprise two classes of layers: linear (convolution, fully-connected, batch normalization, and average-pooling) and non-linear (max-pooling and ReLU). We briefly explain commonly used layers in each category.

Convolution. A convolution layer (CONV) is a linear operation $F(\mathbf{X}, \mathbf{W}, \mathbf{b}) : \mathbb{R}^{C\times D_1\times D_1} \rightarrow \mathbb{R}^{M\times D_2\times D_2}$, where $\mathbf{X} \in \mathbb{R}^{C\times D_1\times D_1}$ is the 3-way input tensor, $\mathbf{W} \in \mathbb{R}^{M\times C\times k\times k}$ is the 4-way weight tensor, $\mathbf{b} \in \mathbb{R}^{M}$ is the bias vector, and $\mathbf{Y} \in \mathbb{R}^{M\times D_2\times D_2}$ is the 3-way output tensor. The plaintext operation of CONV can be represented as a matrix-multiplication followed by bias addition $Y = W \cdot X + \mathbf{b}$ where $W \in \mathbb{R}^{M\times N}$ is achieved by reshaping the original 4-way tensor into a $2D$ matrix and $X \in \mathbb{R}^{N\times L}$ is formed by sliding through the original 3-way tensor and vectorizing the corresponding windows into matrix columns. Each element of the output is computed via a vector dot product (VDP) and the total number of VDPs required for the matrix-multiplication is $M \times L$.

Fully-Connected. The fully-connected (FC) layer takes a vector $\mathbf{x} \in \mathbb{R}^{N}$ and generates the output vector $\mathbf{y} = W \times \mathbf{x} + \mathbf{b}$ where $W \in \mathbb{R}^{M\times N}$ and $b \in \mathbb{R}^{M}$ are the weight and bias, respectively. Similar to CONV, the matrix-vector multiplication consists of M VDPs between rows of W and $\mathbf{x}$.

Batch Normalization. Batch normalization (BN) is a common linear operation applied on the output of CONV layers to adjust the range of numerical values. At test time, BN computes $\mathbf{y}_i^{(BN)} = \alpha_i \mathbf{y}_i + \beta_i$, where α_i and β_i are constant scalars, $\mathbf{y}_i$ is one row of the output $Y \in \mathbb{R}^{M\times L}$ from the preceding CONV, and $\mathbf{y}_i^{(BN)}$ is the corresponding row after BN.

Pooling. Contemporary DNNs include two forms of pooling layers, namely max-pooling (MP) and average-pooling (AP). These layers extract $k \times k$ windows from the input $\mathbf{X} \in \mathbb{R}^{C\times D_1\times D_1}$ and compute the average or the maximum value in the enclosed window as the output. Assuming the $k \times k$ windows are non-overlapping, pooling layers reduce data dimensionality from $C \times D_1 \times D_1$ to $C \times \frac{D_1}{k} \times \frac{D_1}{k}$.

ReLU. This layer often follows a linear layer to introduce non-linearity in the model. A ReLU operation simply replaces negative inputs with zero and keeps positive values intact.

2.3 Cryptographic Primitives

Oblivious Transfer. Oblivious Transfer (OT) [44] allows a receiver Alice to choose one from a set of messages provided by a sender Bob without revealing her choice. In a 1-out-of-2 OT, (OT_λ^2), Bob holds a pair of messages $\{\mu_0, \mu_1\} \in \{0,1\}^\lambda$; Alice holds a choice bit $\sigma \in \{0,1\}$ and obtains μ_σ without revealing σ. An extension of this protocol, called OT extension [28], allows performing a large number of OT_λ^2 with afi xed number of base OTs and linear number of less expensive private-key operations. In this work, we employ two variants of OT extension: random OT (ROT_λ^2) and correlated OT (COT_λ^2) [5]. In ROT_λ^2, instead of choosing his messages, Bob receives random messages $\{\mu_0, \mu_1\}$ and Alice receives μ_σ. The communication cost of one ROT_λ^2 is a κ-bit message embedding the choice bit σ from Alice to Bob. In COT_λ^2, Bob chooses a correlation function $\varphi(\mu)$ and receives a random message μ. Alice receives μ if $\sigma = 0$ and $\varphi(\mu)$ if $\sigma = 1$. The communication cost of one COT_λ^2 is cost of one ROT_λ^2 plus a λ-bit message from Bob to Alice, i.e., $\kappa + \lambda$.

Arithmetic Sharing (AS). We denote the arithmetic share (AS) [6] of an integer x between two parties Alice and Bob as $[\![x]\!]$. For b-bit arithmetic sharing, $[\![x]\!] = [\![x]\!]_A + [\![x]\!]_B \bmod 2^b$, where $[\![x]\!]_A$ is held by Alice and $[\![x]\!]_B$ is held by Bob with $[\![x]\!], [\![x]\!]_A, [\![x]\!]_B \in \mathbb{Z}_{2^b}$. All operations on arithmetic shared values are performed in ring $\mathbb{Z}_{2^b}$, i.e., operations are $\bmod 2^b$.

Addition and Multiplication in AS. In AS, addition of shared variables is free since each party can locally add their shares without communication. Multiplication can be performed through COT following [16]. Let us consider the scalar product $[\![z]\!] = [\![w]\!]_A [\![x]\!]_B$. For each bit $i \in [b]$, Alice and Bob engage in one COT_b^2. Bob acts as the sender with the correlation function $\phi(\mu_i) = \mu_i + [\![x]\!]_B * 2^i$ and receives μ_i. Alice acts as the receiver with choice bit $\sigma_i = [\![w]\!]_A\langle i\rangle$ and receives $\mu_{\sigma_i} = \mu_i + \sigma_i [\![x]\!]_B * 2^i$. Alice and Bob then compute $[\![z]\!]_A = \sum_{i=0}^{b-1} \mu_{\sigma_i}$ and $[\![z]\!]_B = -\sum_{i=0}^{b-1} \mu_i$, respectively. Note that μ_is are random masks that are independent of the input $[\![x]\!]$ and are generated offline before $[\![x]\!]$ is known. Since $[\![z]\!]_B$ only depends on μ_i, it can also be generated offline.

Garbled Circuits. Yao's Garbled Circuit (GC) [59] is a 2PC protocol between Alice, the garbler and Bob, the evaluator, that allows computation on a function's Boolean logic representation. In GC, to share a b-bit integer $x \in \mathbb{Z}_{2^b}$, for each bit $i \in [b]$, Alice holds a global key $\Delta \in \{0,1\}^\kappa$ and a label $L_i^x \in \{0,1\}^\kappa$ while Bob holds $L_i^x \oplus x\langle i\rangle\Delta$. GC is less efficient than AS for addition and multiplication. However, GC is currently the most efficient protocol for logical operations, e.g., `AND` and `XOR`. Moreover, in GC, bit level manipulations, e.g., left or right shift, incur negligible costs. A b-bit private integer can be securely transferred from AS share to GC share (and vice-versa) by executing a b-bit addition through GC [16].

3 COINN OVERVIEW AND THREAT MODEL

The COINN framework is composed of two interlinked components as depicted in Figure 2: (i) model customization on plaintext training data and (ii) secure execution on client's private input. We use the popular ML library, PyTorch, to describe the DNNs and develop our secure execution protocols in C++. In the following, we briefly introduce the incorporated design units.

Plaintext Model Customization. This is a one-time preprocessing performed on pre-trained full-precision DNNs prior to oblivious inference. Plaintext model customization is an important contributor to COINN efficiency and scalability as it enables customization of any given DNN for minimized secure execution cost under an accuracy constraint. Section 4 encloses the details of our plaintext model customization and its core components, i.e., ciphertext aware quantization (§ 4.1), factored matrix-multiplication (§ 4.2), and automated parameter configuration (§ 4.3).

Secure Ciphertext Execution. We perform the linear operations such as `CONV`, `FC` through AS, and the nonlinear operations such as `ReLU`, `MP` through GC. Wherever necessary, we securely convert between AS and GC. We devise efficient cryptographic protocols that complement our optimized DNN models in the ciphertext domain. Our cryptographic components benefit from low-bit quantization performed by our model customization step. We also develop efficient AS-based protocols for both regular and factored matrix-multiplications. A thorough explanation of our end-to-end oblivious inference and cryptographic protocols is provided in Section 5.

3.1 Threat Model

COINN presents privacy-preserving protocols involving two parties: Alice – the server, and Bob – the client. The private inputs of Alice and Bob are trained weight parameters of the DNN and input to the DNN, respectively. At the end of the protocol execution, Bob learns the inference results without revealing any information to Alice. Following previous works on privacy-preserving neural network inference [31, 35, 46], we adopt an honest-but-curious security model where the two parties follow the agreed upon protocol, yet may try to learn more from the information at hand.

Consistent with prior work, we assume the information related to model architecture is public to the client and the server. This information includes number of layers, layer types, layer dimensions, number of bits required to represent the output of nonlinear layers, and AS computation ring size $\mathbb{Z}_{2^b}$. In our factored matrix multiplication (Section 4.2), the client additionally knows the per-layer number of unique weight values V but he is not aware of the distribution of the unique values inside the weight matrices.

Most prior works assume a large bitwidth b across all DNN layers (e.g., 32-bit ring size and activations in [46]). In contrast, COINN uses smaller bitwidths, e.g., it may use $b = 16$ for the ring size and $b = 10$ for the output of nonlinear layers. Exposing the customized b at each layer might reveal some information about the context and/or distribution of the training dataset. It is unclear whether this information can give additional advantage to an attacker. In addition, having lower bitwidths may reduce the computational complexity for extracting the neural network weights. Let us consider an attacker who launches a brute-force attack without any prior knowledge. The computation complexity for such an attack is $O(2^{bn})$, where n is in the order of millions. Therefore, even with the minimum bitwidth ($b = 1$ as in XONN [47]) the attack complexity, i.e., $O(2^n)$, is still exponential in n. Similarly, by knowing the unique size V for factored matrix multiplication, the attack complexity is $O(V^n)$.

A more knowledgeable adversary might try to employ more sophisticated attacks such as model extraction [54], model inversion [20], and membership inference [52]. Similar to related work in oblivious inference [22, 31, 35, 41, 46, 47], COINN does not address these query-based attack algorithms. Mitigating such attacks is also an active area of research and in most cases is orthogonal to our work [1, 12, 47, 51, 58]. Example mitigation strategies include differential privacy, rounding the prediction vector, or returning only the argmax of the prediction to the client. We refer the curious reader to [47]-Appendix B and [41]-Section 8.2 for more discussions.

4 COINN MODEL CUSTOMIZATION

This section elaborates on COINN methodologies for plaintext model preparation devised to minimize the secure execution cost while maintaining inference accuracy.

