frontiers in Computational Neuroscience

ORIGINAL RESEARCH
published: 23 December 2021
doi: 10.3389/fncom.2021.799977

cuSCNN: A Secure and Batch-Processing Framework for Privacy-Preserving Convolutional Neural Network Prediction on GPU

Yanan Bai[1,2], *Quanliang Liu*[2,3], *Wenyuan Wu*[1]* *and Yong Feng*[1]

[1] *Chongqing Key Laboratory of Automated Reasoning and Cognition, Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, China,* [2] *University of Chinese Academy of Sciences, Beijing, China,* [3] *Chongqing School, University of Chinese Academy of Sciences, Chongqing, China*

The emerging topic of privacy-preserving deep learning as a service has attracted increasing attention in recent years, which focuses on building an efficient and practical neural network prediction framework to secure client and model-holder data privately on the cloud. In such a task, the time cost of performing the secure linear layers is expensive, where matrix multiplication is the atomic operation. Most existing mix-based solutions heavily emphasized employing BGV-based homomorphic encryption schemes to secure the linear layer on the CPU platform. However, they suffer an efficiency and energy loss when dealing with a larger-scale dataset, due to the complicated encoded methods and intractable ciphertext operations. To address it, we propose cuSCNN, a secure and efficient framework to perform the privacy prediction task of a convolutional neural network (CNN), which can flexibly perform on the GPU platform. Its main idea is 2-fold: (1) To avoid the trivia and complicated homomorphic matrix computations brought by BGV-based solutions, it adopts GSW-based homomorphic matrix encryption to efficiently enable the linear layers of CNN, which is a naive method to secure matrix computation operations. (2) To improve the computation efficiency on GPU, a hybrid optimization approach based on CUDA (Compute Unified Device Architecture) has been proposed to improve the parallelism level and memory access speed when performing the matrix multiplication on GPU. Extensive experiments are conducted on industrial datasets and have shown the superior performance of the proposed cuSCNN framework in terms of runtime and power consumption compared to the other frameworks.

Keywords: privacy-preserving, convolutional neural network, homomorphic encryption, GPU computation, deep learning, cloud computing

OPEN ACCESS

Edited by:
Song Deng,
Nanjing University of Posts and Telecommunications, China

Reviewed by:
Huyong Yan,
Chongqing Technology and Business University, China
Jin Dong,
Oak Ridge National Laboratory (DOE), United States
Tianchen Wang,
University of Notre Dame, United States

***Correspondence:**
Wenyuan Wu
wuwenyuan@cigit.ac.cn

Received: *22 October 2021*
Accepted: *22 November 2021*
Published: *23 December 2021*

Citation:
Bai Y, Liu Q, Wu W and Feng Y (2021) cuSCNN: A Secure and Batch-Processing Framework for Privacy-Preserving Convolutional Neural Network Prediction on GPU. Front. Comput. Neurosci. 15:799977. doi: 10.3389/fncom.2021.799977

1. INTRODUCTION

Deep learning (DL) has been applied to lots of fields [e.g., visual recognition (He et al., 2016), medical diagnosis (Shen et al., 2017), risk assessment (Deng et al., 2021a,b), and a recommender system (Shi et al., 2020; Wu et al., 2021a,b)], which achieves a superior performance in comparison with human cognition. The DL with a complex neural network (DNN) structure usually requires massive data for training a high-accuracy model. To alleviate the cost of using DL models, cloud providers (e.g., Amazon, Alibaba, Microsoft) are now providing Deep Learning as a Service (DLaS) that offers DL model training and inference APIs for clients. For example,

Google AI[1] provides a series of APIs for AI services (e.g., image classification, personalization recommendation, etc.). By calling these APIs, the client can upload their plaintext data to the cloud, then receive the analysis results (e.g., predication or classification task) by paying certain fees, as shown in **Figure 1**. Due to the fact that users' queries often involve personal privacy information, such as X-ray images or user's behavior trajectory data (Wu et al., 2020), a natural yet essential question about the protection of privacy has been raised: *if massive personal data are collected for model training and prediction, will the disclosing of user-sensitive information increase?* (Riazi et al., 2019; Liu et al., 2020).

Although those cloud providers claim that they will never leak or use users' data for commercial purposes, the increasing number of user data leaks tell us that there is no guarantee on what they promised (Abadi et al., 2016). An intuitive solution to protect user's privacy during DL inference is to give users propriety to download the model from the server and run the model on their platform locally. Nevertheless, this is an undesirable result for the model-holder (e.g., company or hospital) for at least two reasons: (1) The well-trained DL model is considered as the core intellectual property for companies, which is built on the massive collection of data. To avoid the loss of profits, companies require confidentiality to preserve their competitive advantage. (2) The well-trained DL model is known to reveal information about the underlying data used for training. In the case of medical data, this reveals sensitive information about other patients, violating their privacy and perhaps even HIPAA regulations (Assistance, 2003).

Therefore, the target of our work is to design a privacy-preserving service framework where both the model-holder and client can use the well-trained DL model and private data without worries. Two important requirements should be considered:

1. For protecting the privacy of the data owner, their sensitive queries should not be revealed to the model-holder;
2. For the proprietary of the model-holder, the DL model should not be revealed to users, in order to preserve their competitive advantage.

Following this mainstream, several solutions based on various secure computing technologies have been proposed, such as homomorphic encryption (HE)-based (Dowlin et al., 2016), multi-party computing (MPC)-based (Rouhani et al., 2018), and mixed-based solutions (Juvekar et al., 2018). Among them, HE (Gentry and Craig, 2009) is an intuitive yet promising way to evaluate it, which considers the whole neural network as a function and evaluates it in the ciphertext domain thoroughly, such as CryptoNet (Dowlin et al., 2016). Secure multi-party computing is another option for secure function evaluation. Secret sharing (SS) (Shamir, 1979) and garbled circuits (GC) (Yao, 1986) are two representational methods. They can transform a neural network model into an oblivious form and evaluate it with secure two-party computation, such as MinONN (Liu et al., 2017). Besides, mixed-based solutions have been proposed to obtain better performance with trade-off for each advantage, such as Gazzle (Juvekar et al., 2018).

We notice that the CNN inference task requires a lot of inner product operations to finish the convolutional layer. The existing mix-based methods usually adopt the Chinese Remainder Theorem (CRT)-based Single Instruction Multiple Data (SIMD) schemes to execute inner product operations of privacy-preserving CNN. However, it is time-consuming, since rotating operations in privacy-preserving CNN are required to sum up the results among slots. Different with the above solutions, we adopt the GSW-based method to design the matrix multiplication method in the ciphertext space, which is the main motivation of this study. The advantage of the GSW-based solution is that the ciphertext operation is a natural matrix operation without the expensive rotate-and-add strategy. Furthermore, with the rapid development of graphics processing hardware, a GPU is becoming the standard for cloud providers, where CUDA programming makes it possible to harness the computation power of GPU efficiently. Therefore, the use of GPU technology to accelerate matrix multiplication is another important motivation of this study.

On this basis, we introduce cuSCNN, a practical realization of a mixed-based framework that supports the privacy-preserving prediction of convolutional neural networks (CNNs). CNN is one of the most popular neural network architectures in DL. Generally, a CNN model consists of convolutional layers, activation, pooling, and fully connected layers. Convolutional and fully connected layers have linear properties, while activation and pooling are non-linear layers. For cuSCNN, it employs HE to perform the linear operations (e.g., homomorphic addition and multiplication) in each layer, while conducting the non-linear activation functions and pooling operations collaboratively by employing HE and GC jointly. The main contribution of this paper is as follows:

- We propose cuSCNN, an efficient and privacy-preserving neural network prediction framework that keeps user and server data secure. We employ the optimized homomorphic matrix computations for the linear operations in CNN, while adopting GC technology to execute the non-linear operations. Our secure matrix-based computation implements linear operations in the batch mode when dealing with a large-scale dataset.
- We introduce an efficient and natural GSW-based homomorphic matrix encryption scheme to support secure matrix multiplication and addition operations. Furthermore, we propose a hybrid optimization approach to matrix multiplication on GPU to improve the computation efficiency, which combines dual-optimization for I/O and computation.
- We implement cuSCNN on real-world data with varied CNN models and evaluate its performance on the industrial dataset. The experimental results show the superiority and effectiveness of cuSCNN in terms of runtime and power consumption, compared with state-of-the-art works.

The rest of this paper is organized as follows. Section 2 gives the preliminaries. Section 3 overviews the cuSCNN framework.

