BAYHENN: Combining Bayesian Deep Learning and Homomorphic Encryption for Secure DNN Inference

Peichen Xie[1,2*], **Bingzhe Wu**[1,3*] and **Guangyu Sun**[1,2]
[1]Center for Energy-efficient Computing and Applications, Peking University
[2]Advanced Institute of Information Technology, Peking University
[3]Ant Financial Services Group
{xpc, wubingzhe, gsun}@pku.edu.cn

Abstract

Recently, deep learning as a service (DLaaS) has emerged as a promising way to facilitate the employment of deep neural networks (DNNs) for various purposes. However, using DLaaS also causes potential privacy leakage from both clients and cloud servers. This privacy issue has fueled the research interests on the privacy-preserving inference of DNN models in the cloud service. In this paper, we present a practical solution named BAYHENN for secure DNN inference. It can protect both the client's privacy and server's privacy at the same time. The key strategy of our solution is to combine homomorphic encryption and Bayesian neural networks. Specifically, we use homomorphic encryption to protect a client's raw data and use Bayesian neural networks to protect the DNN weights in a cloud server. To verify the effectiveness of our solution, we conduct experiments on MNIST and a real-life clinical dataset. Our solution achieves consistent latency decreases on both tasks. In particular, our method can outperform the best existing method (GAZELLE) by about 5×, in terms of end-to-end latency.

1 Introduction

In the past years, deep neural networks (DNNs) have achieved remarkable progress in various fields, such as computer vision [He *et al.*, 2016; Ren *et al.*, 2017], natural language processing [Vaswani *et al.*, 2017; Devlin *et al.*, 2018], and medical image analysis [Litjens *et al.*, 2017; Havaei *et al.*, 2017]. Recently, deep learning as a service (DLaaS) has emerged as a promising to further enable the widespread use of DNNs in industry/daily-life. Google[1], Amazon[2], and IBM [Bhattacharjee *et al.*, 2017] have all launched DLaaS platforms in their cloud services. Using DLaaS, a client sends its private data to the cloud server. Then, the server is responsible for performing the DNN inference and sends the prediction results back to the client. Obviously, if the private client data are not protected, using DLaaS will cause potential privacy issues. A curious server may collect sensitive information contained in the private data (*i.e.* **client's privacy**).

To address this privacy issue, researchers have employed the homomorphic encryption to perform various DNN operators on encrypted client data [Phong *et al.*, 2018; Ma *et al.*, 2019]. As a result, the cloud server only serves as a computation platform but cannot access the raw data from clients. However, there exist two major obstacles in applying these approaches. First, some common non-linear activation functions, such as ReLU and Sigmoid, are not cryptographically computable. Second, the inference processing efficiency is seriously degraded by thousands of times.

To tackle these problems, a recent work [Zhang *et al.*, 2018] proposes using an interactive paradigm. A DNN inference is partitioned into linear and non-linear computation parts. Then, only the linear computations are performed on the cloud server with encrypted data. The non-linear computations are performed by the client with raw data. However, in such an interactive paradigm, the intermediate features extracted by the linear computations are directly exposed (sent back) to the client. Thus, a curious client can leverage these features to reconstruct the weights of the DNN model held by the cloud [Tramèr *et al.*, 2016; Zhang *et al.*, 2018]. This issue is called the leakage of **server's privacy**.

In fact, a practical solution for secure DNN inference should protect both client's privacy and server's privacy. In addition, it should support DNNs with all types of non-linear activation functions. Unfortunately, there still lacks an effective approach in literature. The limitations of recent works are discussed in Section 2.3. Thus, we propose a new method for secure DNN inference, called BAYHENN. Our key strategy is to combine Bayesian deep learning and homomorphic encryption. To the best of our knowledge, it is the first approach that can protect both client's privacy and server's privacy and support all types of non-linear activation functions at the same time.

BAYHENN follows an interactive paradigm so that all types of activation functions are supported. On the one hand, it uses the homomorphic encryption to provide protection for the client's privacy. On the other hand, it adds proper randomness into DNN weights to prevent the leakage of the server's privacy. This idea is motivated by the learning with

*Equal contribution.
[1]https://cloud.google.com/inference/
[2]https://docs.aws.amazon.com/machine-learning/index.html

error (LWE) problem, in which solving the noisy linear equations system has been proven to be NP-hard [Regev, 2009]. However, *"adding proper randomness"* is a new challenge. Directly injecting well-designed Gaussian Noise into weights will cause a drastic performance degradation [Chaudhuri and Monteleoni, 2008; Zhang *et al.*, 2017]. Therefore, we propose to use a more intrinsic method, the Bayesian neural network, to model the weight uncertainty of DNNs. Armed with the weight uncertainty, we can protect the weight information by sending obscured features to the client.