4.1 Ciphertext-aware Quantization

Most contemporary ML libraries utilize 32-bitfl oating-point format (`FP32`) for data representation. In practice, the extremely high computational cost and complex circuits make `FP32` unsuitable for secure computation. Quantization addresses the aforesaid shortcomings by representing data in the integer format with a lower number of bits. Figure 3 demonstrates how `FP32` values can be converted to low-bit integers through quantization. Let us denote the signed integer format with b bits by `INT-b`. The mapping of an `FP32` parameter x_f to its `INT-b` representation x_q is computed as:

$$x_q = round(s \cdot x_f)\ , \quad s = \frac{2^b - 1}{2 \times \max(|x_f|)} \tag{1}$$

where s is called the scaling factor and $\max(|x_f|)$ denotes the maximum range that parameter x_f can take. In a linear layer with `FP32` inputs X_f, weight parameters W_f, and bias $\mathbf{b}_f$, the output can be approximated using quantized values as:

$$Y_f = W_f \cdot X_f + \mathbf{b}_f \approx \frac{1}{s_w s_x}(W_q \cdot X_q + \frac{s_w s_x}{s_b}\mathbf{b}_q) \tag{2}$$

where s_x, s_w, and s_b denote the quantization scales for the input, layer weights, and the bias, respectively. The quantized version of Y_f is calculated using the corresponding scale s_y as follows:

$$Y_q = round\left(\frac{s_y}{s_w s_x}(W_q \cdot X_q + \frac{s_w s_x}{s_b}\mathbf{b}_q)\right) \tag{3}$$

Y_q is the quantized output of the linear layer which serves as the input of the next layer in a quantized DNN. While evaluating Eq. 3 is straightforward in plaintext, multiplication by the quantization scales $s = \frac{s_y}{s_w s_x}$ and $round(\cdot)$ incur significant costs in ciphertext. In what follows, we first introduce our highly efficient counterparts for these operations designed to minimize the secure execution cost. We then explain how we manage overflow in the low-bit regime.

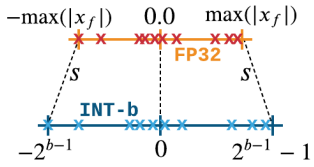

Figure 3: Quantizing FP32 values for INT-b representation.

Optimizing Scaling. In our framework, the matrix-multiplication $W_q.X_q$ as well as the addition with the bias vector are computed efficiently via AS. Scaling the result by $s = \frac{s_y}{s_w s_x}$ in AS would increases the overall multiplicative depth for computing $s(W_q \cdot X_q + \mathbf{b})$, increasing the AS computation bitwidth, thereby sacrificing the overall efficiency. To avoid bit-extending the matrix-multiplication operands, we separate scaling and evaluate it using GC ciphers. In this scenario, for scaling a b-bit number with a scale containing b' nonzero bits, the GC communication cost would be $2b(b' - 1)\kappa$. Instead, we enforce the scale values to be powers of 2, which allows us to implement the previously costly scale operation with $\sim$*zero* cost logical shifts in GC. We do this by replacing the original formula in Eq. 1 with Eq. 4, and fine-tuning the DNN to adjust the quantization and preserve inference accuracy.

$$s = 2^{\left\lceil \log_2 \frac{2^b - 1}{2 \times \max(|x_f|)} \right\rceil} \tag{4}$$

Rounding Workaround. Let us consider an n-bit integer value, right shifted by $n - b$ bits through the scaling step to obtain a fixed-point value with b bits integer and $(n - b)$ bits fraction. Rounding operation in GC works by adding the MSB of the fraction with the b-bit integer. The GC cost is therefore equal to $2b \times \kappa$, which is quite significant considering it has to be repeated for all output elements across all DNN layers. To eliminate this cost, we replace $round(\cdot)$ with the floor operation $\lfloor \cdot \rfloor$ in our plaintext DNNs. Since flooring is equivalent to removing all fraction bits, it incurs no GC cost. To adapt the model weights to this modification, we use the original training data to fine-tune the model. This is done by applying floor during the forward pass and straight-through gradients during the backward pass. Data-free fine-tuning is also possible at the cost of a small accuracy loss as in [25].

Overflow Management. Performing matrix-multiplication requires repeatedly updating an accumulator $y := y + \mathbf{w}[i]\mathbf{x}[i]$. An imminent challenge when moving to the low-bit quantized regime is the occurrence of overflow in the accumulator. To avoid overflows, existing ML libraries for quantization perform accumulations using high-precision data representations, e.g., y is INT-32 while x and w are low-bit. In secure execution, high-precision accumulators are extremely costly. Therefore, we augment the underlying ML library with a new custom operation that simulates DNN execution with low-bit accumulators and directly models the occurrence of overflows. This allows us to match the plaintext inference accuracy with the accuracy obtained via low-bit accumulations in the ciphertext domain. Building upon our customized overflow simulation, we provide an automated strategy that finds the best allocation of bitwidths across DNN layers to minimize accuracy degradation, as will be discussed in Section 4.3. We further develop an overflow-aware training scheme which enables us to adjust the model parameters such that the adverse effect of overflow on inference accuracy is minimized. Since overflow simulation involves non-differentiable operations, we devise an approximate gradient for this function to allow fine-tuning of our quantized models. Details of our overflow simulation and its gradient approximation are included in Appendix A.

4.2 Factored Matrix-Multiplication

Matrix-multiplication accounts for the bulk of computations in DNN inference, which leads to a high communication cost in AS. Our goal in this section is to reduce this cost via factored matrix-multiplication, which replaces the majority of costly multiplications with cheaper conditional additions. Below, we introduce the building blocks of factored matrix-multiplication and explain our method in detail. Later, in Section 5, we present a custom COT-based protocol for our factored matrix-multiplication.

Consider a matrix-multiplication of the form $Y = W \cdot X$, where $Y \in \mathbb{R}^{M \times L}$, $W \in \mathbb{R}^{M \times N}$, and $X \in \mathbb{R}^{N \times L}$. This operation can be broken down into $M \times L$ VDPs, where each VDP operates on vectors of length N, $\mathbf{w} \in \mathbb{R}^N$ and $\mathbf{x} \in \mathbb{R}^N$, corresponding to a row of W and a column of X, respectively. Each VDP therefore requires N multiplications and N additions. We propose the factored VDP as the core operation in factored matrix-multiplication. We start with the definitions of the unique space and the coded representation of vectors involved in VDP.

Definition 4.1. The unique space of $\mathbf{w} \in \mathbb{R}^N$ is the set $\mathbf{c} = \{c_1, \ldots, c_V\}$ such that $\mathbf{w}[i] \in \mathbf{c}$ $(\forall\ i \in [N])$. We refer to V as the unique size of $\mathbf{w}$.

Definition 4.2. Given a vector $\mathbf{w} \in \mathbb{R}^N$ and its unique space $\mathbf{c} = \{c_1, \ldots, c_V\}$, the coded representation of $\mathbf{w}$ is a vector of integer indices $\widetilde{\mathbf{w}} \in [V]^N$ such that $\mathbf{w}[i] = \mathbf{c}[\widetilde{\mathbf{w}}[i]]$.

Knowing the unique space $\mathbf{c}$ and the coded representation $\widetilde{\mathbf{w}}$, the factored VDP can be computed via V multiplications and $N + V$ additions: we first compute N conditional additions, each of which adds an input element to one of V accumulators based on its code:

$$\mathbf{s}[v] = \sum_{x \in \mathbb{S}_v} x, \qquad \mathbb{S}_v = \{\mathbf{x}[i] \mid \widetilde{\mathbf{w}}[i] = v\} \tag{5}$$

Next, a VDP is computed between the accumulated values and the unique space of $\mathbf{w}$, i.e., $\text{VDP}(\mathbf{x}, \mathbf{w}) = \text{VDP}(\mathbf{s}, \mathbf{c})$. The benefits of factored multiplication are most substantial when $V << N$[2]. In general, V can be as large as 2^b, where b is the quantization bitwidth of $\mathbf{w}$. Even after quantizing $\mathbf{w}$ with lower bitwidths, V can be quite large, e.g., $V = 64$ for 6-bit weights. To decrease V, we approximate weight matrices with few representative elements via clustering [49]. We provide details of the clustering algorithm in

[2] N is in the order of 100-10000

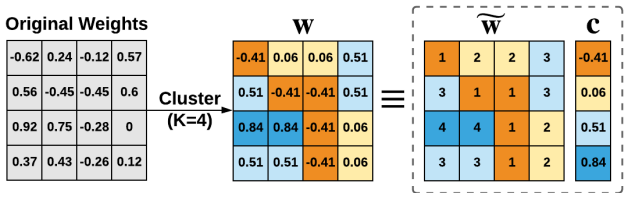

Figure 4: Example 4×4 weight matrix approximated via clustering with $V = 4$. The approximated matrix W can be represented as a tuple $(C, \widetilde{W})$.

Appendix B. Figure 4 shows an example weight matrix and its approximation obtained by clustering. The server performs clustering over plaintext weight matrices and computes $\mathbf{c}$ and $\widetilde{\mathbf{w}}$ for all DNN layers. These values are then used in ciphertext execution.

It is worth noting that the value of V directly affects the tradeoff between the DNN inference accuracy and the secure execution cost. Higher V values achieve higher accuracy but also incur higher secure execution cost. It is a great challenge to determine the per-layer V values and balance this trade-off such that the DNN is executed accurately and efficiently. To address this challenge, we provide an automated algorithm that specifies V for each linear layer in a desired DNN as will be discussed in Section 4.3. Additionally, we develop custom gradient computation methods to enable back-propagation through the clustered weights forfine-tuning and increasing the inference accuracy. We enclose the details of gradient computation for factored weight matrices in Appendix B.

4.3 Automated Parameter Configuration

The quantization bitwidths and the unique spaces across different layers are not independent and they collectively determine the model accuracy as well as the secure execution cost. COINN is equipped with an automated parameter configurator that searches for the optimum number of quantization bits and weight clusters across DNN layers such that: (1) the secure execution cost is minimized and (2) a user-defined constraint on inference accuracy is met. COINN configurator initially reduces the optimization space, and then uses our customized optimizer and score function tofind the optimal DNN. The configurator performs the above process separately for quantization and matrix factorization. Below we explain each component of COINN configurator in detail.

Optimization Space Reduction. For quantization, the bitwidths for the input (b_{inp}), weights (b_w), and the activation (b_{acc}) should be configured at each linear layer. Finding the optimal quantized DNN is therefore equivalent to searching over a parameter space containing $B^{3\mathcal{L}}$ different network configurations where $\mathcal{L}$ and B denote the number of linear layers and the maximum bitwidth budget[3], respectively. Finding the best parameter configuration in such a large space is very time-consuming and thefi nal obtained DNN configuration is often sub-optimal. We observe that many of the bitwidth configurations in this search-space violate the user-defined accuracy constraint. Therefore, prior tofinding the optimal bitwidths, wefi rst identify and eliminate the invalid bitwidth configurations from the search space.

[3]In our experiments we set $B = 16$.