[1]https://ai.google/

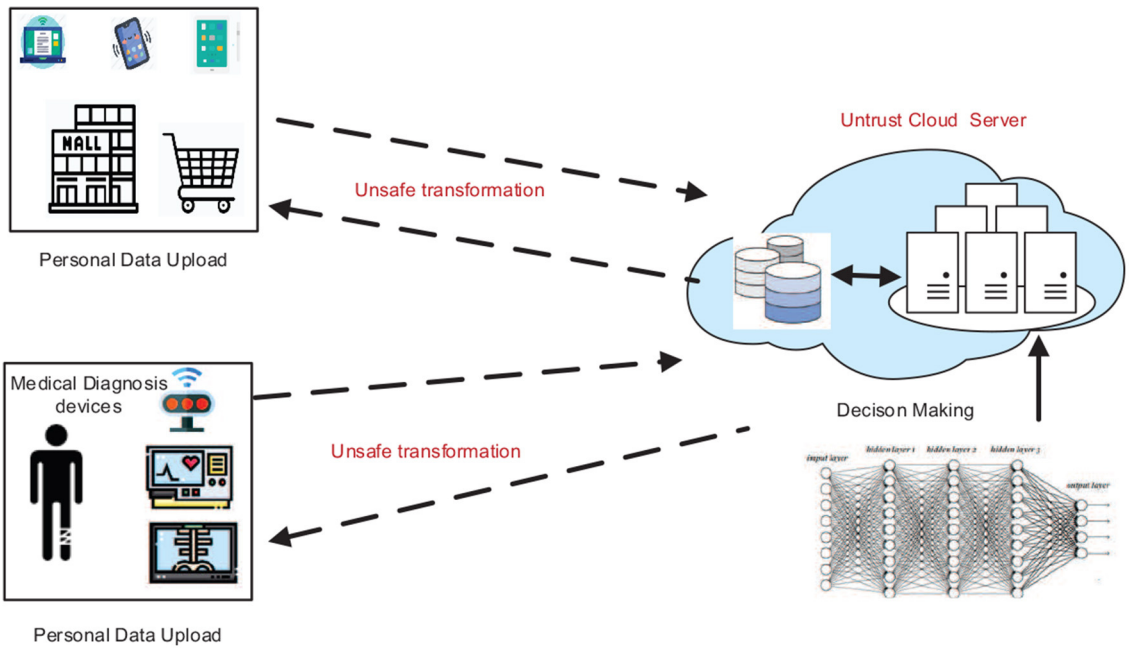

FIGURE 1 | The privacy question of the deep learning model deployed on an untrusted cloud.

Section 4 gives the implementation details of the cuSCNN framework. Section 5 evaluates the performance of cuSCNN. Finally, section 6 concludes this paper.

2. PRELIMINARIES

2.1. Related Work

2.1.1. Privacy-Preserving Neural Network Inference Framework

As the representative solution of homomorphic encryption-based solutions, CryptoNets (Dowlin et al., 2016) can evaluate the trained neural network in the ciphertext domain via utilizing leveled homomorphic encryption (LHE). However, the most critical limitation of CryptoNets is that the computational complexity drastically increases as the depth of layers in the NN model increases. Moreover, due to only adopting the LHE, non-linear functionalities such as the ReLU activation function in CryptoNets cannot be supported. To support the non-linear functionalities and pooling operations, DeepSecure (Rouhani et al., 2018) leverages GC as its backbone cryptographic engine. It can support various activations in the DL model. However, since multiplication is an atomic operation in the DL model and the number of Boolean gates in the multiplication circuit grows $2x$ times concerning the bit width of operands, together with multiple interactions between participants, DeepSecure requires an extensive communication overhead when performing secure privacy-preserving prediction. MiniONN (Liu et al., 2017) transforms a neural network model into an oblivious form and evaluates it with secure two-party computation. In detail, it utilizes the GC to compute the non-linear activation function while incorporating SS and HE-based methods to run the linear operations in the DNN model. Moreover, GAZELLE (Juvekar et al., 2018) is another mixed-protocol solution that uses an intricate combination of HE and GC to carry out the inference phase of the DNN model, which utilizes the GC to perform the non-linear activation function and uses lattice-based HE with packing technology to execute linear operations. As a result, GAZELLE improves the runtime of private inference and reduces communication between the user and the cloud. To improve the efficiency of the ciphertext computations, FALCON (Li et al., 2020) exploits the Fast Fourier transform to accelerate the homomorphic computations in the convolutional and fully connected layers. Unlike the method mentioned above, we introduce GSW-based secure matrix computations to implement the linear layers and leverage the GPU to accelerate the computation efficiency of the proposed approach.

2.1.2. Matrix-Based Homomorphic Encryption Scheme

Matrix-based computations are the core yet time-consuming operations in the neural network. In this context, some matrix-based homomorphic encryption schemes have been proposed. Based on the SIMD technology, Wu and Haven et al. proposed a safety inner product method on packed ciphertexts (Wu and Haven, 2012). Lu et al. (2016) modified the matrix-vector multiplication for secure statistical analysis over HElib. Duong et al. (2016) proposed a homomorphic matrix multiplication scheme on the packed ciphertext over RLWE. Later, Mishra et al. (2017) designed an enhanced version of the matrix multiplication, but there were useless terms in the ciphertexts. Besides, it is only suitable for a one-depth homomorphic

TABLE 1 | Meaning of notation in the homomorphic encryption scheme.

Notations	The meaning
$\|\mathbf{x}\|_\infty$	The maximum norm of $\mathbf{x}$
$\|\mathbf{x}\|_2$	The Euclidean norm of $\mathbf{x}$
$< \mathbf{x}, \mathbf{y} >$	The inner product of two vectors $\mathbf{x}$ and $\mathbf{y}$
x_i	The ith element of vector $\mathbf{x}$
$[\mathbf{X}\|\|\mathbf{Y}] \in \mathbb{Z}^{m\times(n_1+n_2)}$	The column concatenation of $\mathbf{X}$ with $\mathbf{Y}$, where $\mathbf{X} \in \mathbb{Z}^{m\times n_1}, \mathbf{Y} \in \mathbb{Z}^{m\times n_2}$
$\begin{bmatrix}\mathbf{Y}\\ \mathbf{X}\end{bmatrix} \in \mathbb{Z}^{(m_1+m_2)\times n}$	The row concatenation of $\mathbf{X} \in \mathbb{Z}^{m_1\times n}$ with $\mathbf{Y} \in \mathbb{Z}^{m_2\times n}$
$\mathbf{X}_i$	The ith column vector of $\mathbf{X}$
$\mathbf{X}(p:q, r:s)$	The submatrix consisting of rows p to q and columns r to s of the matrix $\mathbf{X}$.
$a \xleftarrow{U} \mathbf{D}$	a is chosen from set $\mathbf{D}$ uniformly at random
$\mathbf{I}_r$	The identity matrix with size of $r \times r$
$\mathbf{X}_{ij} \in \{0,1\}^{r\times r}$	The matrix with 1 in the position (i,j) and 0 in the others
λ	Security parameters, the scheme can resist 2^λ attacks
$mod q$	Modulus q with the range of values is $[-(q-1)/2, (q-1)/2]$
$round(x)$	Rounding $x \in \mathbb{R}$
$\lceil x \rceil$	Rounding up $x \in \mathbb{R}$
$\lfloor x \rfloor$	Rounding down $x \in \mathbb{R}$

multiplication scenario, due to the significant expansion rate of ciphertexts. Wang et al. (2017) modified Duong's methods for flexible matrix computation, but their modification was much less efficient for matrices of larger size. Jiang et al. (2018) presented a novel matrix encoding method that can encrypt more than one matrix in a single ciphertext and adapted an efficient evaluation strategy for generic matrix operations via linear transformations. However, the methods mentioned above were all constructed based on the second-generation HE scheme with unnecessary key switching, which suffers efficiency and precision loss when dealing with large-scale data. Hiromasa et al. (2016) first conducted a GSW-FHE scheme for matrix homomorphism computations (i.e., HAO). They optimized the bootstrapping technique proposed by Alperin-Sheriff and Peikert (2014). However, all these improvements target binary plaintext, which dramatically restricts its application in the real world.

2.2. Notations and Definitions

Assume that vectors are in column form and are written using bold lower-case letters e.g., $\mathbf{x}$, while bold capital letters are used to denote matrices, e.g., $\mathbf{X}$. We introduce gadget matrix $\mathbf{G}$ and the function G^{-1} by lemma 1. In order to facilitate readers to understand, the meanings of the notations mentioned in the encryption scheme are shown in **Table 1**.