In summary, the contributions of our paper are as follows:

- We provide a novel insight that using the Bayesian neural network can prevent leakage of a server's privacy under an interactive paradigm.
- Based on this insight, we build a practical solution for secure DNN inference. Our solution is more (about $5\times$) efficient than the best existing method, and is capable of supporting all types of activation functions.
- We validate the proposed solution on a real-life clinical dataset, which is less explored in previous studies.

2 Preliminary

In this part, we first introduce some basic notations in DNN inference. Then, the potential privacy leakages in DLaaS and the goal of this paper are presented. At last, we summarize the limitations of previous works on secure DNN inference.

2.1 DNN Inference

We start from the process of a DNN inference. For simplicity, the following discussion is based on a fully connected neural network. It can be easily switched to the context of a convolutional neural network, as shown in our experiments.

We take the image classification as an example in this part. It can be extended to other types of tasks as well (*e.g.* regression). The image classification aims to assign a label t to the input image $\mathbf{x}$. To this end, a classification model $\mathbb{M} : t = f(\mathbf{x}; \boldsymbol{\theta})$ is trained on some pre-collected data. In a context of deep learning, the f is a highly non-linear function, which is formed by alternately stacking several linear (*e.g.* fully connected layer) and non-linear (*e.g.* activation function) operations[3].

Formally, for a neural network, we denote $\mathbf{W}^{(i)}$ and $\mathbf{b}^{(i)}$ as weights and biases corresponding to the i-th linear layer. To calculate outputs (*i.e.* activations) of the i-th non-linear layer (denoted as $\mathbf{a}^{(i)}$), a linear transformation is firstly performed on the inputs from previous layers following:

$$\mathbf{z}^{(i)} = \mathbf{W}^{(i)}\mathbf{a}^{(i-1)} + \mathbf{b}^{(i)} \tag{1}$$

Then, the nonlinear activation function is used to obtain output activation results:

$$\mathbf{a}^{(i)} = \varphi^{(i)}(\mathbf{z}^{(i)}) \tag{2}$$

[3]Most previous DNNs satisfy this topology. Note that one can easily integrate two adjacent linear/nonlinear layers into one "logic" layer. For example, we can combine one batch normalization and one convolution layer together.

	Client's Privacy	Server's Privacy	Support all activation func.
CryptoNets	Yes	Yes	—
GELU-Net	Yes	—	Yes
GAZELLE	Yes	Yes	—
Ours	Yes	Yes	Yes

Table 1: Feature comparison among different solutions.

where $\varphi^{(i)}$ denotes the activation function (*e.g.* ReLU).

In summary, the classification model $\mathbb{M}$ can be reformulated into:

$$\mathbb{M} : t = f(\mathbf{x}; \{(\mathbf{W}^{(i)}, \mathbf{b}^{(i)})\}_{i=1}^{n}) \tag{3}$$

2.2 Privacy Leakage in DLaaS

In a common setting, there are two parties involved in a process of DLaaS. Specifically, at the beginning, a *client* sends its private data $\mathbf{x}$ into a cloud *server*, which holds the DNN model $\mathbb{M}$ for inference. Then, the server can perform the DNN inference and send the prediction results back to the client. Considering that both the client and the server are semi-honest, privacy leakages may occur on both sides:

- **Client's Privacy Leakage:** From the perspective of a client, its input data $\mathbf{x}$ is directly exposed to the cloud server in a non-private setting. Thus, the server may collect the sensitive information contained in $\mathbf{x}$.
- **Server's Privacy Leakage:** A cloud server also holds the private data, *i.e.* the weight parameters and the DNN structure of $\mathbb{M}$. A curious client may attempt to obtain these valuable model information.

In this paper, we focus on a secure DNN inference. Our goal is to prevent the privacy leakage of the input data (client-side) and the model parameters (server-side). The protection of a model structure is out of the scope of this paper, yet part of the DNN model structure (such as the filter and stride size in the convolution layers and the type of each layer) is also protected in our solution[4].

2.3 Limitation of Previous Work

In this subsection, we discuss the limitations of previous works, which are most related to ours (listed in Table 1). We do not consider [Bourse *et al.*, 2018; Sanyal *et al.*, 2018], since they only work on binary neural networks and do not outperform the state-of-the-art.

CryptoNets [Gilad-Bachrach *et al.*, 2016] is the first system for homomorphic encryption based neural network inference. However, its end-to-end latency for a single input is extremely high, especially on a DNN. More importantly, CryptoNets cannot support most of the common activation functions, such as Sigmoid and ReLU, or the common pooling functions, such as Max-Pooling. These two issues limit the use of CryptoNets in real-life scenarios.