This process is performed on a per-layer basis: for each linear layer in the network, we discard the subset of its corresponding quantization bitwidths that violate the accuracy constraint. In doing so, we keep the remaining layers in full-precision format. Note that such per-layer analysis allows us to shrink the original search space but does not determine the optimal bitwidth configuration across all layers. This is due to the fact that the per-layer analysis does not reflect the effect of inter-layer correlations on inference accuracy when all DNN layers are simultaneously quantized. We therefore devise an optimizer to search the reduced parameter space obtained from the per-layer analysis tofi nd the optimal bitwidth configuration across all layers.

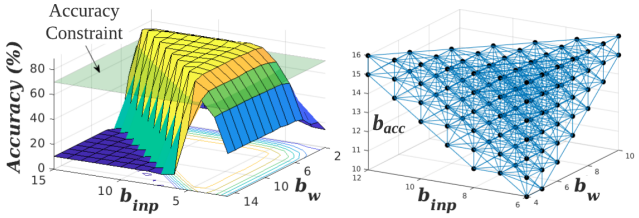

Figure 5: (left) Inference accuracy versus the input and weight bits of a CONV layer in an example DNN. (right) $3D$ visualization of the layer's valid bitwidth configurations.

Figure 5-(left) demonstrates the model accuracy versus the input and weight bitwidths for one layer of an example DNN when the activation bit is set to the maximum value ($b_{acc} = 16$). As seen, due to the occurrence of overflow, many of the configurations fall below the accuracy constraint plane. Using this intuition, we construct a $3D$ mesh of valid bitwidth configurations that comply with the accuracy constraint for each layer as shown in Figure 5-(right). Each node corresponds to a tuple (b_{inp}, b_w, b_{acc}) and its neighbors are nodes with a maximum bit distance of 1. As seen in this example, the search space for one layer is reduced to $\sim \frac{1}{8}$, which provides a lot of saving for the overall DNN, i.e., $\sim (\frac{1}{8})^{\mathcal{L}}$. Our optimizer then traverses this mesh tofi nd the optimal DNN configuration.

For clustering, the per-layer configuration comprises only one parameter, i.e., the unique size V, which undergoes a similar process for identifying the valid optimization space.

Optimizer. We develop a novel genetic algorithm [57] with customized graph operations to traverse our constructed mesh of valid configurations andfi nd the optimal quantized/clustered DNN. Our genetic algorithm operates on a *population* of *individuals* where each individual corresponds to a candidate DNN configuration. Optimization is performed iteratively and the population is gradually evolved to obtain better DNN configurations that have higher accuracy and/or lower secure execution cost. At each iteration, all members of the current population are evaluated in terms of the secure execution cost and the inference accuracy. We utilize a customized score function to combine these two (conflicting) metrics and assign a measure of optimality to each individual. We then perform a random selection from the population where individuals with higher scores have higher chances of being selected. Each selected individual is then randomly tweaked by moving along the configuration mesh to adjacent neighbor nodes. This is equivalent

to performing small-scale changes in the model architecture to explore new (unseen) configurations andfi nd the optimal DNN.

Score Function. The objective of parameter optimization for secure inference is to minimize the secure execution cost while enforcing the inference accuracy to be higher than a user-defined threshold (constraint). The objective of this constrained optimization can be embedded into a single score function that absorbs both accuracy and secure execution cost. Let us denote the DNN configuration (quantization/clustering parameters) as $\mathbf{p} \in \mathbb{R}^d$ and the corresponding accuracy and secure execution cost as $\mathcal{A}(\mathbf{p})$ and $C(\mathbf{p})$, respectively. For a given DNN configuration, the secure execution cost $C(\mathbf{p})$ is the cumulative per-layer costs calculated using Table 6 in Appendix D and the accuracy $\mathcal{A}(\mathbf{p})$ is measured on a held out validation dataset. We adapt the score function from ML-customization literature [30], which use fractions and an exponential penalty function [43] to enforce the inference accuracy constraint. Our score function is defined as:

$$\mathcal{S}(\mathbf{p}) = \frac{C_{max} - C(\mathbf{p})}{\xi(\mathcal{A}(\mathbf{p}))}, \tag{6}$$

where C_{max} is the execution cost of the reference DNN prior to optimization. The numerator of the score function encourages minimization of the ciphertext execution cost $C(\mathbf{p})$ and the denominator $\xi(\cdot)$ enforces a strict lower bound (threshold) for the accuracy using exponential penalty methods [7, 43] as follows:

$$\xi(\mathcal{A}(\mathbf{p})) = \begin{cases} \mathcal{A}_{max} - \mathcal{A}(\mathbf{p}) & \mathcal{A}(\mathbf{p}) > \mathcal{A}_{min} \\ \mathcal{A}_{max} - \mathcal{A}(\mathbf{p}) + e^{\mathcal{A}(\mathbf{p}) - \mathcal{A}_{min}} & otherwise \end{cases} \tag{7}$$

where $\mathcal{A}_{max}$ is the accuracy of the reference point DNN. As seen, $\xi(\cdot)$ puts a linear penalty on points with a high accuracy but exponentially increases the penalty when the accuracy drops below the lower bound $\mathcal{A}_{min}$. As we show in our experiments, this score function ensures that our genetic algorithmfi nds a DNN configuration that has significantly lower ciphertext cost compared to the baseline (plaintext) model with comparable accuracy.

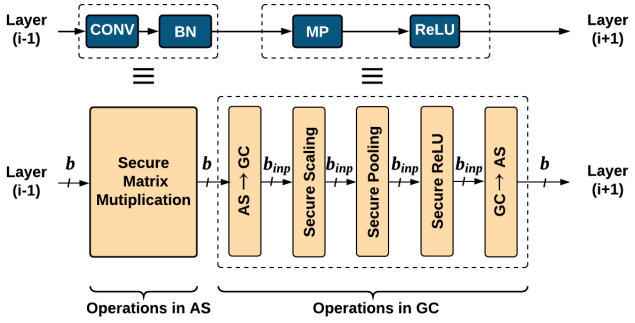

Figure 6: Plaintext operations and their equivalent ciphertext realization in COINN oblivious inference framework.

5 CRYPTOGRAPHIC PROTOCOLS

Figure 6 illustrates operations in plaintext DNNs and their corresponding secure computation in COINN framework. The linear layers – CONV and FC are executed through secure matrix-multiplication protocols in the AS domain. We provide protocols for both regular and factored matrix-multiplication for these two linear layers. We exploit the data repetition inherent in the computation of matrix products to achieve a significant reduction in the communication cost. The outputs of the linear operations in the AS domain are securely converted to the GC domain for computation of the non-linear layers – maxpool (MP) and ReLU. The scaling operation is also embedded into the AS to GC (and vice versa) conversion. Our design is optimized for the amortized setting where the same server-client pair runs multiple inferences without retraining the model, which is the common scenario in real-world applications.

Besides the aforementioned layers, COINN also supports batch normalization (BN) and average pooling (AP). These layers are fused into their preceding CONV/FC layers. Using this trick, heavy cryptographic operations such as the division protocol of CrypTFlow2 [46] can be avoided, allowing us to evaluate AP and BN at zero cost. Details of our layer fusion is enclosed in Appendix C.

5.1 Matrix-Multiplication

As explained in Section 3, the weight matrix W is only known by Alice, i.e., $[\![W]\!] = [\![W]\!]_A$ and $[\![W]\!]_B = \mathbf{0}$ while the activation matrix $[\![X]\!]$ is shared between Alice and Bob. We need to compute the matrix product $[\![Y]\!] = [\![W]\!] \cdot [\![X]\!] = [\![W]\!]_A \cdot [\![X]\!]_A + [\![W]\!]_A \cdot [\![X]\!]_B$. Since Alice can locally compute $[\![W]\!]_A \cdot [\![X]\!]_A$, we focus on secure computation of $[\![Z]\!] = [\![W]\!]_A \cdot [\![X]\!]_B$.

Regular Matrix-Multiplication. The product $[\![Z]\!] \in \mathbb{Z}_{2^b}^{M \times L}$ of $[\![W]\!]_A \in \mathbb{Z}_{2^b}^{M \times N}$ and $[\![X]\!]_B \in \mathbb{Z}_{2^b}^{N \times L}$ can be computed with MNL scalar multiplications. This approach requires $MNLb$ invocation of COT_b^2 [16], thus incurring a communication cost of $MNLb(\kappa + b)$ bits. The communication cost of one instance of COT_λ^2 is $\kappa+\lambda$, where thefi rst term (the cost of one ROT_λ^2) is independent of the message bitwidth λ. Since the computation of the matrix product involves dot product of each row of $[\![W]\!]_A$ with L columns of $[\![X]\!]_B$, we compute the matrix product with MNb invocations of COT_{Lb}^2. This approach reduces the number of COTs by increasing the message length, thereby reducing the total communication cost to $MNb(\kappa + Lb)$. Compared to the protocols with independent multiplications [16, 35], the cost is reduced by $\frac{L(\kappa+b)}{\kappa+Lb}$. This cost can be further reduced in the amortized setting as will be explained in Section 5.2.

Factored Matrix-Multiplication. We now present our protocol for securely computing the factored matrix-multiplication explained in Section 4.2. Wefi rst define the one-hot encoded representation of a vector.

Definition 5.1. Given a vector $\mathbf{w} \in \mathbb{R}^N$ and its coded representation $\widetilde{\mathbf{w}} \in [V]^N$ w.r.t. its unique space $\mathbf{c} = \{c_1, \ldots, c_V\}$, the one-hot encoded representation of $\widetilde{\mathbf{w}}$ is a matrix $\widetilde{W} \in \{0,1\}^{V \times N}$ such that $\widetilde{W}[v, n] = 1$ if $\widetilde{w}[n] = v$ and 0 otherwise $(\forall v \in [V], n \in [N])$.

We will be using the following notations to explain our secure factored matrix-multiplication:

- The collection of unique spaces for all rows of $[\![W]\!]_A$: $\left\{[\![\mathbf{c}]\!]_A^{(m)} \in \mathbb{Z}_{2^b}^V\right\}_{m \in [M]}$
- The collection of one hot encodings of all rows of $[\![W]\!]_A$ w.r.t. $[\![\mathbf{c}]\!]_A^{(m)}$: $\left\{[\![\widetilde{W}]\!]_A^{(m)} \in \{0,1\}^{V \times N}\right\}_{m \in [M]}$
- Partial sum: $\left\{[\![S]\!]_A^{(m)} \in \mathbb{Z}_{2^b}^{V \times L}\right\}_{m \in [M]}$

Using the above notations, the product $[\![Z]\!] = [\![W]\!]_A \cdot [\![X]\!]_B$ is computed as:

$$[\![S]\!]_A^{(m)}[v,l] = \sum_{n=1}^{N} [\![\widetilde{W}]\!]_A^{(m)}[v,n] \cdot [\![X]\!]_B[n,l]; \quad \forall m \in [M], v \in [V], l \in [L] \tag{8}$$

$$[\![Z]\!][m,l] = \sum_{v=1}^{V} [\![\mathbf{c}]\!]_A^{(m)}[v] \cdot [\![S]\!]^{(m)}[v,l]; \quad \forall m \in [M], l \in [L] \tag{9}$$

Eq. 8 represents conditional accumulation and Eq. 9 represents dot product of length V vectors of b-bit integers. Note that the number of integer multiplications is reduced from MNL in regular matrix-multiplication to MVL in the factored version ($V \ll N$). The majority of the cost is now incurred by the conditional accumulation, which is computed through COT. We leverage our optimization presented for the regular matrix-multiplication, i.e., merging the COT messages involving the same selector bit, for both Eq. 8 and 9 to reduce the communication cost.