Lemma 1 (Micciancio and Peikert, 2012). *Let matrix* $\mathbf{C} \in \mathbb{Z}_q^{n\times m}$, *there are a fixed and primitive matrix* $\mathbf{G} \in \mathbb{Z}_q^{n\times nl}$ *and a deterministic, randomized function* G^{-1} *that can be calculated by:* $\mathbb{Z}_q^{n\times m} \to \mathbb{Z}_q^{nl\times m}$ *such that* $\mathbf{X} \xleftarrow{R} G^{-1}(\mathbf{C})$ *is sub-Gaussian with parameter O(1) and always satisfies* $\mathbf{GX} = \mathbf{C}$.

Let $l = \lfloor \log q \rfloor + 1$ and $\mathbf{g}^T = (2^0, 2^1, \ldots, 2^{l-1})$, $\mathbf{I}_n$ is the unit matrix with n rank, then the gadget matrix can be defined as $\mathbf{G} := \mathbf{I}_n \otimes \mathbf{g}^T \in \mathbb{Z}_q^{n\times nl}$.

2.3. GSW-Based Homomorphic Matrix Encryption Scheme

Generally, a HE scheme consists of four algorithms HE=(Keygen, Enc, Dec, Eval) and can be illustrated as follows:

- *KeyGen*(*params*): Given the security parameter λ, the main function of *KeyGen*(*params*) is to produce a secret key **sk**, a public key **pk**, and a public evaluation key **evk**.
- $Enc_{\mathbf{pk}}(m)$: Based on the created public key **pk**, the encryption algorithm encrypts a plaintext $m \in \mathbf{M}$ into a ciphertext $c \in \mathbf{C}$.
- $Dec_{\mathbf{sk}(c)}$: Using the created secret key **sk**, it can recover the original plaintext m from the ciphertext c.
- $Eval_{evk}(f, c_1, \ldots, c_l)$: Under the ciphertext space $\mathcal{C}$ with the evaluation key **evk**, the ciphertext c_f can be calculated by using the function $f : \mathcal{M}^l \to \mathcal{M}$ to $c_1, \ldots, c_l$

The original GSW scheme is proposed by Gentry, Sahai, and Waters (Gentry and Craig, 2009). It adopts the approximate eigenvector method based on the plaintext space $\mathbf{M}$ to construct the ciphertext space $\mathbf{C}$. Based on this scheme, Bai et al. proposed a homomorphic matrix encryption scheme (Bai et al., 2020), which can be described as follows:

Given the security parameter λ and the multiplication depth of circuit L, $l = \lfloor \log q \rfloor + 1$. The integer modulus is $q = q(\lambda, L) := 2^{l-1}$, the lattice dimension $n = n(\lambda, L)$, and the noise distribution $\chi = \chi(\lambda, L)$ follows a sub-Gaussian distribution over $\mathbb{Z}$. Meanwhile, let $m = m(\lambda, L) := O((n+r)l)$, and $N := (n+r)l$. $\mathbf{G} = \mathbf{I}_{n+r} \otimes \mathbf{g}^T \in \mathbb{Z}_q^{(n+r)\times N}$ can be calculated, where $\mathbf{g}^T = \{2^0, 2^1, \ldots, 2^{l-1}\}$.

- HE.KeyGen(n, q, χ, m): The key generation method mainly includes two parts, i.e., the secret key **sk** and public key **pk**:
  - For **sk**, it first samples a secret key matrix $\bar{\mathbf{S}} \leftarrow \chi^{r\times n}$, then the secret key matrix can be obtained as follows:

$$\mathbf{S} := \left[\mathbf{I}_r || -\bar{\mathbf{S}}\right] \in \mathbb{Z}_q^{r\times(n+r)} \tag{1}$$

  - For **pk**, it first generates a uniformly random matrix $\mathbf{A} \xleftarrow{U} \mathbb{Z}_q^{n\times m}$, noise matrix $\mathbf{E} \xleftarrow{R} \chi^{r\times m}$, and $R_{ij} \xleftarrow{U} \{0,1\}^{m\times N}$ (for all $i, j = 1, \ldots, r$), then the public key matrix $\mathbf{B}$ is:

$$\mathbf{B} := \begin{bmatrix}\bar{\mathbf{S}}\mathbf{A} + \mathbf{E}\\ \mathbf{A}\end{bmatrix} \in \mathbb{Z}_q^{(n+r)\times m} \tag{2}$$

$$\mathbf{P}_{ij} := \mathbf{B}\mathbf{R}_{ij} + \begin{bmatrix}\mathbf{M}_{ij}\mathbf{S}\\ 0\end{bmatrix}\mathbf{G} \in \mathbb{Z}_q^{(n+r)\times N} \tag{3}$$

  Hence, the output of keygen(n, q, χ, m) is $\mathbf{sk} := \mathbf{S}$, $\mathbf{pk} := \left\{\mathbf{P}_{i,j}, \mathbf{B} || 1 \le i, j \le r\right\}$.
- HE.SecEnc(**sk**, **M**): Sample the random matrix $\bar{\mathbf{A}} \xleftarrow{U} \mathbb{Z}_q^{n\times N}$ and $\mathbf{E} \xleftarrow{R} \chi^{r\times N}$, then the ciphertext $\mathbf{C}$ can be computed by :

$$\mathbf{C} := \begin{bmatrix}\bar{\mathbf{S}}\bar{\mathbf{A}} + \mathbf{E}\\ \bar{\mathbf{A}}\end{bmatrix} + \begin{bmatrix}\mathbf{M}\mathbf{S}\\ 0\end{bmatrix}\mathbf{G} \in \mathbb{Z}_q^{(n+r)\times N} \tag{4}$$

- HE.PubEnc(**pk**,**M**): Sample a random matrix $R \xleftarrow{U} \{0,1\}^{m\times N}$, and the ciphertext can be denoted by

$$\mathbf{C} := \mathbf{BR} + \sum_{i=0}^{r-1}\sum_{j=0}^{r-1} M_{i,j}\cdot \mathbf{P}_{i,j} \in \mathbb{Z}_q^{(n+r)\times N} \tag{5}$$

- HE.Dec(**S**,**C**): The processing of the decryption algorithm can be described as follows:

  Step 1: Compute the matrix $\mathbf{H} = \mathbf{SC} \in \mathbb{Z}_q^{r\times N}$;

  Step 2: Denote the matrix $\mathbf{H}'_{i,j} \in \mathbb{Z}_q^{r\times rl}$, where $i \in \{1,2,\ldots,r\}$, and $j \in \{1,2,\ldots,rl\}$. Meanwhile, the noise matrix $\mathbf{E}'$ has the same size as $\mathbf{H}'$. Hence,

$$\mathbf{H}' = \mathbf{E}' + \begin{bmatrix} \mathbf{M}_{0,0} & \cdots & 2^l\mathbf{M}_{0,0} & \cdots & \mathbf{M}_{0,r} & \cdots & 2^l\mathbf{M}_{0,r} \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ \mathbf{M}_{r,0} & \cdots & 2^l\mathbf{M}_{r,0} & \cdots & \mathbf{M}_{0,r} & \cdots & 2^l\mathbf{M}_{0,r} \end{bmatrix} \tag{6}$$

  Step 3: Recover each element (i.e., $m_{i,j}$) in the plaintext matrix **M** via the function Dec1Num($\mathbf{H}'(i, jl:(j+1)l-2)$), where $1 \le i \le r$ and $1 \le j \le r$. The implementation details of Dec1Num can be found in Bai et al. (2020).

- HE.MatAdd(C_1, C_2): Given the two ciphertext matrices $\mathbf{C}_1 \in \mathcal{Z}_q^{(n+r)\times N}$ and $\mathbf{C}_2 \in \mathcal{Z}_q^{(n+r)\times N}$, the homomorphic matrix addition can be defined as:

$$\mathbf{C}_{add} = \mathbf{C}_1 + \mathbf{C}_2 \in \mathbb{Z}_q^{(n+r)\times N} \tag{7}$$

- HE.MatMult($\mathbf{C}_1, \mathbf{C}_2$): For $\mathbf{C}_1, \mathbf{C}_2 \in \mathbb{Z}_q^{(n+r)\times N}$, it first computes $G^{-1}(\mathbf{C}_2) \in 0,1^{N\times N}$, then outputs:

$$\mathbf{C_{mult}} := \mathbf{C}_1 \cdot \mathbf{C}_2 = \mathbf{C}_1 G^{-1}(\mathbf{C}_2) \in \mathbb{Z}_q^{(n+r)\times N} \tag{8}$$

To implement the privacy-preserving linear operations in cuSCNN, two kinds of homomorphic computation should be supported: HE.MatAdd and HE.MatMul. HE.Mat means that we can encrypt the plaintext matrix as approximate eigenvalues of the ciphertext matrix correspondingly, where the secret key is the eigenvector. Since the ciphertext calculation of GSW is based on the matrix computation, which cannot cause the expansion of the ciphertext dimension, it can significantly eliminate the unnecessary key conversion brought by BGV-based solutions. HE.MatAdd represents the homomorphic addition between two matrixes in the ciphertext domain, while HE.MatMul means the homomorphic multiplication between two matrices.