[4]Juverka *et al.* [2018] have explained how homomorphic encryption can hide this information.

GELU-Net [Zhang *et al.*, 2018] proposes using an interactive paradigm to address the issues of CryptoNets. However, the interactive paradigm may result in server's private weight leakage. Thus, GELU-Net needs to limit the number of queries requested by one specific client.

GAZELLE [Juvekar *et al.*, 2018] is state-of-the-art work combining garbled circuits and homomorphic encryption. Garbled circuits can be used to protect the client's weights [Liu *et al.*, 2017; Mohassel and Zhang, 2017; Rouhani *et al.*, 2018]. However, such a method introduces non-negligible computation/communication costs, and does not support those activation functions which are garbled-circuit unfriendly (*e.g.* Sigmoid).

Compared with these works, our proposed solution can overcome all the above limitations. Moreover, unlike these works, we validate our solution on a real-life clinical dataset, which further shows the practicability of our solution in real-life applications.

3 Our Approach

In this section, we first describe the high-level protocol of BAYHENN and then detail its implementation for secure linear/non-linear computations.

3.1 Protocol at a High Level

Here, we elaborate our protocol for secure DNN inference at a high level. The core idea for designing our protocol is based on the combination of homomorphic encryption and deep Bayesian inference.

At a high level, our protocol follows an interactive paradigm and comprises of two sub-protocols, namely, the SLC (**S**ecure **L**inear **C**omputation) and the SNC (**S**ecure **N**on-linear **C**omputation) protocols. Consider a basic setting where the centric server holds the model $\mathbb{M}$ and the client holds the private data $\mathbf{x}$. The inference task $t = f(\mathbf{x}; \boldsymbol{\theta})$ is decomposed into linear and non-linear computations based on Equation (1) and (2). Each of them is performed by different parties (the server and the client) using the SLC and SNC protocols, respectively. To be specific, the client firstly encrypts their private data $\mathbf{x}$ and sends the encrypted data to the server as model input. Then, the server can perform the linear computations on the encrypted input following the SLC protocol. Due to most commonly used non-linear activation functions are not cryptographically computable, the output is sent back to the client for the further non-linear computation following the SNC protocol. Then, the client can re-encrypt the computed results and send them to the server for computation of the next layer. Once all computation tasks are finished, the client can receive the final prediction t. The whole procedure of our protocol is depicted in Algorithm 1. Two core sub-protocols, SLC and SNC, are detailed in the following part.

Algorithm 1 Secure DNN inference

Input: $(\mathbf{x};\ S, P(\boldsymbol{\theta}|\mathcal{D}), \{\varphi^{(i)}\}_{i=1}^{n})$
Output: $(t;\ \varnothing)$

Initiation
Server:
for $k \leftarrow 1$ **to** S **do**
  samples $\boldsymbol{\theta}_k = \{\mathbf{W}_k^{(i)}, \mathbf{b}_k^{(i)}\}_{i=1}^{n}$ from $P(\theta|\mathcal{D})$
end for
Server: $\Theta \leftarrow \{\boldsymbol{\theta}_k\}_{k=1}^{S}$
Client: $\mathbf{a}^{(0)} \leftarrow \mathbf{x}$
for $i \leftarrow 1$ **to** n **do**
  $\tilde{\mathbf{z}}^{(i)} \leftarrow \mathrm{SLC}(\mathbf{a}^{(i-1)};\ \Theta)$
  $\mathbf{a}^{(i)} \leftarrow \mathrm{SNC}(\tilde{\mathbf{z}}^{(i)};\ \varphi^{(i)})$
end for
Client: $\mathbf{p} \leftarrow \frac{1}{S}\sum_{i=1}^{S} \mathbf{a}_k^{(n)}$
Client: $t \leftarrow \mathrm{argmax}_k\, p_k$

3.2 SLC Protocol

The aim of this protocol is to enable the linear computations (*e.g.* convolution) while providing the protection for both the client's data and the server's parameters. In the following parts, we first introduce some basic functions in a vectorizable homomorphic encryption scheme. Then, we demonstrate how SLC protocol can successfully achieve the server-level (server's parameters) and client-level (client's data) privacy protection.