Algorithm 1 presents the protocol for computing the partial sums through conditional accumulation. Since the protocol requires MVN COT^2_{Lb}, the communication cost of computing the partial sums is $MVN(\kappa + Lb)$. The dot product of Eq. 9 can then be computed following the technique presented in MiniONN [35] with a communication cost of $MVb(\kappa + Lb)$. Thus the total cost of computing factored matrix-multiplication is $MV(N + b)(\kappa + Lb)$. Since in practice, $b \ll N$, we approximate the cost as $MVN(\kappa + Lb)$.

Proof Sketch. The security proof of Algorithm 1 directly follows from the security guarantee of OT. Observe that all the communication between Alice and Bob is performed through OT which ensures the privacy of both the selection bits and messages. Moreover, the correlation function chosen by Bob ensures that Alice never receives an unmasked version of any element of $[\![X]\!]_B$. Furthermore, every instance of OT involves freshly generated unique masks that ensures the security of the one-time pad.

5.2 Linear Layers in the Amortized Setting

The mean communication cost of computing both regular and factored matrix-multiplication is further reduced in the amortized setting where one server-client pair runs a large number of inferences with the same trained model but different inputs, i.e, W remains constant while X changes in each inference. In case of regular matrix-multiplication, since, W does not change, the number of COTs remains the same while message length increases to JLb, where J is the number of inferences. The mean cost per matrix-multiplication is therefore $MNb(\kappa + JLb)/J \approx MNLb^2$ for large J. Similarly, for factored matrix-multiplication, the mean amortized cost is $MVNLb$. More importantly, in this setting, the number of communication rounds remains constant (= 2), irrespective of the number of inferences J. Our protocol execution is split into setup, offline and online phases as described below.

Setup Phase. This is performed once per server-client pair irrespective of the number of inferences J. In this phase, for regular matrix-multiplication, Alice and Bob perform the MNb ROT^2_{JLb} as part of the MNb COT^2_{JLb} for matrix-multiplication computation. In practice, following the state-of-the-art OT libraries [48, 55], Alice receives MNb κ-bit seeds $\gamma_q; \forall q \in [MNb]$ and Bob receives κ-bit seeds γ_{0q} and $\gamma_{1q}; \forall q \in [MNb]$ which are later expanded to b-bit messages $\forall j \in [J], l \in [L]$ through Cryptographically Secure Pseudo Random Number Generator (CS-PRNG). This makes sure that the memory requirement is independent of the number of inferences J. The communication cost of the setup phase is $MNb\kappa$. Similarly, the communication cost of the setup phase for factored matrix-multiplication is $MV(N + b)\kappa$.

Offline and Online Phases. These two phases are performed once per inference j. The offline and online phases involve computation *before* and *after* the input X is available, respectively. We employ the technique proposed by Slalom [53], to ensure that most of the cost corresponds to the offline phase. In this technique, in the offline phase, Alice and Bob securely compute the matrix product $[\![Z']\!] = [\![W]\!]_A \cdot [\![U]\!]_B$, where $[\![U]\!]_B \in \mathbb{Z}_{2^b}^{N \times L}$ is a random matrix generated and known by Bob. They locally expand the seeds obtained in the setup phase for the particular inference index j and for each column $l \in [L]$ of X and completes the COT^2_{Lb}. The communication cost of this phase for each $j \in [J]$ for regular and factored matrix-multiplications are $MNLb^2$ and $MVNLb$ respectively. In the online phase, Bob directly sends $F = [\![X]\!]_B - [\![U]\!]_B$ to Alice who locally computes $[\![Z]\!]_A = [\![Z]\!]_A + [\![W]\!]_A[\![F]\!]_A$. Bob sets $[\![Z]\!]_B = [\![Z']\!]_B$. The communication cost in this phase negligible compared to that of the offline phase.

Number of Communication Rounds. In the proposed setting, the only communication from Alice to Bob occurs in the setup phase. The offline and online phases involve communication from Bob to Alice, only. Thus the number of communication rounds is 2, irrespective of the number of inferences J. This reduces the adverse effect of increased network latency in the Wide Area Network (WAN) setting.

Switching between Regular and Factored Multiplication. Based on the optimal unique size allocated to each layer's weights by the model configurator, our protocol automatically switches between regular and factored multiplication to maximize efficiency. Switching is done when the costs of both multiplications are equal, i.e., $MNLb^2 = MVNLb$ in the amortized setting, which renders the switching point $b = V$, i.e., when the number of unique values in each row of the weight matrix is equal to the AS shares' bitwidth.

5.3 Non-linear Layers

COINN GC domain incorporates the following four stages:

(i) AS to GC Conversion. A variable $[\![x]\!] \in \mathbb{Z}_{2^b}$ shared between Alice and Bob through AS is securely converted to its share in the GC domain by securely computing the addition function through GC with inputs from Alice, the garbler and Bob, the evaluator as $[\![x]\!]_A$ and $[\![x]\!]_B$, respectively. Before the addition, Bob obtains the Yao share for his input $[\![x]\!]_B$ through COT, which requires two rounds of communication. In this particular scenario, Bob's share is generated through multiplication in the AS domain. According to the multiplication technique described in Section 2.3, his shares are independent of his input. Therefore, we perform the COTs for all the layers in parallel during the offline phase, which reduces one round of communication for each layer in the online phase.

Algorithm 1: Protocol for Computing Conditional Accumulation

Input : From Alice, one-hot encoding of weight matrix $[\![W]\!]_A$: $\left\{[\![\widetilde{W}]\!]_A^{(m)} \in \{0,1\}^{V\times N}\right\}_{m\in[M]}$

Input : From Bob, share of the activation $[\![X]\!]$: $[\![X]\!]_B \in \mathbb{Z}_{2^b}^{N\times L}$:

Output : Partial sum $\left\{[\![S]\!]^{(m)} \in \mathbb{Z}_{2^b}^{V\times L}\right\}_{m\in[M]}$

OT message received by Alice, $\left\{\mu'^{(l)} \in \mathbb{Z}_{2^b}^{M\times V\times N}\right\}_{l\in[L]}$

OT message received by Bob, $\left\{\mu^{(l)} \in \mathbb{Z}_{2^b}^{M\times V\times N}\right\}_{l\in[L]}$

1 Bob chooses a set of correlation functions $\varphi_{m,v,n}^{(l)}(\cdot)$ as $\left\{\varphi_{m,v,n}^{(l)}(\mu^{(l)}[m,v,n])\right\}_{l\in[L]} = \left\{\mu^{(l)}[m,v,n] + [\![X]\!]_B[n,l]\right\}_{l\in[L]}; \forall m \in [M], v \in [V], n \in [N]$

2 **foreach** $m \in [M], v \in [V], n \in [N]$ **do**

3 Alice and Bob run COT_{Lb}^2 where

Bob acts as sender with correlation functions $\left\{\varphi_{m,v,n}^{(l)}(\cdot)\right\}_{l\in[L]}$ and receives $\left\{\mu^{(l)}[m,v,n]\right\}_{l\in[L]}$

Alice acts as receiver with choice bits $[\![\widetilde{W}]\!]_A^{(m)}[v,n]$ and receives $\left\{\mu'^{(l)}[m,v,n]\right\}_{l\in[L]} = \left\{\mu^{(l)}[m,v,n] + [\![\widetilde{W}]\!]_A^{(m)}[v,n]\cdot[\![X]\!]_B[n,l]\right\}_{l\in[L]}$

4 Alice sets $[\![S]\!]_A^{(m)}[v,l] = \sum_{n=1}^{N}\left(\mu'^{(l)}[m,v,n]\right); \forall m \in [M], v \in [V], l \in [L]$

Bob sets $[\![S]\!]_B^{(m)}[v,l] = \sum_{n=1}^{N}\left(\mu^{(l)}[m,v,n]\right); \forall m \in [M], v \in [V], l \in [L]$

(ii) Scaling. As a result of the optimizations presented in Section 4.1, scaling is performed through bit shift, which can be evaluated in GC with no additional communication cost. Scaling converts the bitwidth of the shared variables from b to b_{inp}, where, b_{inp} is the input bitwidth of the next CONV/FC layer. This approach significantly reduces the GC execution cost for nonlinear layers.

(iii) MaxPool and ReLU. An MP operation with a window size of $k \times k$ requires $k^2 - 1$ comparison and multiplexing operations, each of which incurs a communication cost of $2 \cdot \kappa \cdot b_{inp}$ bits. Note that MP (ReLU (x))=ReLU (MP (x)), thus we perform MP before ReLU as it shrinks the size of the activation tensor by a factor of k^2, thereby reducing the ReLU cost. Each ReLU includes b_{inp} AND operations requiring $2 \cdot \kappa \cdot b_{inp}$ bits of communication.

(iv) GC to AS Conversion. For this operation, Alice generates a random b-bit integer which is added to the ReLU output and the sum is revealed to Bob. This operation does not require COT since there is no input from Bob. During conversion, the values are sign-extended to b bits to match the AS ring size. It is worth noting that our computations in the AS domain are performed modulo 2^b and the GC circuit for b-bit addition automatically takes care of the modulo operation [16]. On the contrary, to benefit from SIMD operations in HE, the modulus is chosen as a prime number. Therefore, the circuit for modular addition requires b-bit addition, subtraction, and multiplexing, thereby increasing the cost of HE-GC conversion [31]. In summary, the HE-GC conversion (as seen in [31]) requires $\sim 2.5\times$ more computation/communication compared to AS-GC conversion (as seen in [16]).

5.4 Cost Breakdown and Comparison with Previous Works

To explain the source of runtime improvement in the proposed method, we summarize the cost complexity of different phases of execution of the linear layers in COINN and compare them with prior work in Table 1. For HE-based works, the complexity refers only to the computation cost. For OT-based works, the complexity refers to both communication and computation cost while communication is usually the dominant factor. The number of communication rounds for all works is equal to the number of layers, except for XONN which has constant number of rounds. The performance gains of COINN over prior work stem from two main reasons:

Table 1: Cost break down of different phases of linear layers in COINN and previous works. N_{slot} is the number of slots in vectorized HE operations. $CostMult(q)$ is the cost of one scalar multiplication in $\mathcal{Z}_q$ in HE. q the cipher-text modulus which is $\sim 3\times$ larger than plain-text modulus $p \approx 2^{b_{acc}}$.