2.4. GPU-Based Computing

A graphics processing unit (GPU) is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device. GPU adopts a large number of computing units and ultra-long pipelines, but it only has straightforward control logic and eliminates cache. Their highly parallel structure makes them more efficient than CPUs for algorithms that process large data blocks in parallel. CUDA (an acronym for Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) model created by Nvidia, which allows GPU to be compatible with various programming languages (e.g., C++, Fortan, and Python) and applications. The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements to execute compute kernels. In CUDA, *kernels* are functions that are executed on GPU, which are executed by a batch of threads. Meanwhile, the batch of threads is organized as a grid of thread blocks. Thus, a GPU with more blocks can execute a CUDA program in less time than a GPU with fewer blocks. As shown in **Figure 2**, threads in a block are organized into small groups of 32 called *wraps* for execution on the processors, and *wraps* are implicitly synchronous; however, threads in different blocks are asynchronous. CUDA assumes that the CUDA kernel, i.e., CUDA program, is executed on a GPU (drive), and the rest of the C program is executed on the CPU (host). CUDA threads access data from multiple memory hierarchies. Each thread has a private register and local memory, and each thread block has shared memory visible to all threads within the same thread block. All threads can access global memory.

3. THE cuSCNN FRAMEWORK

In this section, we design a privacy-preserving CNN prediction framework. Consider a cloud-based medical diagnosis scenario where a user wants to know his health status from an X-ray image. In our setting, we have two roles:

1. The cloud service provider (*S*) holds trained classifier models and has resources for storage and processing. He has a business interest in computation and making predictions on encrypted data from clients.
2. Clients (*C*) are the customers of the service provider. He uploads his private image to the cloud server via API interference and receives the result by paying specific service fees.

3.1. Overview

We introduce the execution flow of cuSCNN at a high level. Suppose that client *C* owns the input data (e.g., an X-ray image) and the server *S* holds a convolutional neural network model. For client *C*, the input is private. For server *S*, the details of the trained CNN model are also private, which includes the weights of convolutional and fully connected layers. The target of cuSCNN is to preserve privacy for the client and server when performing the CNN model.

Hypothesis: Both *S* and *C* are *semi-honest* Paverd et al. (2014), we presume they follow the cuSCNN protocol and never deviate from it, although they might attempt to infer more information based on the data they receive and transmit. Specifically, *C* leaks no information about the input contents, intermediate calculation result, and classified results to the cloud. The input data are factual, never using fake data. For the cloud server, the weights of the CNN model are kept secret from the client, but it does not hide the model architecture.

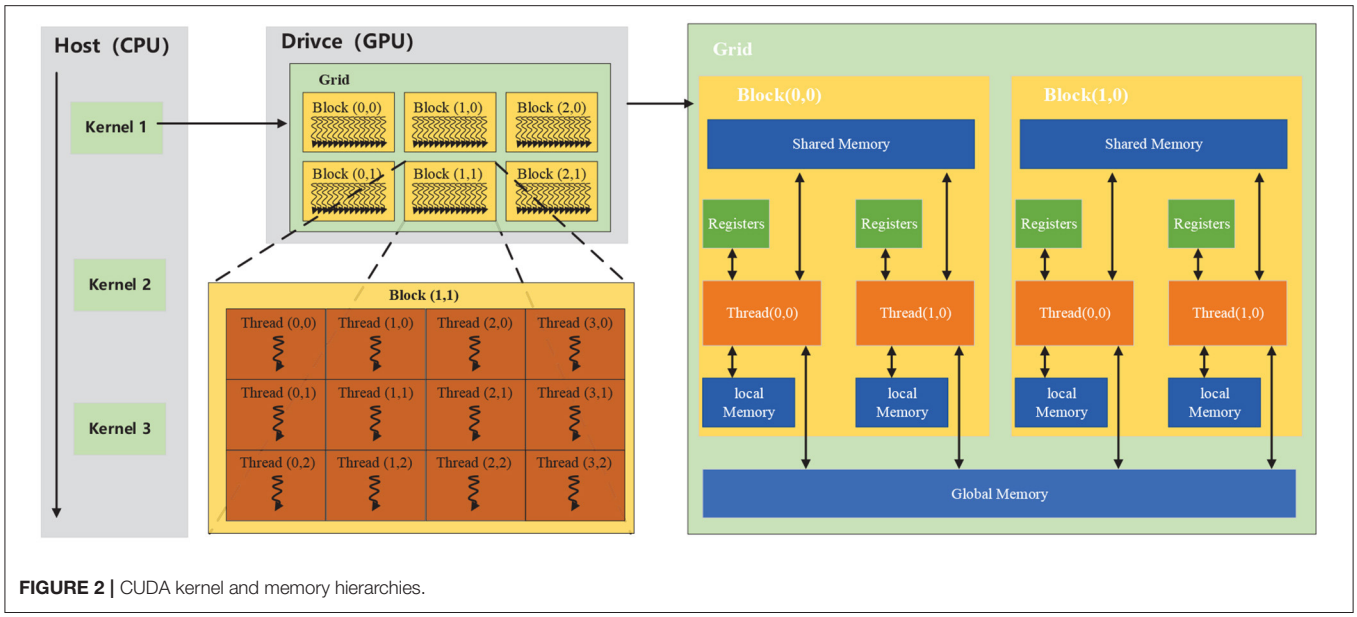

FIGURE 2 | CUDA kernel and memory hierarchies.

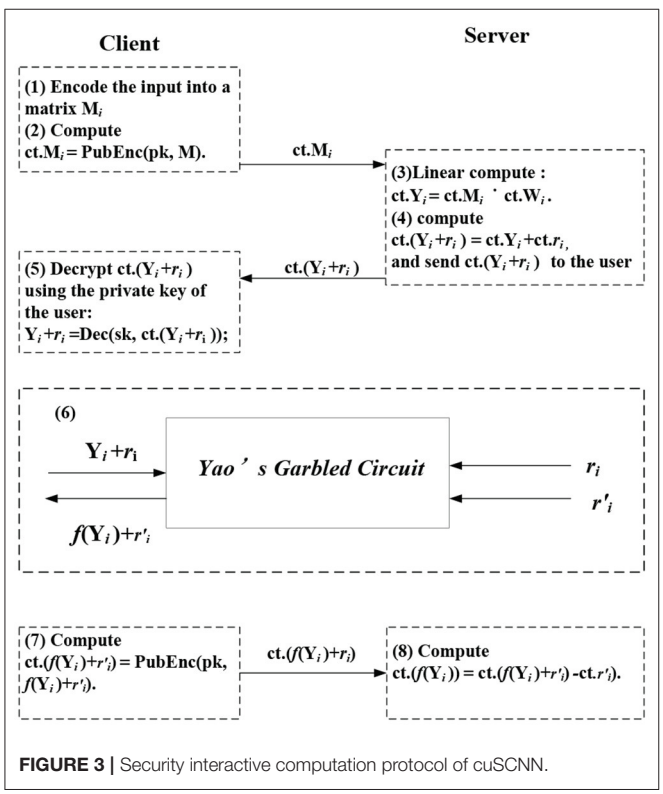

FIGURE 3 | Security interactive computation protocol of cuSCNN.

There are two phases of the framework, including off-line and online phases. In the off-line phase, the cloud generates shares $\mathbf{r}$ and $\mathbf{r}'$ used for Yao's garbled circuit. Besides, the cloud encrypts these shares and their weight matrices using the client's public key. In the online phase, the convolutional and fully connected operations are linear computations, while the activation functions are non-linear. The execution flow of the proposed cuSCNN is indicated in **Figure 3**.

- **Evaluate linear layers (i.e., Conv and FC layer)**: For $i \in [1, 2, ...l]$, l is the number of hidden layers, C firstly encodes the input data into matrix $\mathbf{M}_i$, then it encrypts $\mathbf{M}_i$ via calling the public-key encryption algorithm, denoted by $ct.\mathbf{M}_i$ = PubEnc(pk, $\mathbf{M}_i$), and uploads $ct.\mathbf{M}_i$ to the cloud server S. S utilizes the encryption scheme to execute matrix-matrix multiplication in convolutional layers and vector-matrix multiplication fully connected layers.