Vectorizable Homomorphic Encryption

To ensure the correctness of the linear operations in ciphertext, we adopt homomorphic encryption, which provides three basic functions: the encryption function $\mathbb{E}$, the decryption function $\mathbb{D}$ and the evaluation function Eval. To be specific, $\mathbb{E}$ is responsible for encrypting the plaintext x to the ciphertext $\tilde{x}$ under a public key. In contrast, $\mathbb{D}$ is to decrypt the ciphertext into the plaintext under a secret key. The homomorphic operations are instantiated by the evaluation function. For two elements $x_1, x_2 \in \mathcal{R}$ ($\mathcal{R}$ denotes the plaintext space as a ring), we can establish the following equations:

$$\mathbb{D}(\mathrm{Eval}(\mathbb{E}(x_1), \mathbb{E}(x_2), +)) = x_1 + x_2 \tag{4}$$

$$\mathbb{D}(\mathrm{Eval}(\mathbb{E}(x_1), x_2, \times)) = x_1 \times x_2 \tag{5}$$

where $+$ and $\times$ denote the normal addition and multiplication in the plaintext space $\mathcal{R}$, respectively. For simplicity, we will use the notation $\tilde{x}_1 \oplus \tilde{x}_2$ and $\tilde{x}_1 \otimes x_2$ in the following part to represent the instantiation of the ciphertext-ciphertext addition $\mathrm{Eval}(\mathbb{E}(x_1), \mathbb{E}(x_2), +)$ and the ciphertext-plaintext multiplication $\mathrm{Eval}(\mathbb{E}(x_1), x_2, \times)$ respectively. Note that the ciphertext-ciphertext multiplication is not needed in our approach.

It is impractical and inefficient to directly apply a conventional homomorphic encryption scheme to the DNN inference. Firstly, the above functions are defined over the ring $\mathcal{R}$ whereas the parameters in a DNN model are typically floating points. Then, operations (*e.g.* convolution) in a DNN model are always performed on the high-dimensional tensors, which leads to the inefficiency of the straightforward solution (*i.e.* to compute the tensor multiplication/addition element by element). To tackle these two issues, we adopt a *vectorizable* homomorphic encryption scheme[5], which involves the Encode function. This function is capable of

[5]The vectorization is also know as "batching" or "SIMD" in the literature.

packing a group of floating points to a single element in $\mathcal{R}$ (*i.e.* one plaintext). On the contrary, the Decode function is to transform the plaintext into a number of floating points. Specifically, we implement the encoding and decoding functions following prior works [Gilad-Bachrach *et al.*, 2016; Juvekar *et al.*, 2018].

Client-side Privacy Protection

To protect the privacy of the client's data, we make use of homomorphic encryption. Homomorphic encryption enables the client to encrypt the private data and the server to conduct the linear computation on the encrypted data. Here, we use notations in Section 2 to explain how a vectorizable homomorphic encryption scheme works in DNN inference. To compute the output of the i-th linear layer ($\mathbf{z}^{(i)}$), the client first encrypt the i-th input vector $\mathbf{a}^{(i-1)}$ into $\tilde{\mathbf{a}}^{(i-1)}$ via the encoding and encryption functions. For the linear computation, we can always decompose it into a number of multiply-accumulate operations of vectors. Thus we can use the Equation 4 and 5 and the above vectorization technique to achieve the computation of the linear layer. For simplicity, we formulate the process of the linear computation as[6]:

$$\tilde{\mathbf{z}}^{(i)} = \mathrm{Encode}(\mathbf{W}^{(i)}) \otimes \tilde{\mathbf{a}}^{(i-1)} \oplus \mathbb{E}(\mathrm{Encode}(\mathbf{b}^{(i)})) \quad (6)$$

With a little abuse of notations, we use $\otimes$ to represent the multiplication between an unencrypted matrix and an ecrypted vector. Armed with the above equations, the server can achieve linear computation without knowing any information about the client's data.

Server-side Privacy Protection

As discussed in Section 2.2, the interactive paradigm comes with the risk of the weight leakage. To tackle this issue, motivated by the learning with error (LWE) problem [Regev, 2009], we propose using Bayesian neural networks to build the model parameters with moderate uncertainty.

The BNN is to model the weight as the probability distributions instead of the fixed floating points in the traditional DNN. Here, we borrow some notations from the DNN inference. The target of the BNN is to predict the label distribution $P(t|\mathbf{x})$. Given the training dataset $\mathcal{D}$, we can estimate the posterior distribution of the weights $P(\boldsymbol{\theta}|\mathcal{D})$ via Bayes by Backprop [Blundell *et al.*, 2015]. Then, the posterior distribution can be used for label prediction. Specifically, given a test data $\mathbf{x}$ (client's data in our case), the label distribution can be obtained following:

$$P(t|\mathbf{x}) = \mathbb{E}_{\boldsymbol{\theta}\sim P(\boldsymbol{\theta}|\boldsymbol{\mathcal{D}})}(P(t|\mathbf{x},\boldsymbol{\theta})) \quad (7)$$

To compute the expectation defined in Equation 7, we can sample a number of models based on the posterior distribution. These models share the same model architecture while having different parameters sampled from $P(\boldsymbol{\theta}|\mathcal{D})$. Then, we can perform the inferences of these models and average the outputs for the final prediction. In summary, the whole process can be represented as:

$$P(t|\mathbf{x}) \approx \frac{1}{S}\sum_{k=1}^{S} P(t|\mathbf{x},\boldsymbol{\theta}_k) \quad (8)$$

[6]Here, we take FC layer as an example, in practice, it is natural to extend such a process to the convolution layer.