Work	Per-layer Complexity	
	One time setup	Per-inference
Gazelle/Delphi/CTF2 (HE)	-	$O(\frac{MNL}{N_{slot}}).CostMult(q)$
MiniONN/CTF2 (OT)	-	$O(MNb_{acc}(\kappa + Lb_{acc}))$
XONN (GC)	-	$O(MNLb_{acc}^2\kappa)$
COINN – regular (OT)	$O(MNb_{acc}\kappa)$	$O(MNLb_{acc}^2)$
COINN – factored (OT)	$O(MVb_{acc}\kappa)$	$O(MVLb_{acc}(N + b_{acc}))$

- **Separating setup time.** We move a large part of the computation/communication of the (OT-based) linear layers of COINN to a one-time setup phase without affecting security. In contrast, previous OT-based methods (MiniONN [35], CrypTFlow2 [33]) repeat these operations for every inference [4]. Moreover, separation of setup and per-inference phases is not readily applicable in the HE-based methods (Gazelle [31], Delphi [41], CrypTFlow2 [33]) or GC-based methods (XONN [47]).

[4]The preprocessing phase of Delphi [41] is equivalent to our offline phase and needs to be repeated per inference to ensure security.

- **Optimizing parameters.** Prior works use a large butfi xed bitwidth b_{acc} for all linear layers. COINN customizationfinds smaller values of b_{acc} that vary from one layer to another, significantly reducing the secure execution cost while preserving the accuracy. Additionally, our factored matrix multiplication can further reduce the execution cost of linear layers when $V < b_{acc}$. Note that by reducing the computational bitwidth, COINN also reduces the cost of protocol conversion and nonlinear layers as formalized in Table 6.

6 RELATED WORK

In this section, we review the related work that employ similar settings as ours, i.e., cryptographically secure two-party protocols where the server owns the model and the client owns the input. There are two classes of techniques: Homomorphic Encryption (HE) [21], which is heavy on computation and Multi-Party Computation (MPC) techniques such as Garbled Circuits (GC) [59] and Arithmetic Sharing (AS) [6], which are heavy on communication.

CryptoNets [23] is perhaps the pioneer of 2-party oblivious inference. More efficient variants and compilers have since been proposed for optimized DNN inference [10, 11, 13, 15, 26, 50]. HE-based methods such as [23] allow outsourcing the majority of the computations to the more capable party, i.e., the server. However, frameworks that are entirely based on HE replace the nonlinear activations with HE-friendly polynomial approximations, resulting in reduced inference accuracy. Oblivious inference based on GC has also been proposed [8] which provides better accuracy but suffers from long run times due to the large communication cost of multiplications in GC. To mitigate this, XONN [47] presents a GC-based framework for Binarized Neural Networks (BNN) where all multiplications are replaced with cost-free XNOR operations. Nevertheless, the binary weights and activations in a BNN have an adverse effect on the inference accuracy.

At present, most efficient secure inference engines employ a hybrid approach –using the most efficient cryptographic primitive for a particular layer. MiniONN [35] employs a combination of AS, GC, and HE. Follow-up works Gazelle [31] and Delphi [41], support efficient HE-based linear operations along with GC-based nonlinear functions, and perform secure protocol conversion when necessary [31, 35]. Subsequent works [34, 47] have pointed out security vulnerabilities in HE-based methods, safeguarding against which would result in increased runtime. CrypTFlow2 [46] proposes a hybrid protocol that supports both HE and AS-based linear layers and has custom protocols for secure comparison (used in ReLU and MP) which incur less communication at the cost of higher number of communication rounds compared to GC.

A parallel line of work in oblivious inference focuses on applying optimizations to reduce the secure execution cost of previously proposed security protocols. The contributions in this domain can be categorized in two separate directions: (1) adjusting the parameters for the secure protocol, and (2) changing the DNN architecture for improved secure execution. In thefi rst category, recent work [9, 39] adjust the HE parameters for hybrid HE-GC protocols, i.e., Gazelle and Delphi, to reduce the secure execution cost. The methods in the second category [22, 38, 41] reduce the number of ReLU activations throughout the network to reduce the GC communication and runtime in hybrid HE/AS and GC protocols.

Perhaps the most related model-adjustment techniques to COINN are the quantization in [2, 14]. These works have two major differences with COINN quantization. Firstly, they simply use homogeneous bitwidths for all DNN weights/activations. We show that by solving the challenging problem of heterogeneous bitwidth selection, secure execution cost can be significantly lowered without hurting model accuracy. Secondly, the aforesaid works use the available quantization schemes optimized for the plaintext domain [29, 56], while COINN develops a new cipher domain optimized quantization scheme that replaces costly quantization operations with variants that incur a negligible GC cost. For example, we leverage logarithmic representation for the quantization scale in Eq. 4. It is worth mentioning that our quantization is different than [42], which applies logarithmic encoding to the matrix multiplication operands themselves. By doing so, [42] replaces multiplications with bit-wise shifts and conditional additions. However, the ciphertext computation corresponding to bit-wise shift is not efficiently realizable in AS. Hence, unlike [42], we usefi xed-point representation for our matrix multiplication operands (layer inputs and weights) to keep them consistent with the AS domain. After multiplication, the result is converted to GC to perform scaling and nonlinear operations. Since bit-wise shifting is free in GC, we enforce our quantization scale to be a power of two using the logarithm operation in Eq. 4.

COINN bridges the gap between protocol design and ML model adjustment to optimize the ciphertext execution of both linear and non-linear operations. Compared to works that only optimize the cryptographic protocols [31, 35, 46], the contributions of our work lie in designing security-aware low-bit quantization and introduction of factored multiplication and its accompanying custom secure execution protocol. Compared to works that optimize the ML model [22, 38, 39, 41, 47], our model adjustment techniques are scalable to many-layer architectures trained for complex tasks such as ImageNet. Additionally, COINN quantization and factored multiplication together with our automated parameter configurator achieve a better accuracy-runtime tradeoff compared to prior model adjustment methods such as modifying ReLU layers [22, 38, 41].

7 EXPERIMENTS

In this section, we empirically evaluate the performance of COINN in various settings. We perform a detailed study of the efficiency gains achieved by each of COINN optimizations, namely, quantization, clustering, and end-to-end parameter configuration, in Section 7.1. Next, we provide a side-by-side comparison of COINN with recent works in Section 7.2, in terms of the ciphertext execution time, showing 4.7×–36.8× faster inference on contemporary DNNs in LAN setting. We further show that COINN achieves better performance compared to prior work in the high-latency setting.

Evaluation Setup. We use the PyTorch library for training the FP32 DNNs and develop our security-aware quantization, clustering, and automated parameter configuration with PyTorch backend for easy utilization by the community. Our ciphertext execution uses OT, and CS-PRNG implementations from EMP-toolkit [55] and GC implementation from TinyGarble2 [27]. For fast matrix-multiplication, we utilize the Intel intrinsic instructions and represent matrices with the Eigen library [24].

Table 2: COINN benchmarks.

Model	Layers	Acc	MACs	Params
MiniONN [35]	6 CONV, 1 FC, 2 MP, 6 ReLU	88.3	6.1e7	1.6e5
ResNet32	31 CONV, 1 FC, 1 AP, 31 ReLU	68.7	6.9e7	4.7e5
ResNet110	109 CONV, 1 FC, 1 AP, 109 ReLU	94.1	2.5e8	1.7e6
ResNet50	49 CONV, 1 FC, 1, MP, 1 AP, 49 ReLU	76.1	4.1e9	2.5e6

We run our ciphertext evaluations using 4 threads on machines with 2.2 GHz Intel Xeon CPU and 16 GB RAM. For runtime measurements, we consider two real-world network settings, namely LAN with a throughput of 1.25 GBps, round trip time of 0.25ms, and WAN with a throughput of 125 MBps, round trip time of 100ms. We simulate the network settings via Linux Traffic Control[5].

Benchmarks. We perform evaluations on the CIFAR-10, CIFAR-100, and ImageNet classification benchmarks. The number of classes in these datasets is 10, 100, and 1000, respectively. Table 2 presents details of our benchmarked DNNs along with their FP32 accuracy. We evaluate the 7-layer network from MiniONN [36] and ResNet110 on CIFAR-10, ResNet32 on CIFAR-100, and ResNet50 on ImageNet dataset. Our DNN benchmarks cover a wide range of parameter sizes (0.5M to 23M) and number of MAC operations (60M to 4B) commonly observed in real-world models.

Accuracy Measurement. Throughout the evaluations, we report the secure model accuracy, which is measured efficiently (and correctly) by simulating ciphertext operations in PyTorch. The correctness is validated by matching all DNN layers' activations in secure inference with those from PyTorch on randomly selected inputs.

7.1 Evaluation of COINN Optimizations

In this section, we provide a breakdown of the savings in secure execution cost as a result of COINN's model adjustment methods and protocol optimization.

Low-Bit Heterogeneous Quantization. We illustrate the benefits of our quantization scheme in reducing the secure communication cost, while maintaining accuracy, for a large scale real-world DNN – ResNet32. Figure 7 presents the communication cost and accuracy of secure execution as a function of the bitwidth. The numerical labels on the horizontal axis represent homogeneous quantization (equal bitwidths across all layers), where each label is $b_{inp} = b_w$ with b_{acc} set to $2b_{inp} + 1$. The label 16 represents the configuration implemented in prior works [41, 46] which we use as a baseline. Figure 7 shows that while reducing the bitwidth in the homogeneous setting results in a linear reduction of ciphertext communication, it also results in a significant drop in accuracy.

To mitigate the undesireable accuracy drop of homogeneous quantization, our automated parameter configuratorfi nds a heterogeneous allocation of per-layer bitwidths that simultaneously ensures high accuracy and low communication cost. The rightmost label, Q in Figure 7, represents the COINN optimized model with heterogeneous quantization bitwidths across layers. This optimal set of bitwidths results in a communication cost equivalent to the 6-bit homogeneous model and achieves an accuracy comparable to the 16-bit baseline. Such optimization of per-layer bitwidths is made possible via our secure computation-aware quantization which accurately simulates the effect of low-bit quantization in ciphertext. This allows us to explore the trade-off between communication cost and model accuracy. We present the heterogeneous bitwidths found by COINN configurator for ResNet32 in Figure 8-a.