  Take the Conv layer, for instance, C feeds the convolutional layer with an encrypted input matrix $ct.\mathbf{M}_i$, S computes $\mathbf{Y}_i = \mathbf{W}_i \cdot ct.\mathbf{M}_i$. $\mathbf{W}$ is the filter's matrices. The fully connected layer is similar except for homomorphic vector-matrix multiplication.
- **Evaluate non-linear functions (i.e., activation and pooling layer):** S and C perform designed secure computation protocols, i.e., Yao's garbled circuit to keep data secure. Concretely, for layer i, S homomorphically adds encryption of the share $\mathbf{r}_i$ to obtain the encryption of $\mathbf{Y}_i + \mathbf{r}_i$, and send it to the client. The client decrypts it using his private key to obtain the plaintext $\mathbf{Y}_i + \mathbf{r}_i$. Next, $\mathbf{Y}_i + \mathbf{r}_i$ is held by the client and $\mathbf{r}_i$ and $\mathbf{r}'_i$ are held by the server as inputs are conducted in garbled circuit evaluation. The output of it is $f(\mathbf{Y}_i) + \mathbf{r}'_i$ (the activation function is denoted by f). Then the client encrypts it using their public key and transmits the ciphertext to the server, the server homomorphically adds the encryption of the share $\mathbf{r}'_i$ to get the encryption of $f(\mathbf{Y})_i$.

3.2. Neural Networks Architecture

In DL, CNN is a popular category of neural network, most commonly applied to analyzing visual imagery. It usually consists of an input and output layer, as well as multiple hidden layers. In most cases, a CNN takes an input and processes it through a sequence of hidden layers to classify it into one of the potential

TABLE 2 | Layers description of CNN.

Layers	Description
Layer-1[Conv-1]	Input image: 28 × 28, kernel size: 5 × 5, stride: (1,1), number of output channels: 5, padding = VALID, activation = ReLU.
Layer-2[FC-1]	Fully connecting with 5 × 3 × 3 = 845 inputs and 100 outputs, activation = ReLU.
Layer-3[FC-2]	Fully connecting with 100 inputs and 10 outputs activation = softmax.

classes. Hidden layers typically consist of a series of linear (e.g., convolutional, fully connected) layers and non-linear (e.g., activation function and pooling) layers.

For the Conv layer, the convolution operator forms the fundamental basis of the convolutional layer. It has convolutional kernels with size $k \times k$, a stride of (s, s), and a mapcount of h. Given an image $I \in \mathbb{R}^{w\times w}$ and a convolution kernel $\mathbf{W} \in \mathbb{R}^{k\times k}$, the convolved image $\mathbf{Y} \in \mathbb{R}^{d_k\times d_k}$ can be computed as follows:

$$\mathbf{Y} = Conv(I, \mathbf{W})_{i',j'} = \sum_{0\le i,j\le k} W_{i,j} \cdot I_{s\cdot i'+i, s\cdot j'+j} \tag{9}$$

where the range of (i',j') is $[0, \lceil \frac{(w-k)}{s} \rceil + 1]$, and $\lceil\cdot\rceil$ denotes the least integer greater than or equal to the input. For multiple kernel cases, it can be expressed as:

$$\mathbf{Y} = Conv(I, \mathcal{W}) = \left((Conv(I, \mathbf{W}^{(0)}), \cdots, Conv(I, \mathbf{W}^{(h-1)})\right) \in \mathbb{R}^{d_k\times d_k\times h} \tag{10}$$

For the FC layers, it connects n_I nodes to n_O nodes, which can form as the matrix-vector multiplication of an $n_O \times n_I$ matrix. Note that the output of the convolutional layer has a form of tensor, so it should be flattened before the FC layer.

4. cuSCNN DESIGN

We next utilize a commonly used CNN in privacy protection work (Dowlin et al., 2016; Rouhani et al., 2018) to describe the design details. The network topology contains one convolutional layer, one fully connected layer with ReLU activation function, and the second fully connected layer applying the softmax activation function for probabilistic classification. **Table 2** describes our neural networks to the MNIST dataset and summarizes the parameters.

4.1. Encryption of Images

We assume that a neural network is trained with the plaintext dataset in the clear. For the CNN architecture in **Table 2**, $w = 28$, $k = 5$, $d_k = 13$, $s = 2$, and $h = 5$. Suppose that the client has a two-dimensional image $I \in \mathbb{Z}^{w\times w}$. For $0 \le i,j < 5$, $0 \le i',j' < 13$, by taking the elements $I_{s\cdot i'+i, s\cdot j'+j}$, we extract the image feature to an extended matrix $\mathbf{M}$ with the size of 25×169. For bias, we add the vector $[1, \ldots, 1]^{169}$ to the first row. For a matrix $\mathbf{M}$ with the size of 26×169, it is blocked into $b_{num} = 7$ sub-matrices $\mathbf{M_b}$ for parallel computation, where $b_{num} = \lceil N_i/(k^2+1)\rceil, N_i = d_k \times d_k$. Since this CNN can deal with 846 images in FC-1, we design the framework to compute 846 images at once to achieve this maximum throughput. At the encryption phase, the client C encrypts the $\mathbf{M}_b$ using the public key of a HE scheme.

For PubEnc$(pk, \mathbf{M}_b)$, we first sample a random matrix $\mathbf{R} \leftarrow \{0,1\}^{m\times N}$ uniformly, then the encrypted image can be computed by (11).

$$ct.\mathbf{M}_b = PubEnc(pk, \mathbf{M}_b) := \mathbf{BR} + \sum_{0\le i,j\le r} M[i,j] \cdot \mathbf{P}_{(i,j)} \in \mathbb{Z}_q, \tag{11}$$

4.2. Encryption of Trained Model

The model provider encrypts the trained prediction model values such as multiple convolution kernel values $\mathbf{W}$ and weights (matrices) of FC layers.

The provider begins with a procedure for encrypting the multiple convolutional kernels. Each kernel is extended into a one-row vector of size k^2, and the bias is connected to the first column. Hence, h kernels are expanded into a matrix with a size of 5×26. Then the provider pads $(k^2+1) - h$ (i.e., 21) rows with zeros to form a square matrix. Finally, the model provider encrypts the plaintext matrix into a ciphertext, denoted by $ct.\mathbf{W}_1$.

Next, the first FC and the second layer are specified by 100×846 and 10×101 matrices. They can pad 746 and 91 rows with zeros to become two square matrices. Then the model provider encrypts the two matrices respectively, and the ciphertexts are $ct.\mathbf{W}_2$ and $ct.\mathbf{W}_3$.

4.3. Homomorphic Evaluation of Neural Networks

The public cloud takes ciphertexts of the images from the data owner and the neural network prediction model from the model provider at the prediction phase. Since the data owner uses a batch of 864 different images, the FC-1 layer is specified as a matrix multiplication: $\mathbb{Z}^{100\times 846} \times \mathbb{Z}^{846\times 846} \to \mathbb{Z}^{100\times 846}$, and the FC-2 layer is represented as a matrix multiplication: $\mathbb{Z}^{10\times 101} \times \mathbb{Z}^{101\times 101} \to \mathbb{Z}^{10\times 101}$. The FC-1 layer inputs 846 computational image results to the FC-2 layer, and the FC-2 layer can deal with 101 images at once, so the FC-2 layer needs to execute nine times to finish the 846 image prediction task.

Homomorphic Conv-1 layer: For $0 \le i < 846$, $0 \le j < 7$, the public cloud takes the ciphertexts $ct.\mathbf{M}_b^{(i,j)}$ and $ct.\mathbf{W}_1$, and it performs the following computation on ciphertexts:

$$ct.\mathbf{C_1} \leftarrow \sum_{0\le i<846, 0\le j<7} Mult(ct.\mathbf{M}_b^{(i,j)}, ct.\mathbf{W}_1) \in \mathbb{Z}_q^{(n+r)\times N}. \tag{12}$$

Secure activation layer: In order to protect the convolutional result $\mathbf{Y}$ and safely compute the activation function, the framework adopts Yao's garbled circuits method similar to GAZALLE and FALCON. The ReLU function is defined by $f(\mathbf{x}) = max(\mathbf{x}, 0)$, the cloud generates sharing r in the

preprocessing phase, C and S share the input $\mathbf{x}$ additively, i.e., S holds r, while C holds $max(\mathbf{x}, 0) - r$. The two parties jointly compute GT and MUX circuits to get $f(\mathbf{x}) + r'$, which is sent to C. C loads the 846 images to form a square matrix $\mathbf{M}_2$, then we encrypt it into ciphertext $ct.\mathbf{M}_2$ and send it to the cloud.