Algorithm 2 SLC

Input: $(\mathbf{a}^{(i)};\ \mathbf{W}^{(i)}, \mathbf{b}^{(i)})$
Output: $(\mathbf{z}^{(i)};\ \varnothing)$

Client: $\tilde{\mathbf{a}}^{(i)} \leftarrow \mathbb{E}(\mathrm{Encode}(\mathbf{a}^{(i)}))$
Client: sends $\tilde{\mathbf{a}}^{(i)}$ to the server
Server: starts S threads
for $k \leftarrow 1$ **to** S **do**
  $\tilde{\mathbf{z}}_k^{(i)} \leftarrow \mathrm{Encode}(\mathbf{W}_k^{(i)}) \otimes \tilde{\mathbf{a}}_k^{(i-1)} \oplus \mathbb{E}(\mathrm{Encode}(\mathbf{b}_k^{(i)}))$
end for
Server: $\tilde{\mathbf{z}}^{(i)} \leftarrow \{\mathbf{z}_k^{(i)}\}_{k=1}^{S}$
Server: sends $\tilde{\mathbf{z}}^{(i)}$ to the client

Algorithm 3 SNC

Input: $(\tilde{\mathbf{z}}^{(i)};\ \varphi^{(i)})$
Output: $(\mathbf{a}^{(i)};\ \varnothing)$

Server: sends $\varphi^{(i)}$ to server
Client:
for $k \leftarrow 1$ **to** S **do**
  $\mathbf{z}_k^{(i)} \leftarrow \mathrm{Decode}(\mathbb{D}(\tilde{\mathbf{z}}_k^{(i)}))$
  $\mathbf{a}_k^{(i)} \leftarrow \varphi^{(i)}(\mathbf{z}_k^{(i)})$
end for
Client: $\mathbf{a}^{(i)} \leftarrow \{\mathbf{a}_k^{(i)}\}_{k=1}^{S}$

where S denotes the number of sampling. $\boldsymbol{\theta}_k$ is the k-th sampling result and inherits the formulation from Equation 3 in which $\boldsymbol{\theta}_k = \{\mathbf{W}_k^{(i)}, \mathbf{b}_k^{(i)}\}_{i=1}^{n}$. The computation of Equation 8 can be treated as the computation of $\{P(t|\mathbf{x},\boldsymbol{\theta}_k)\}_{k=1}^{S}$, which is naturally parallelizable. Since the modern server easily allows process-level parallelism, the computation can be seen as S parallel processes of DNN inference and do not significantly increase the computation time.

During the inference of a BNN, due to the features exposed to the client are inaccurate compared with the normal DNN inference, it is hard for the client to solve the exact weights of the server. The hardness comes from the challenge of the LWE problem, where solving the noisy linear system is as difficult as solving lattice problems[7]. Note that the weights of the BNN are the parameters of massive probability distributions. In our case, each $\mathbf{W}^{(i)}$ follows the multivariate Gaussian distribution, and the weights of the layer are the mean and variance vectors of the Gaussian distribution.

Put all things together, we can build the SLC protocol, which can prevent the leakage of weights and input data simultaneously. The SLC protocol is depicted in Algorithm 2.

3.3 SNC Protocol

This protocol is to enable secure non-linear computations of DNNs. In our protocol, the non-linear computations are performed on the unencrypted data and are executed on the client individually. Specifically, to compute the output of the i-th non-linear layer ($\mathbf{a}^{(i)}$), the client first decrypts the input $\tilde{\mathbf{z}}^{(i)}$ into $\mathbf{z}^{(i)}$ via the decryption function and the decoding func-

[7]The lattice problems are used as a base to build homomorphic encryption schemes [Acar *et al.*, 2018].

tion. In the context of Bayesian inference, we will get S vectors of floating points. Then, following Equation 2, the client can apply the activation function $\varphi^{(i)}$ to each of these vectors $\mathbf{z}_k^{(i)}$ simultaneously. In summary, the SNC protocol is described in Algorithm 3. In the execution process of the SNC protocol, the client does not need to send any private information to the server, and the server does not expose the weight parameters explicitly/implicitly to the client.