[5]https://man7.org/linux/man-pages/man8/tc.8.html

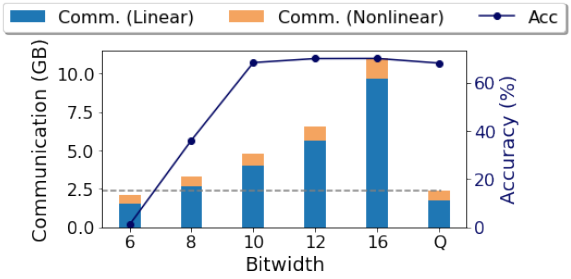

Figure 7: Effect of quantization bitwidth on communication cost (bars) and accuracy (curve). The numbers on the horizontal axis show the bitwidth for homogeneous quantization of weights/inputs across all layers. Label Q represents the heterogeneous bitwidths found by COINN.

Factored Matrix-Multiplication. Figure 7 shows that the bulk of total communication cost in a quantized model corresponds to linear operations. We now showcase how COINN further reduces this cost via factored matrix-multiplication. Figure 10 presents the communication cost and accuracy as a function of the number of unique elements in each layer's weight matrices V. The label Q represents our model with heterogeneous quantization bitwidths from Figure 7. The numeric labels to its left represent models with a uniform selection of V across all layers. Such naïve selection results in accuracy degradation, particularly for small V. Our automated parameter configuratorfi nds a heterogeneous allocation of V across DNN layers that balances the tradeoff between inference accuracy and ciphertext communication. The result is an optimal DNN represented with the label Q+C that reduces the secure communication cost of the quantized model by 1.4× while maintaining the original model accuracy. We present the heterogeneous number of per-layer clusters found by our configurator for this benchmark in Figure 8-b.

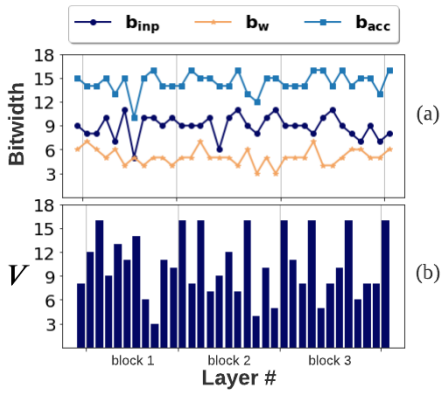

Figure 8: Heterogeneous parameters across ResNet-32 layers found by COINN configurator. (a) Quantization bitwidths. (b) Number of clusters V.

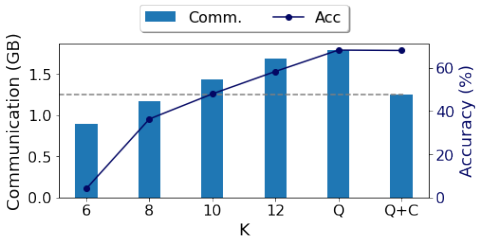

Figure 10: Effect of factored multiplication on inference accuracy and communication cost of linear operations. The label Q on the horizontal axis shows the baseline quantized DNN. The numbers to its left represent the homogeneous V used to cluster all layer weights. The label Q+C stands for the heterogeneous V configuration found by COINN.

Holistic Optimization. Figure 9 presents the reduction in communication cost achieved by applying COINN automated quantization and clustering on all benchmarks. As our baseline design, we adopt the bitwidths from prior work [41], i.e., 16-bit inputs/weights and 32-bit activations, and perform regular matrix-multiplication. For COINN results, wefi rstfi nd heterogeneous quantization configurations using our genetic algorithm andfi ne-tune the model to regain accuracy. We show the optimized quantized model via Q on Figure 9. Next, we use our automated parameter configurator to find the weight clusters for each layer andfi ne-tune the resulting model once more to obtain Q+C. The linear operations in the Q and Q+C are performed via regular and factored Matrix-Multiplication, respectively. As seen, byfi nding the best set of heterogeneous bitwidths across DNN layers, COINN successfully reduces the secure communication for linear and nonlinear layers by 3.9×–4.3× and 1.9×–2.2×, respectively. By optimizing the weight clusters, we further improve the efficiency of linear layers by 4.8×–8.1×.

Table 3 provides the total runtime and communication cost of our baseline, Q, and Q+C configurations in both LAN and WAN settings. The evaluation verifies the effect of our optimization on the runtime: applying Q+C reduces the baseline runtime by 2.6×–3.9× and 2.3×–4.2× in LAN and WAN settings, respectively. The effect of COINN optimizations on standalone micro-benchmarks of the CONV and ReLU is presented in Appendix E.

Setup Time Separation. Finally, we evaluate the effect of introducing the one-time setup phase to reduce the amortized per-inference cost. The setup phase is only performed thefi rst time a connection is established between the client and server and is independent of the number of inferences. In the previous section (Table 1), we showed the complexity of linear layers in the setup and per inference phases. We now show the effect of this optimization through experimental evaluation. Figure 11 presents the breakdown of setup time and amortized inference time for each of the four benchmarks under LAN and WAN settings. As expected, separating the setup time from oblivious inference significantly reduces the runtime.

Table 3: Evaluation of COINN in LAN and WAN settings. Q and C denote quantization and clustering, respectively.

Model	Comm. (GB)			LAN Time (s)			WAN Time (s)		
	Base	Q	Q+C	Base	Q	Q+C	Base	Q	Q+C
MiniONN	8.7	2.3	1.0	4.85	1.9	1.45	74.6	26.5	18.5
Res32	10.4	2.4	1.9	9.8	3.8	3.68	143.9	67.1	62.9
Res110	37.6	9.7	6.8	36.0	14.2	14.0	518.1	242.8	226.0
Res50	583.1	148.0	122.0	571.46	165.3	145.7	4994	1420.4	1189.7

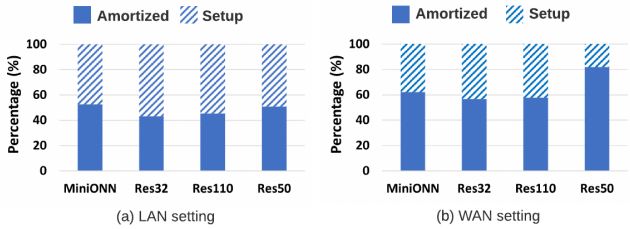

Figure 11: Breakdown of setup and amortized times for the under LAN and WAN settings.

7.2 Comparison with Prior Work

In this section, we compare COINN amortized runtime with the prior art in oblivious inference. In Table 4, we report the performance of COINN along with four contemporary works, namely, XONN [47] with extremely low-bit (binary) weights/activations, Delphi [41] with a hybrid HE-GC protocol, SafeNet [38] which perform ML optimization for Delphi's secure protocol, and CrypTFlow2 [46] which is the current state-of-the-art in oblivious inference. For a fair and accurate comparison, we re-run the open-source codes provided by Delphi[6] and CrypTFlow2[7] to obtain runtime/communication measurements on our machines. For the remaining works [38, 47], we directly report the numbers from the original papers since no public code was available.

[6]https://github.com/mc2-project/delphi
[7]https://github.com/mpc-msri/EzPC/tree/master/SCI

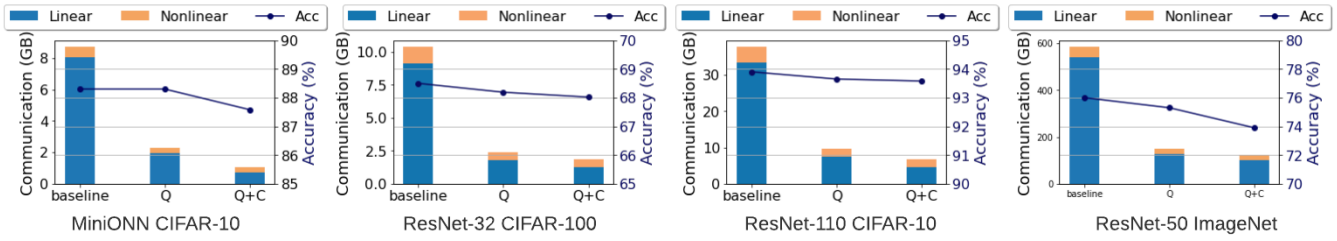

Figure 9: Communication for baseline and COINN optimized models, where Q represents quantized model and Q+C further applies clustering to enable factored multiplication.

Table 4: Performance comparison of COINN with best prior work. "Improv." shows the improvement in total runtime. CTF2 refers to CrypTFlow2 [46].

		LAN		WAN		Acc.
		Runtime (s)	Improv.	Runtime (s)	Improv.	(%)
MiniONN	XONN	33.5	23.1×	-	-	83.0
	Delphi	49.9	34.4×	59.8	3.2×	82.9
	SafeNet	53.4	36.8×	-	-	85.1
	CTF2 (HE)	20.8	14.4×	55.4	3.0×	86.0
	CTF2 (OT)	11.9	8.2×	108.2	5.8×	86.0
	COINN	1.45	1×	18.5	1×	87.6
Res32	Delphi	88.8	24.0×	145.9	2.3×	65.7
	SafeNet	128.0	34.6×	-	-	67.5
	CTF2 (HE)	32.6	8.8×	136.9	2.2×	68.0
	CTF2 (OT)	18.7	5.1×	176.7	2.8×	68.0
	COINN	3.7	1×	62.9	1×	68.1
Res110	CTF2 (HE)	110.3	7.8×	448.2	2.0×	94.1
	CTF2 (OT)	65.4	4.7×	579.3	2.6×	94.1
	COINN	14.0	1×	226.0	1×	93.4
Res50	CTF2 (HE)	893.2	6.1×	1463.3	1.2×	76.1
	CTF2 (OT)	1139.8	7.8×	4241.8	3.6×	76.1
	COINN	145.7	1×	1189.7	1×	73.9

Table 4 shows COINN achieves 4.7×–36.8× faster ciphertext execution in the LAN setting compared to prior work. Even though in the high latency setting the benefit margins are smaller, COINN still outperforms the best methods to date. This is achieved by optimizing both non-linear and linear computations/communications through quantization and factored multiplication. Furthermore, COINN achieves 0.6%– 4.7% higher accuracy with 23.1×–36.8× faster secure runtime compared to prior crypto/ML co-optimization work, namely [38, 41, 47].

Evaluation on Large-scale Benchmarks. To fully demonstrate the efficacy and scalability of COINN model adjustment techniques and custom secure protocols, we evaluate two exceptionally complex DNNs, namely, ResNet110 on CIFAR-10 and ResNet50 on ImageNet datasets. Thefi rst benchmark, i.e., ResNet110, is challenging due to the extremely high dimensionality of the parameter configuration space: there are 330 bitwidths and 110 clustering parameters that require per-layer adjustment. The second benchmark, i.e., ResNet50, is the largest DNN ever studied in the secure computation domain with over 4 Billion scalar multiplications and additions.

In Table 4, we present the runtime for the large scale networks and compare our results with the state-of-the-art CrypTFlow2. In the LAN setting, COINN achieves 4.7×–7.8× and 6.1×–7.8× runtime improvement compared to CrypTFlow2's OT-based and HE-based implementations, respectively. In the WAN setting, COINN achieves 2.6×–3.6× and 1.2×–2× runtime improvement compared to CrypTFlow2's OT-based and HE-based implementations, respectively. It is worth noting that the relatively lower improvement margin achieved by COINN in one specific setting (1.2× for ResNet50, WAN, HE) is due to the heavy imbalance of the cost towards linear layers in this particular benchmark.