The FC-1 layer: The cloud firstly performs a homomorphic addition operation to remove sharing, and then it carries out the homomorphic matrix multiplication:

$$ct.\mathbf{C}_2 \leftarrow Mult(ct.\mathbf{M'}_2, ct.\mathbf{W}_2) \in \mathbb{Z}_q^{(n+r)\times N}. \quad (13)$$

Next, the cloud and the user conduct the activation operation by the garbled circuit. Afterward, the user sends the ciphertext $ct.\mathbf{C}_3$ to the cloud.

The FC-2 layer The homomorphic evaluation in FC-2 is similar to FC-1, except for executing nine times to finish 846 image predictions.

$$ct.\mathbf{C}_3^{(i)} \leftarrow Mult(ct.\mathbf{M'}_3^{(i)}, ct.\mathbf{W}_3) \in \mathbb{Z}_q^{(n+r)\times N}, 0 \le i < 9. \quad (14)$$

The activation operation of FC-2 is a softmax function, since $y_i = \frac{e^{z_i+r'}}{\sum_{j=1}^{num\_out} e^{z_j+r'}} = \frac{e^{z_i}}{\sum_{j=1}^{num\_out} e^{z_j}}$, where z_i is the ith $i \in [1, num\_out]$ input elements of the last fully connected layer, D decrypts the ciphertext and gets the prediction result directly.

Please note that the plaintext of the scheme is a square matrix, and the length of the input vector is set to 846 ($5 \times 13 \times 13 + 1$) in the example. Thus, to maximize the use of plaintext space to improve operating efficiency, we need the number of input images to be 846. In the general case, the number of input images takes the max length of the fully connected layers input vectors in the proposed framework.

4.4. Hybrid Optimization Approach on GPU for Efficient Matrix-Based Computation

To improve homomorphic matrix multiplication efficiency and utilize the powerful computing ability of GPU, we propose a hybrid optimization approach to execute the homomorphic matrix multiplication on GPU.

Given two matrices $\mathbf{A}$ and $\mathbf{B}$ with the size of $r \times r$, the straightforward way is to open a thread for computing each element of its output matrix $\mathbf{C}$. For parallel matrix multiply operation, each thread loads a row of $\mathbf{A}$ (i.e., $\mathbf{A}(\mathbf{i}, :)$) and a column of $\mathbf{B}$ ($\mathbf{B}(:, \mathbf{j})$), then c_{ij} can be computed via making an inner product of these two vectors (i.e., $c_{ij} = \mathbf{A}(\mathbf{i}, :) \cdot \mathbf{B}(:, \mathbf{j})$). However, the delay in accessing the shared memory on the GPU is quite significant (almost 100 clock cycles). For example, suppose that the matrix elements are stored in the memory following the rows first way, then a row of $\mathbf{A}$ can be saved in the memory continuously, and it can utilize the super large shared memory bandwidth of the GPU to load multiple elements with a short accessing delay. However, for the matrix $\mathbf{B}$ with a large size r, the memory address of elements in a column is internal with r elements. It means that most of the data are useless except the required column of elements in a load time. As a result, the memory access efficiency of this parallel method is appalling, since it is almost impossible for this access mode to hit the cache line.

To address this problem, we introduce a partitioning algorithm for matrix multiplication computation on GPU. For the partition method as shown in **Figure 4A**, the key is to determine *how to maximize the use of limited shared memory space*. The shared memory (SM) is an on-chip cache located on the GPU, which can be as fast as the first level cache, and threads in the same thread block can exchange data through SM. The only disadvantage is that the capacity of SM is limited. To use this small piece of high-speed memory, we divide the matrix into a set of small pieces in each dimension. Suppose that the slice size is T, the output matrix $\mathbf{C}_{00}$ can be written as:

$$\mathbf{C}_{00} = \sum_{i=0}^{bk-1} A_{0,i} \cdot B_{i,0} \quad (15)$$

where $bk = \lceil \frac{r}{T} \rceil$ is the block numbers of matrices $\mathbf{A}$ and $\mathbf{B}$. Note that the small slice matrix will degenerate into a single element when the small slice size T becomes 1. If the small piece is regarded as an element, the size of the whole matrix is reduced by T times.

Each piece of the output matrix $\mathbf{C}$ can assign a thread block with a group of threads to compute the result, where each thread corresponds to an element in the piece. In detail as shown in **Figure 4B**, each thread stores one element of block $\mathbf{B}(:, j)$ and one column of C_{ij} in its register. $\mathbf{A}(i, :)$ is stored in the shared memory of Block(0,0), which can be accessed by the threads in Block(0,0). Instead of using the inner product to perform matrix multiplication, we adopt the outer product to optimize the computation. For example, it first performs the outer product between the first column of $\mathbf{A}(:, 0)$ and the first row of $\mathbf{B}(0, :)$ and updates $\mathbf{C}_{ij}$. Then the $\mathbf{C}_{ij}$ is updated via $\mathbf{A}(:, 1)$ and $\mathbf{B}(1, :)$. Executing the iterations in a similar way until T times, the updated $\mathbf{C}_{ij}$ can be obtained. Finally, each thread stores one column of $\mathbf{C}_{ij}$ from its register to global memory.

As we know, the time complexity of naive matrix multiplication is $O(r^3)$. Due to leveraging the proposed partition method, the big matrices $\mathbf{A}$ and $\mathbf{B}$ can be divided into bk blocks with slice size T. For each slice, the time complexity is $O(T)$ when calling T threads to perform it in parallel. Hence, the total time complexity of the proposed matrix multiplication on GPU is $O((bk)^2 \times T)$.

4.5. Security Analysis

We prove that the encryption scheme defined above is IND-CPA secure under the LWE hardness assumption.

Theorem 1. *For any adversary $\mathcal{A}$ there exists an adversary such that $AdvCPA(\mathcal{A}) < 2AdvLWE(\mathcal{B})$.*

Proof: $\boldsymbol{G_0}$: A challenger $\mathcal{C}$ first initializes the encryption scheme and setup parameters, then generates a public key pk and a private key sk. The adversary $\mathcal{A}$ obtains the public key and selects two challenge plaintexts m_0 and m_1 from the plaintext space, and sends them to the challenger $\mathcal{C}$. $\mathcal{C}$ chooses $b \in [0, 1]$ at random, and encrypts m_b using the public key, then sends the ciphertext to adversary $\mathcal{A}$. The adversary guesses the plaintext

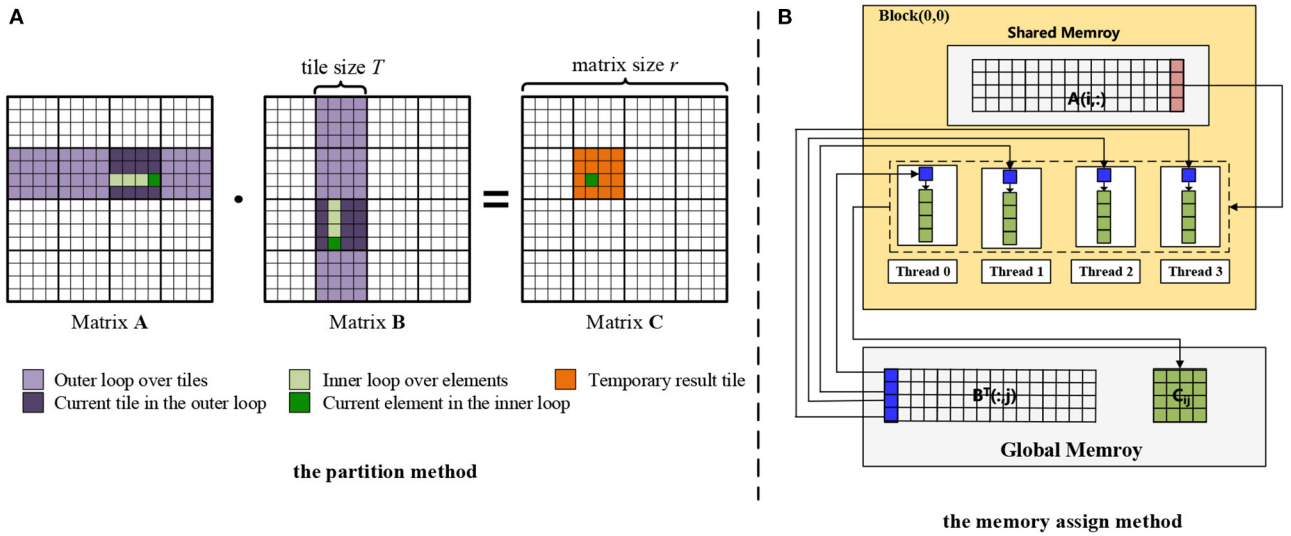

FIGURE 4 | Hybrid optimization approach on GPU.

corresponding to the ciphertext and outputs b'. If $b = b$, the adversary attacks successfully, and the advantage of the adversary is $AdvCPA(\mathcal{A}) = |Pr[b = b' \; in \; G_0] - 1/2|$.