4 Experiment

4.1 Implementation Details

Considering efficiency and comparability, we instantiate our solution BAYHENN using the SEAL library[8], the homomorphic encryption library widely used by previous works. For a fair comparison, we adopt SEAL's BFV scheme, which is also used by the state-of-the-art solution GAZELLE. In this scheme, we set the security level to 128-bit, the degree of the polynomial modulus to 2048, the plaintext modulus to 20-bit, and the noise standard deviation to 4.

We deploy BAYHENN and previous works on a centric server with an Intel Xeon E5-2650 v3 2.30 GHz CPU and a client PC with an Intel Core i3-7100 3.90 GHz CPU. The bandwidth between the server and the client is 100 Mbps. To evaluate efficiency, we use the end-to-end latency (including protocol setup and communication) as a metric to measure the performance of different solutions.

4.2 Performance Evaluation

Model Setup

For each DNN architecture used in our experiments, we need to train three individual versions to meet the requirements of different secure DNN inference methods. We list these three versions as follows:

- `Normal`: Used by GAZELLE and GELU-Net.
- `Square`: Replacing all the activation functions with the square function and replacing all the pooling functions with the sum pooling. Used by CryptoNets.
- `Bayes`: Bayesian version of the base DNN, which is trained by Bayes By Backprop. Used by BAYHENN.

Digit Classification

The first task we consider is the hand-written digit classification. We build our model based on the MNIST dataset, which consists of 60,000 training and 10,000 validation images of 28 by 28 pixels. We adopt the LeNet5 [LeCun *et al.*, 1998] as the basic network architecture with the input size of 28×28. We select the optimizer following the prior work [Shridhar *et al.*, 2018]. Specifically, we make use of Adam for `Bayes` and `Square`. The learning rate is set to 0.001. For `Normal`, we select the Momentum SGD as the optimizer. The learning rate and momentum are set to 0.01 and 0.9, respectively. For a fair comparison, all these versions are trained without data augmentation or Dropout. The sampling number of `Bayes` is set to 4.

[8]https://github.com/microsoft/SEAL

Framework	Version	Scheme	Accuracy (%)	Latency (s)
CryptoNets	`Square`	YASHE'	96.09	3593.22
GELU-Net	`Normal`	Paillier	99.05	107.49
GAZELLE	`Normal`	BFV	99.05	6.29
Ours	`Bayes`	BFV	98.93	1.36

Table 2: Performance comparison on digit classification.

Framework	Version	Scheme	Accuracy (%)	Latency (s)
CryptoNets	`Square`	YASHE'	81.66	—
GELU-Net	`Normal`	Paillier	83.58	4755.59
GAZELLE	`Normal`	BFV	83.58	21.64
Ours	`Bayes`	BFV	83.36	4.17

Table 3: Performance comparison on breast cancer classification.

The overall results are shown in Table 2. From the results, our solution achieves the best latency in contrast to the other three solutions. For the classification performance, there is a slight accuracy decrease from `Bayes` (used by our solution) to `normal`. We infer this decrease may be caused by that the first-order optimizer can not efficiently optimize the Bayesian neural network. On the other hand, both `Bayes` and `Normal` can outperform `Square` by a large margin. This can be caused by the gradient vanish issue in DNN training, which is brought by the square activation function. In terms of efficiency, our solution outperforms the state-of-the-art work (GAZELLE) by 4.93 s in latency (a speedup of 4.63×) with an accuracy drop of 0.12%.

Classification on Breast Cancer

To further demonstrate the effectiveness of our approach, we apply it to a publicly available dataset for invasive ductal carcinoma (IDC) classification[9], which contains 277,524 patches of 50×50 pixels (198,738 IDC-negative and 78,786 IDC-positive). We chose a modified version of AlexNet [Shridhar *et al.*, 2018] as the base network architecture. For preprocessing, the input image is resized to 32×32. We adopt the similar training strategy of digit classification, *i.e.* training without data augmentation or Dropout.

The overall results[10] are shown in Table 3. From the results, BAYHENN has achieved consistent improvements on the task of IDC classification. In terms of end-to-end latency, BAYHENN significantly outperforms GAZELLE as well as other two previous works. In particular, BAYHENN can outperforms GAZELLE by 17.47 s, which shows a 5.19× speedup. In contrast to GELU-Net, our method can speed up for 1140×. This drastic improvement can be attributed to the use of a more advanced homomorphic encryption scheme.