7.3 Model Customization Runtime

COINN plaintext model customization (Section 4) is a one-time process performed on the pre-trained model by the server irrespective of the number of inference or the number of clients. Table 5 outlines the runtime of each customization step on one GPU, across various benchmarks. For better comparison, we normalize the customization andfi ne-tuning runtimes by the time required for training the baseline DNN on the same hardware. Forfi ne-tuning, the number offi ne-tuning epochs for each model is determined such that the validation accuracy reaches a convergence plateau. Regarding model customization, we terminate the genetic algorithm when the best obtained score does not improve for more than 5 iterations. Note that COINN customization step enjoys a linear speedup as the number of GPU cores increases. This is due to the independence of score evaluations inside a population [30].

Table 5: Runtime of COINN model customization andfine-tuning, normalized by the target DNN's training time on one NVIDIA Titan XP GPU. Here, Q and C denote the quantization and clustering stages, respectively.

Model	Training	COINN	Customization		Fine-tuning	
	(minutes)	Steps	# iter	Runtime	# iter	Runtime
MinioNN	11.6	Q	30	2.29×	20	0.50×
		C	20	2.12×	5	0.14×
Res32	34.2	Q	20	1.48×	20	0.56×
		C	30	1.94×	20	0.65×
Res110	107.6	Q	30	1.89×	5	0.14×
		C	30	1.43×	20	0.59×
Res50	14,040.0	Q	20	0.05×	5	0.14×
		C	30	0.16×	2	0.07×

8 CONCLUSION AND FUTURE WORK

We present COINN, an oblivious DNN inference framework that outperforms state of the art in both accuracy and efficiency. Through a unique combination of complimentary optimizations in ML and crypto domains, COINN brings us one step closer to real life deployment of contemporary DNNs in the privacy-preserving setting. The enhanced performance of COINN roots in three innovations, namely, ciphertext-aware quantization, enhanced data reuse, and automated parameter configuration. Our contributions in the plaintext are accompanied by efficient custom cryptographic protocols. We performed rigorous empirical analysis on every step of our optimization process to demonstrate their effect on reducing the secure communication and oblivious inference runtime. Our evaluations on practical DNN benchmarks showed an end-to-end runtime speedup of 4.7×–14.4× over the best prior work.

While this paper considers oblivious inference, optimizing oblivious training is an interesting future direction. Our amortized AS-based matrix-multiplication can benefit linear operations in forward and backward passes during training. However, remaining challenges for training include adjusting scales during training, secure quantization customized for training, and exploring the impact of model updates on data privacy.

REFERENCES

[1] Martín Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. *arXiv preprint arXiv:1607.00133* (2016).

[2] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J Kusner, and Adrià Gascón. 2019. QUOTIENT: two-party secure neural network training and prediction. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1231–1247.

[3] Abdalla Alameen and Ashu Gupta. 2020. Optimization driven deep learning approach for health monitoring and risk assessment in wireless body sensor networks. *International Journal of Business Data Communications and Networking (IJBDCN)* 16, 1 (2020), 70–93.

[4] Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. 2015. Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning. *Nature biotechnology* 33, 8 (2015), 831.

[5] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. 2013. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 535–548.

[6] Mikhail Atallah, Marina Bykova, Jiangtao Li, Keith Frikken, and Mercan Topkara. 2004. Private collaborative forecasting and benchmarking. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*. 103–114.

[7] Thomas Bäck, David B Fogel, and Zbigniew Michalewicz. 1997. *Handbook of evolutionary computation*. CRC Press.

[8] Marshall Ball, Brent Carmer, Tal Malkin, Mike Rosulek, and Nichole Schimanski. 2019. Garbled Neural Networks are Practical. *IACR Cryptol. ePrint Arch.* 2019 (2019), 338.

[9] Song Bian, Masayuki Hiromoto, and Takashi Sato. 2019. DArL: Dynamic parameter adjustment for LWE-based secure inference. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1739–1744.

[10] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. 2018. Fast homomorphic evaluation of deep discretized neural networks. In *Annual International Cryptology Conference*. Springer, 483–512.

[11] Alon Brutzkus, Ran Gilad-Bachrach, and Oren Elisha. 2019. Low latency privacy preserving inference. In *International Conference on Machine Learning*. PMLR, 812–821.

[12] Qingrong Chen, Chong Xiang, Minhui Xue, Bo Li, Nikita Borisov, Dali Kaarfar, and Haojin Zhu. 2018. Differentially private data generative models. *arXiv preprint arXiv:1812.02274* (2018).

[13] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. 2018. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *arXiv preprint arXiv:1811.09953* (2018).

[14] Anders Dalskov, Daniel Escudero, and Marcel Keller. 2020. Secure evaluation of quantized neural networks. *Proceedings on Privacy Enhancing Technologies* 2020, 4 (2020), 355–375.

[15] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. 2019. CHET: an optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 142–156.

[16] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY-A Framework for Efficient Mixed-Protocol Secure Two-Party Computation.. In *NDSS*. The Internet Society.

[17] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. 2020. Improved primitives for MPC over mixed arithmetic-binary circuits. In *Annual International Cryptology Conference*. Springer, 823–852.

[18] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. 2017. Dermatologist-level classification of skin cancer with deep neural networks. *Nature* 542, 7639 (2017), 115.

[19] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. 2019. A guide to deep learning in healthcare. *Nature medicine* 25, 1 (2019), 24.

[20] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *ACM CCS*.

[21] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 169–178.

[22] Zahra Ghodsi, Akshaj Veldanda, Brandon Reagen, and Siddharth Garg. 2020. Cryptonas: Private inference on a relu budget. In *Advances in Neural Information Processing Systems*.

[23] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *International Conference on Machine Learning*.

[24] Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.

[25] Matan Haroush, Itay Hubara, Elad Hoffer, and Daniel Soudry. 2020. The knowledge within: Methods for data-free model compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8494–8502.

[26] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. 2017. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189* (2017).

[27] Siam U Hussain, Baiyu Li, Farinaz Koushanfar, and Rosario Cammarota. 2020. TinyGarble2: Smart, Efficient, and Scalable Yao's Garble Circuit. In *ACM Workshop on Privacy-Preserving Machine Learning in Practice(PPMLP)*.

[28] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending Oblivious Transfers Efficiently.. In *Crypto*, Vol. 2729. Springer.

[29] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2704–2713.

[30] Mojan Javaheripi, Mohammad Samragh, Tara Javidi, and Farinaz Koushanfar. 2020. GeneCAI: gene tic evolution for acquiring c ompact AI. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 350–358.

[31] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. {GAZELLE}: A low latency framework for secure neural network inference. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 1651–1669.

[32] Veton Kepuska and Gamal Bohouta. 2018. Next-generation of virtual personal assistants (microsoft cortana, apple siri, amazon alexa and google home). In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 99–103.

[33] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. Cryptflow: Secure tensorflow inference. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 336–353.

[34] Baiyu Li and Daniele Micciancio. 2020. On the security of homomorphic encryption on approximate numbers. *IACR Cryptol. ePrint Arch* 2020 (2020), 1533.

[35] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. 2017. Oblivious neural network predictions via minionn transformations. In *SIGSAC Conference on Computer and Communications Security*. ACM, 619–631.

[36] Jian Liu, Mika Juuti, Yao Lu, and N Asokan. 2017. Oblivious Neural Network Predictions via MiniONN transformations. In *CCS*. ACM.

[37] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[38] Qian Lou, Yilin Shen, Hongxia Jin, and Lei Jiang. 2021. {SAFEN}et: A Secure, Accurate and Fast Neural Network Inference. In *International Conference on Learning Representations*.

[39] Qian Lou, Bian Song, and Lei Jiang. 2020. AutoPrivacy: Automated Layer-wise Parameter Selection for Secure Neural Network Inference. In *Advances in Neural Information Processing Systems*.

[40] Iacopo Masi, Yue Wu, Tal Hassner, and Prem Natarajan. 2018. Deep face recognition: A survey. In *2018 31st SIBGRAPI conference on graphics, patterns and images (SIBGRAPI)*. IEEE, 471–478.

[41] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. 2020. DELPHI: A cryptographic inference service for neural networks. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*.

[42] Daisuke Miyashita, Edward H Lee, and Boris Murmann. 2016. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025* (2016).

[43] Parwadi Moengin. 2011. Exponential Penalty Methods for Solving Linear Programming Problems. In *Proceedings of the World Congress on Engineering and Computer Science*, Vol. 2.

[44] Moni Naor and Benny Pinkas. 2005. Computationally secure oblivious transfer. *Journal of Cryptology* 18, 1 (2005).

[45] Alvin Rajkomar, Eyal Oren, Kai Chen, Andrew M Dai, Nissan Hajaj, Michaela Hardt, Peter J Liu, Xiaobing Liu, Jake Marcus, Mimi Sun, et al. 2018. Scalable and accurate deep learning with electronic health records. *npj Digital Medicine* 1, 1 (2018), 18.

[46] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. CrypTFlow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 325–342.

[47] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin E Lauter, and Farinaz Koushanfar. 2019. XONN: XNOR-based Oblivious Deep Neural Network Inference.. In *USENIX Security*.

[48] Peter Rindal. [n. d.]. libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe.

[49] Mohammad Samragh, Mohammad Ghasemzadeh, and Farinaz Koushanfar. 2017. Customizing neural networks for efficient fpga implementation. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 85–92.

[50] Amartya Sanyal, Matt Kusner, Adria Gascon, and Varun Kanade. 2018. TAPAS: Tricks to accelerate (encrypted) prediction as a service. In *International Conference on Machine Learning*. PMLR, 4490–4499.

[51] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *SIGSAC Conference on Computer and Communications Security*. ACM.
[52] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 3–18.
[53] Florian Tramer and Dan Boneh. 2019. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. In *International Conference on Learning Representations*.
[54] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 601–618.
[55] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. 2016. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit.
[56] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. 2018. Training and Inference with Integers in Deep Neural Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HJGXzmspb
[57] Lingxi Xie and Alan Yuille. 2017. Genetic cnn. *arXiv preprint arXiv:1703.01513* (2017).
[58] Ziqi Yang, Bin Shao, Bohan Xuan, Ee-Chien Chang, and Fan Zhang. 2020. Defending model inversion and membership inference attacks via prediction purification. *arXiv preprint arXiv:2005.03915* (2020).
[59] Andrew Yao. 1986. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*.