$\boldsymbol{G_1}$: In $\boldsymbol{G_1}$, the public key $pk := \mathbf{P}_{(i,j)}, \mathbf{B}$ used in $\boldsymbol{G_0}$ is substituted by a uniform random matrix $\mathbf{B} \leftarrow \mathbb{Z}_q^{(n+r)\times m}$, and $\mathbf{P}_{(i,j)}$ is substituted by a uniform random matrix $\mathbf{P}'_{(i,j)} \leftarrow \mathbb{Z}_q^{(n+r)\times N}$. It is possible to verify that there exists an adversary $\mathcal{B}$ with the same running time, such that $|Pr[b = b' \; in \; G_1] - Pr[b = b' \; in \; G_0]| \leq AdvLWE(\mathcal{B})$, since the circular security and LWE assumption, to distinguish $\mathbf{B}$ and $\mathbf{B}'$, $\mathbf{P}$ and $\mathbf{P}'$ for $\mathcal{B}$ is almost impossible.

$\boldsymbol{G_2}$: In G_2, the value in the generation of the challenge ciphertext $\mathbf{C}$ is substituted with uniform random elements to form matrix $\mathbf{C}'$ in G_1. The adversary distinguishes $\mathbf{C}$ and $\mathbf{C}'$ which is as hard as solving the LWE problem, so there exists an adversary $\mathcal{B}$ with the same running time as that of $|Pr[b = b' \; in \; G_2]Pr[b = b' \; in \; G_1]| \leq AdvLWE(\mathcal{B})$. Notice that in G_2, the values in $\mathbf{C}$ from the challenge ciphertext are independent of bit b, hence, $Pr[b = b' \; in \; G_2] = 1/2$.

In summary, AdvCPA($\mathcal{A}$) < 2AdvLWE($\mathcal{B}$).

5. EXPERIMENTAL EVALUATION

In this section, we conduct extensive experiments on a real network to evaluate the effectiveness of the proposed cuSCNN. We mainly focus on the following questions (RQs):

- **RQ1**: How the performance of the proposed matrix multiplication method performs;
- **RQ2**: How the proposed homomorphic matrix encryption scheme performs compared to the existing methods;
- **RQ3**: How the performance of cuSCNN on each layer compares to the state-of-the-art networks.

5.1. Experimental Settings

We implement cuSCNN in C++. Specifically, we use cuBLAS library to implement the matrix multiply operations on GPU, and utilize the ABT framework to implement Yao's garbled circuits. For the homomorphic matrix encryption scheme, we set the plaintext module $q = 2^{30}$ (i.e., $l = 30$), which has a 30-bit length and is enough for all the intermediate values. The generation noise follows sub-Gaussian distribution with variance $var = q/8m$, $n = 600$, the security level can achieve 128.

We tested cuSCNN on two computers, both of which are equipped with Intel Xeon(R) E5-2680 CPU with 4 2.40 Hz cores, and a GeForce GTX 1080Ti GPU. The operation system is CentOS 7.9. One of them worked as client C, and the other play as server S. We took experiments in the LAN setting similar to previous work (Juvekar et al., 2018; Li et al., 2020). Each experiment was repeated 100 times and we report the mean in this paper.

The MNIST database (Modified National Institute of Standards and Technology database) is a dataset of images representing handwritten digits by more than 500 different writers. It is commonly used as a benchmark for machine learning systems. The MNIST database contains 60,000 training images and 10,000 testing images. The format of the images is 28×28 and the integer value of each pixel represents a level of gray with a range 0 to 255. Moreover, each image is labeled with the digit it depicts.

5.2. Performance of Matrix Multiplication on GPU

In this part, we test the timing performance of proposed optimization methods on matrix multiplication, which is the core and time-consuming operation in DL-based applications. In our method, the matrix tile size (T) is a key factor. We set the matrix size to 1,024 (i.e., $r = 1,024$), and the range of

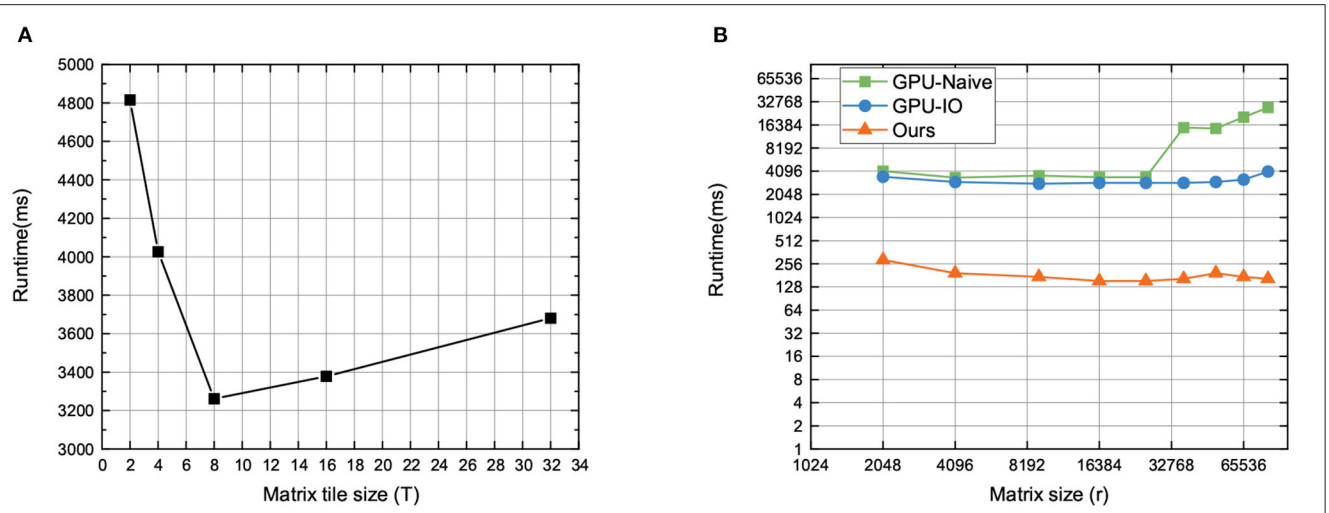

FIGURE 5 | Performance of matrix multiplication methods on GPU.

tile size is [2, 4, 8, 16, 24, 32]. The test results are shown in **Figure 5A**. We can observe that the runtime of the proposed method varies with different matrix tiles, and the optimized performance is achieved when the matrix tile size is 8. On the one hand, the inner reason is that the number of threads in each block is decreasing, but the amount of shared memory required in each block is not decreasing, after continuously increasing the matrix tile size. As a result, it will reduce the number of active threads in a streaming multiprocessor (SMP) due to the limited total number of blocks. That is, the occupancy will be reduced. In addition, calculating more elements per thread uses more registers. The number of registers in each thread will in turn affect the number of active threads in SMP, and then affect occupancy.

Then, we evaluate the proposed matrix multiplication method with two baselines on GPU. In detail, we adopt three different methods to execute matrix multiplication, including the naive way (i.e., GPU-Naive), I/O optimization (i.e., GPU-IO) method, and our optimization method. The GPU-Naive method only adopts the straightforward method to perform matrix multiplication, without considering the effect of matrix split and reunion in memory, while the GPU-IO method adopts the block matrix multiplication with matrix split, without considering the matrix reunion in memory. **Figure 5B** is the running time of HE.MatMult with different methods. We find that: (1) Our proposed optimization method has the best effectiveness with varying matrix size, since the running time of our methods is the lowest compared to the other methods; (2) with increasing matrix size, our method can maintain stable execution efficiency with little running time increased. That is because our method can effectively reduce the influence of IO bandwidth on performance by jointly using shared memory and registers. Furthermore, it has a higher computation efficiency via the fine-grained blocking method. Therefore, it can make more efficient use of GPU hardware computing resources.

TABLE 3 | The comparison result of homomorphic matrix encryption schemes.