All the above results indicate that our solution can significantly speed up the secure inference of the DNN compared with previous works. Moreover, as discussed previously, BAYHENN can provide the protection of the server's weights without limiting the number of requests by one client.

[9]http://www.andrewjanowczyk.com/use-case-6- invasive-ductal-carcinoma-idc-segmentation/

[10]The computation of AlexNet using CryptoNets cannot be completed within an acceptable time.

Version	MNIST			IDC		
	Train	Test	Gap	Train	Test	Gap
`Normal`	99.79	99.05	0.74	86.63	83.58	3.05
`Square`	97.87	96.09	1.78	82.67	81.66	1.01
`Bayes`	98.97	98.93	0.04	84.17	83.36	0.81

Table 4: The gap between training and testing accuracy (%).

Framework	LeNet5 on MNIST	AlexNet on IDC
CryptoNets	595.50	—
GELU-Net	0.83	2.48
GAZELLE	77.95	252.60
Ours	0.81	6.32

Table 5: The communication costs (MB).

4.3 Discussion

In this section, we provide some discussions from the following aspects.

Regularization of BNN

In addition to validating on the testing dataset, we also perform the evaluation on the training dataset to explore the regularization effect of the Bayesian neural network. The results are shown in Table 4. Here we use the gap between training and testing accuracy to quantitatively measure the degree of overfitting. From the results, we observe that the `Bayes` version has achieved the smallest gap. For the task of digit classification, `Bayes` decreases the gap from 0.74 to 0.04 compared with `normal`, while there is a gap decrease of 2.24 in the task of IDC classification. We infer the regularization effect comes from the penalty term (the KL distance between the prior and variational distribution) in the objective function (more details can be found in the prior work [Blundell *et al.*, 2015]). These results indicate that the Bayesian neural network leads to a stronger regularization effect than the traditional methods. Moreover, this fact shows our solution can be used in the scenario where the training data is scarce.

Communication Cost

As shown in Table 5, the communication cost can be a non-negligible component of the end-to-end latency. Among all the tested frameworks, GAZELLE is most affected by the communication cost (more than 90% of the latency comes from data transmission). We point out that this is mainly due to the garbled circuits of ReLU and MaxPool, whose size is proportional to the size of intermediate features of the DNN. On the contrary, without the use of garbled circuits, BAYHENN makes a remarkable decrease in data transmission, which leads to 4.63× and 5.19× speedup for LeNet5 and AlexNet respectively. In addition, the communication cost of GELU-Net is even lower. Indeed, we do need to transfer multiple intermediate features in ciphertext due to the multiple sampling, but this is a reasonable trade-off between communication costs and privacy protection.

Other Homomorphic Encryption Schemes

For a fair comparison, we make use of the BFV scheme in our solution. In fact, we have also conducted a series of experiments to test the performance of other homomorphic encryption schemes including YASHE' and CKKS (vectorizable), and Paillier (not vectorizable). Among these schemes and BFV, there is no single scheme that can outperform others for any DNN. Specifically, our solution with the CKKS scheme has a latency of 1.61 s and 10.07 s (with 1.38 MB and 7.75 MB communication costs) for LeNet5 and AlexNet respectively (in addition, YASHE' and Paillier are much slower). This result indicates that the BFV scheme is more efficient for the above benchmarks. However, we also noted that CKKS will outperform BFV when the shape of intermediate features enlarges (*e.g.* to 64×64). Therefore, the best scheme should be chosen according to the shape and size of the features.

Extension on RNN

In the above experiments, we have evaluated the effectiveness of our solution on CNN based models. However, it is natural to enable the secure inference of recurrent neural networks (RNNs). There have been some prior works that focus on training a Bayesian RNN. A recent work [Fortunato *et al.*, 2017] has built a Bayesian RNN with LSTM cells. This model is used for language modeling task and has achieved comparable performance compared with a normal RNN. All basic operations in the Bayesian RNN model are included in the scope of our solution. Thus we can easily integrate the model into the protocol in Algorithm 1. In the context of Bayesian RNN, GAZELLE cannot provide the support for the RNN model like our solution. This is because the computation of the Tanh/Sigmoid functions (exist in an LSTM cell) is particularly expensive under the garbled circuit protocol.

5 Conclusion

In this paper, we introduce BAYHENN, a practical solution for secure DNN inference. The key innovation lies in the strategy to combine Bayesian deep learning and homomorphic encryption. Armed with this strategy, our solution is capable of achieving secure inference of DNN models with arbitrary activation functions, while our solution enjoys 5× speedup in contrast to the best existing work. Applying this method in the DLaaS scenario is promising and implies a wide range of real-life applications. This research also points out a new direction, to apply Bayesian deep learning on the tasks about privacy protection. For example, previous works that focus on training a DNN model under differential privacy, can benefit from the insight in our paper.