A OVERFLOW MANAGEMENT

Eq. 10 presents our overflow simulation, which models the loss of MSB bits in case of overflow for an `INT-b` accumulator.

$$\underset{x>0}{overflow}(x) = \begin{cases} x \bmod 2^b, & (x \bmod 2^b) < 2^{b-1} \\ x \bmod 2^b - 2^b, & otherwise \end{cases}$$
$$\underset{x<0}{overflow}(x) = \begin{cases} x \bmod 2^b, & (x \bmod 2^b) \geq -2^{b-1} \\ x \bmod 2^b + 2^b, & otherwise \end{cases} \tag{10}$$

Here, *mod* represents the modulo operation and $x \bmod 2^b$ checks for the occurrence of an overflow. In the forward pass (during DNN inference or training), the above operation is applied on all layer outputs to account for the occurrence of overflow according to the secure execution bitwidth. By leveraging the proposed overflow simulation, we accurately measure the secure execution accuracy in the presence of (occasional) overflows. This, in turn, allows us to find the best bitwidths throughout the network that strike a good balance between overflow, accuracy, and secure execution cost. Additionally, we provide afi ne-tuning methodology that further compensates the reduced inference accuracy caused by overflows as explained in the following.

Gradients for Overflow. To enablefi ne-tuning of models that include our overflow simulation in their computation graph (see Eq. 10), we provide a smooth approximation for the gradients of overflow. Let x be a scalar value, $\overline{x}$ denote its value after overflow, and $\nabla_{\overline{x}}$ be the gradient of the training loss function with respect to $\overline{x}$. We compute the gradient with respect to x as follows:

$$\nabla_x = \begin{cases} \nabla_{\overline{x}} & if \quad x = \overline{x} \\ 0 & otherwise \end{cases} \tag{11}$$

B CLUSTERING

Here we provide details on the clustering algorithm that reduces the unique size of weight matrices. Given a vectorized weight matrix $\mathbf{w} \in \mathbb{R}^N$ and a given unique size V, we aim tofi nd the unique space $\mathbf{c} \in \mathbb{R}^V$ and the coded representation $\widetilde{\mathbf{w}} \in \{1, \ldots, V\}^N$ that solve the following optimization:

$$\min_{\mathbf{c}, \widetilde{\mathbf{w}}} \sum_{i=1}^{N} (\mathbf{c}[\widetilde{\mathbf{w}}[i]] - \mathbf{w}[i])^2 \tag{12}$$

We use Lloyd's K-means algorithm [37] tofi nd the solution to the above optimization. It starts with a random set for $\mathbf{c}$ and sets $\widetilde{\mathbf{w}}[i]$ to the index of the value in the unique space that is closest to $\mathbf{w}[i]$:

$$\widetilde{\mathbf{w}}[i] = \underset{j \in \{1,\ldots,V\}}{arg\min} |\mathbf{w}[i] - \mathbf{c}[j]|, \quad \forall i \in \{1, \ldots, N\} \tag{13}$$

Next, the elements of the unique space are updated as follows:

$$\mathbf{c}[j] = average(\{\mathbf{w}[i] | \widetilde{\mathbf{w}}[i] = j\}), \quad \forall j \in \{1, \ldots, V\} \tag{14}$$

By repeating 13 and 14, the unique space and the coded representation are computed. For each DNN layer, we run the K-means algorithm on the weights with different V and pre-compute the unique space and coded values. During the automated design exploration (Section 4.3), we use these pre-computed values to cluster the weights of DNN layers.

Table 6: COINN secure execution cost for core operations in a DNN. Here, κ is the security parameter that is set to 128.

	$\text{input}_{\text{dim}} \xrightarrow{\text{operation}} \text{output}_{\text{dim}}$	Ciphertext Cost		Parameters
Mat-Mult	$X_{N\times L} \xrightarrow{W_{M\times N}} Y_{M\times L}$	regular	$MNLb_{acc}^2$	b_{acc}: accumulator bitwidth
		factored	$MVLb_{acc}(N+b_{acc}+1)$	V: unique size of W
MaxPool	$X_{C\times D_1\times D_1} \xrightarrow{k\times k} Y_{C\times D_2\times D_2}$	$4\kappa CD_2D_2(k^2-1)b_{inp}$		b_{inp}: next layer input bitwidth k: window size
ReLU	$X_{C\times D_1\times D_1} \xrightarrow{>0} Y_{C\times D_1\times D_1}$	$2\kappa CD_1D_1b_{inp}$		b_{inp}: next layer input bitwidth
AS → GC	$X_{C\times D_1\times D_1} \xrightarrow{b_{acc}\to b_{inp}} Y_{C\times D_1\times D_1}$	$5\kappa CD_1D_1b_{acc}$		b_{acc}: accumulator bitwidth
GC → AS	$X_{C\times D_1\times D_1} \xrightarrow{b_{inp}\to b_{acc}} Y_{C\times D_1\times D_1}$	$3\kappa CD_1D_1b_{inp}$		b_{inp}: next layer input bitwidth b_{acc}: accumulator bitwidth

Fine-tuning Clustered Parameters. Let $\mathbf{w}$ be a weight vector, $\mathbf{c}$ denote the set of cluster centers, and $\overline{\mathbf{w}}$ be its approximated version after clustering. During forward propagation, $\overline{\mathbf{w}}$ is computed based on $\mathbf{w}$ and $\mathbf{c}$, then it is used to perform CONV and FC. During backward propagation, $\mathbf{w}$ and $\mathbf{c}$ are updated. Since clustering is a non-differentiable operation, we need to approximate its gradient computation to enablefi ne-tuning. Let $\nabla_{\overline{\mathbf{w}}}$ be the gradient of the training loss function with respect to the approximated value. We compute the gradient with respect to $\mathbf{w}$ as $\nabla_{\mathbf{w}} = \mathbf{m} \bigodot \nabla_{\overline{\mathbf{w}}}$ where $\bigodot$ denotes element-wise multiplication and $\mathbf{m}$ is defined as follows:

$$\mathbf{m}[i] = \begin{cases} 1 & if \quad min(\mathbf{c}) < \mathbf{w}[i] < max(\mathbf{c}) \\ 0 & otherwise \end{cases} \tag{15}$$

Let $I_j = \{i | \widetilde{\mathbf{w}}[i] = j\}$ be the set of indices where $\mathbf{w}$ is approximated via $\mathbf{c}[j]$. The gradient with respect to $\mathbf{c}[j]$, i.e., the j-th element of the unique space, is computed as:

$$\nabla_{\mathbf{c}}[j] = \sum_{i \in I_j} \nabla_{\overline{\mathbf{w}}}[i] \cdot \frac{1}{|I_j|} \tag{16}$$

C LAYER FUSION

Batch Normalization. Recall that BN operates on the output of its preceding CONV layer, i.e., $Y \in \mathbb{R}^{M\times L}$, It multiplies each row by α_i and adds β_i to the result. Naïve implementation of the BN would treat this layer independently which incurs a non-negligible secure execution cost. Instead, we fuse the BN operation into the preceding CONV layer so that the combination of CONV +BN can be realized via a single matrix-multiplication. The i-th row of Y is originally computed in the preceding CONV layer as $\mathbf{y}_i = \mathbf{w}_i \cdot X + \mathbf{b}_i$. Application of BN on this row vector renders:

$$\begin{aligned} BN(Y_i) &= \alpha_i \mathbf{y}_i + \beta_i &&= \alpha_i(\mathbf{w}_i \cdot X + \mathbf{b}_i) + \beta_i \\ &&&= \alpha_i \mathbf{w}_i \cdot X + \alpha_i \mathbf{b}_i + \beta_i \end{aligned} \tag{17}$$

We thus remove the BN layer and set the preceding CONV's weight matrix rows to $\alpha_i \mathbf{w}_i$ and bias values to $\alpha_i \mathbf{b}_i + \beta_i$.

Average Pooling. Average pooling works by computing the sum over $k \times k$ windows of convolution outputs and dividing the summation result by k^2. Prior work [46] implements an additional protocol to securely evaluate the division at an extra cost. In contrast, we propose to fuse the AP with a linear layer in the plaintext model to remove its overhead. Similar to the fusion of BN, we can avoid division by k^2 by dividing the weight and bias of the preceding CONV/FC layer by k^2, and computing the sum instead of average values in the pooling layer. In our setting, we perform average pool layers with zero cost in AS as summation is free in this protocol.

D CIPHERTEXT COMMUNICATION COST

Table 6 summarizes the communication cost associated with different ciphertext operations described in Section 5. These cost are incorporated into our automated design customization tool presented in Section 4.3. Note that for the linear layers, we report the amortized costs (see Section 5.2).

Table 7: Evaluation on convolution layers of COINN with regular matrix multiplication

Input	Kernal	LAN			WAN		
$C \times H \times W$	$N \times F \times F$	$b = 8$	$b = 16$	$b = 32$	$b = 8$	$b = 16$	$b = 32$
$16 \times 32 \times 32$	$16 \times 3 \times 3$	0.021	0.073	0.317	0.703	1.217	3.557
$32 \times 16 \times 16$	$32 \times 3 \times 3$	0.021	0.071	0.284	0.711	1.202	3.350
$64 \times 8 \times 8$	$64 \times 3 \times 3$	0.027	0.070	0.268	0.703	1.220	3.139

Table 8: Evaluation on convolution layers of COINN with factored matrix multiplication, $b = 16$

Input	Kernal	LAN			WAN		
$C \times H \times W$	$N \times F \times F$	$V = 8$	$V = 12$	$V = 16$	$V = 8$	$V = 12$	$V = 16$
$16 \times 32 \times 32$	$16 \times 3 \times 3$	0.039	0.056	0.073	0.902	1.107	1.217
$32 \times 16 \times 16$	$32 \times 3 \times 3$	0.037	0.054	0.073	0.810	1.011	1.203
$64 \times 8 \times 8$	$64 \times 3 \times 3$	0.038	0.053	0.071	0.812	1.013	1.227

Table 9: Evaluation on ReLU of COINN (including AS-GC conversions)

Input	LAN			WAN		
$C \times H \times W$	$b = 8$	$b = 16$	$b = 32$	$b = 8$	$b = 16$	$b = 32$
$16 \times 32 \times 32$	0.063	0.084	0.116	1.581	1.645	1.664
$32 \times 16 \times 16$	0.053	0.058	0.062	1.019	1.130	1.232
$64 \times 8 \times 8$	0.018	0.025	0.033	0.413	0.421	0.432

E EVALUATION ON MICROBENCHMARKS

We present the evaluation results on standalone linear and nonlinear layers of COINN in tables 7, 8, and 9. Observe that the run-time for the convolution layers with regular matrix-multiplication increases quadratically with the bitwidth b. The quantization and automated parameter configurator of COINN thus greatly enhance the performance by minimizing the bitwidths. The run-time of convolution layers with factored multiplication increases linearly with the number of unique elements, Vm and becomes equal to that of regular matrix-multiplication when $V = b$. This is consistent with the analysis in Section 5.2. Table 9 shows the run-time of combined AS to GC, ReLU, and GC-AS operations. As expected, the run-time increases linearly with b.