Despite the superiority of our method, there is much room for further improvement. From the view of optimization, how to design a better algorithm to optimize the Bayesian neural network is crucial to improving the model accuracy. Another direction is to use some hardware devices (*e.g.* FPGAs) to accelerate the computation of secure DNN inference.

Acknowledgements

This work was supported by National Natural Science Foundation of China (No.61572045).

References

[Acar *et al.*, 2018] Abbas Acar, Hidayet Aksu, et al. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys*, 51(4):79:1–79:35, 2018.

[Bhattacharjee *et al.*, 2017] Bishwaranjan Bhattacharjee, Scott Boag, et al. IBM deep learning service. *IBM Journal of Research and Development*, 61(4):10, 2017.

[Blundell *et al.*, 2015] Charles Blundell, Julien Cornebise, et al. Weight uncertainty in neural networks. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 1613–1622. JMLR.org, 2015.

[Bourse *et al.*, 2018] Florian Bourse, Michele Minelli, et al. Fast homomorphic evaluation of deep discretized neural networks. In *Advances in Cryptology – CRYPTO 2018*, volume 10993, pages 483–512. Springer, 2018.

[Chaudhuri and Monteleoni, 2008] Kamalika Chaudhuri and Claire Monteleoni. Privacy-preserving logistic regression. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems*, pages 289–296. Curran Associates, Inc., 2008.

[Devlin *et al.*, 2018] Jacob Devlin, Ming-Wei Chang, et al. BERT: Pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[Fortunato *et al.*, 2017] Meire Fortunato, Charles Blundell, and Oriol Vinyals. Bayesian recurrent neural networks. *CoRR*, abs/1704.02798, 2017.

[Gilad-Bachrach *et al.*, 2016] Ran Gilad-Bachrach, Nathan Dowlin, et al. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33nd International Conference on Machine Learning*, volume 48, pages 201–210. JMLR.org, 2016.

[Havaei *et al.*, 2017] Mohammad Havaei, Axel Davy, et al. Brain tumor segmentation with deep neural networks. *Medical Image Analysis*, 35:18–31, 2017.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, et al. Deep residual learning for image recognition. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778. IEEE Computer Society, 2016.

[Juvekar *et al.*, 2018] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *Proceedings of the 27th USENIX Security Symposium*, pages 1651–1669. USENIX Association, 2018.

[LeCun *et al.*, 1998] Yann LeCun, Léon Bottou, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[Litjens *et al.*, 2017] Geert J. S. Litjens, Thijs Kooi, et al. A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42:60–88, 2017.

[Liu *et al.*, 2017] Jian Liu, Mika Juuti, et al. Oblivious neural network predictions via MiniONN transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 619–631. ACM, 2017.

[Ma *et al.*, 2019] Xu Ma, Xiaofeng Chen, and Xiaoyu Zhang. Non-interactive privacy-preserving neural network prediction. *Information Sciences*, 481:507–519, 2019.

[Mohassel and Zhang, 2017] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy*, pages 19–38. IEEE Computer Society, 2017.

[Phong *et al.*, 2018] Le Trieu Phong, Yoshinori Aono, et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345, 2018.

[Regev, 2009] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):34:1–34:40, 2009.

[Ren *et al.*, 2017] Shaoqing Ren, Kaiming He, et al. Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[Rouhani *et al.*, 2018] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: Scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference*, pages 2:1–2:6. ACM, 2018.

[Sanyal *et al.*, 2018] Amartya Sanyal, Matt J. Kusner, et al. TAPAS: Tricks to accelerate (encrypted) prediction as a service. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 4497–4506. JMLR.org, 2018.

[Shridhar *et al.*, 2018] Kumar Shridhar, Felix Laumann, et al. Bayesian convolutional neural networks with variational inference. *CoRR*, abs/1806.05978, June 2018.

[Tramèr *et al.*, 2016] Florian Tramèr, Fan Zhang, et al. Stealing machine learning models via prediction APIs. In *Proceedings of the 25th USENIX Security Symposium*, pages 601–618. USENIX Association, 2016.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, et al. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pages 6000–6010, 2017.

[Zhang *et al.*, 2017] Jiaqi Zhang, Kai Zheng, et al. Efficient private ERM for smooth objectives. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 3922–3928. ijcai.org, 2017.

[Zhang *et al.*, 2018] Qiao Zhang, Cong Wang, et al. GELU-Net: A globally encrypted, locally unencrypted deep neural network for privacy-preserved learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 3933–3939. ijcai.org, 2018.