Matrix size	Method	Enc(s)	HE.MatAdd(s)	HE.MatMult(s)	Dec. (s)
32 × 32	selMC	6.998	7.345	10.639	0.0768
	Jiang's	0.09	0.01	15.592	0.0543
	Ours	0.679	0.204	**0.946**	0.067
64 × 64	selMC	7.82	8.21	12.287	0.312
	Jiang's	0.196	0.01	37.793	0.705
	Ours	0.8	0.233	**1.24**	0.222
128 × 128	selMC	9.843	10.402	15.824	1.305
	Jiang's	–	–	–	–
	Ours	1.127	0.291	**1.525**	0.862

Bold values indicate our methods have a lower running time than the comparison methods.

5.3. Performance of Homomorphic Matrix Encryption Scheme

In this part, we test the performance of our method compared with Jiang's scheme (Jiang et al., 2018) and seIMC (Bai et al., 2020). For Jiang's method, it is a BGV-based secure matrix computation scheme that includes a novel matrix encoding method and an efficient evaluation strategy for basic matrix operations (e.g., matrix addition and multiplication). For seIMC, it is a GSW-based secure matrix computation scheme. We set the security level of seIMC and Jiang to 80 in this experiment. To achieve this security level, the cyclotomic ring dimension of our homomorphic encryption is chosen as $n = 450$, based on the estimator of Albrecht et al. (2015). The parameter settings of Jiang's and SeIMC schemes are the same as in Jiang et al. (2018) and Bai et al. (2020). **Table 3** is the comparison results of the three mentioned secure matrix computation schemes. From the result, we find that: (1) Compared to the BGV-based scheme (i.e., Jiang's scheme) and SeIMC, the running time of our

TABLE 4 | Benchmarks of cuSCNN in Conv and FC layers.

Layer	Input	Filter/output	Time (ms)			Time per image (ms)
			Setup	Online	Total	
Conv layer	(28 × 28 × 1, 846)	(5 × 5 × 1, 5)	2696.9636	0.0074	2696.971	3.19
FC layer	(846, 846)	(100, 846)	820.523	0.077	820.6	0.97
	(101, 846)	(10, 846)	760.109	0.091	760.2	0.9

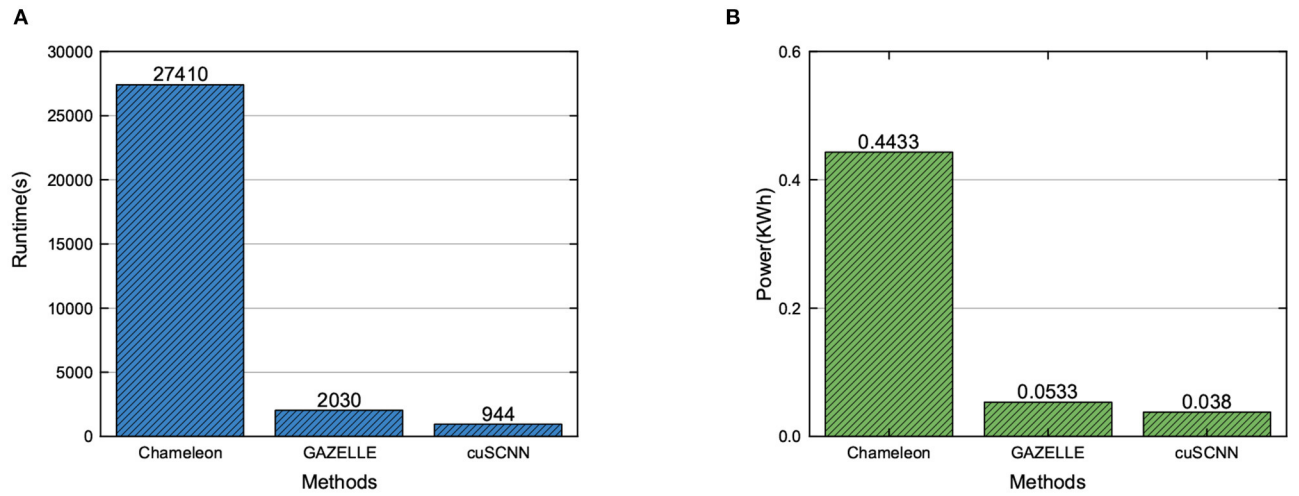

FIGURE 6 | Performance comparison of privacy-preserving neural network frameworks in runtime and power consumption.

GSW-based scheme is faster in terms of homomorphic matrix multiplication and decryption. (2) GSW-based solutions can deal with a large-scale matrix, while Jiang's scheme fails to cope with it. Hence, the results demonstrate that our secure matrix computation solution is more suitable for real applications with large-scale data.

5.4. Performance Evaluation for cuSCNN

In this part, we evaluated our cuSCNN framework in an individual layer, and compared it with state-of-the-art methods. By using the proposed homomorphic matrix encryption to secure matrix computations, Conv and FC layers are the main advantage in cuSCNN. For the implementation of cuSCNN, we replace implementations of Conv and FC layers in GAZELLE with proposed optimization methods, while we also adopt the GC to perform the ReLU operation.

Runtime of each layer required for cuSCNN are presented in **Table 4**. Furthermore, we set $T = 8$ for all of the matrix multiplication operations on GPU.

In **Table 4**, we present the timing result of Conv and FC layers with different input sizes. We notice that: (1) Due to adopting the GPU to accelerate the online computing part, the running time of the online part is less then 1 ms either in the Conv layer or FC layer. Hence, the dominant cost of evaluating cuSCNN is that of performing the setup part, including the memory switch between CPU and GPU, the assignment, and initialization operations. (2) Compared to the FC layer, cuSCNN spends almost 3× more time executing the Conv layer's convolutional operations.

Finally, we evaluate the performance of the cuSCNN framework on the MNIST dataset, compared to the previous approaches. For comparison with previous approaches, we adopt the same CNN network architecture for all mentioned models. The CNN model takes a gray scale image with size 28 × 28 as input and has one Conv, two FC, and two ReLU layers. As the comparison framework is performed on a CPU perform, to conduct a fair comparison, we present the runtime (including computation time and communication time) and power consumptions of different models when dealing with 10,152 images. The images are able to predict with 99.1% accuracy. For the power consumption of each approach, we adopt a similar method as proposed in Tian et al. (2018). The compared results are shown in **Figure 6**. From the figure, we can find that: (1) Compared to the existing MPC-based framework, the mixed frameworks can enjoy a better runtime and power consumptions, which can trade-off the advantage of different secure computation technologies, as shown in **Figure 6A**. (2) The performance of cuSCNN outperforms GAZELLE in terms of runtime and power consumption, as shown in **Figure 6B**. That is because cuSCNN adopts the matrix-matrix multiplication to perform the Conv and FC layers, while GAZELLE utilizes the matrix-vector multiplication to finish these layers. Thus, cuSCNN can execute a set of images in one iteration. With the advantage of GPU's

powerful computing ability, cuSCNN designed a hybrid parallel approach to implement the homomorphic matrix computations in Conv and FC layers. Therefore, it demonstrates that cuSCNN has a higher efficiency in executing the privacy-preserving neural networks.

6. CONCLUSION

The increasing popularity of cloud-based deep learning poses a natural question about privacy protection: if massive personal data are collected for model training and prediction, will this result in a rise in disclosing sensitive information? This paper focuses on tackling the privacy-preserving deep learning problem of a client that wishes to classify private images utilizing a convolution neural network (CNN) trained by a cloud server. Our target is to build efficient protocols whereby the cloud server executes the prediction task but also allows both client and model data to remain private. We find that matrix-based computations are the core operations in the neural network prediction task. However, the existing solutions have the limitations of computational efficiency and perform in a serial mode. To track it, this study proposes cuSCNN, a secure and efficient framework to perform the privacy prediction task of a convolution neural network, which utilizes the HE and GC jointly in a batch mode. The hybrid optimization approach is proposed to accelerate the execution of secure matrix computations on GPU to deal with the large-scale dataset. Extensive experiments conducted on the real network show that cuSCNN achieves a better performance on running time and power consumption than the state-of-the-art methods, when dealing with the larger-scale dataset. In the next step, we will conduct comprehensive experiments on different GPUs to evaluate the performance of the proposed method, including at the server level, desktop level, and embedded levels.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author/s.

AUTHOR CONTRIBUTIONS

YB completed the framework's design, implemented the encryption scheme, and wrote and revised this paper. QL performed the optimization method on GPU for matrix multiplication. WW gave the main idea of the experiment flow design. YF made constructive suggestions on the organization, writing, and revision of the paper. All authors contributed to the article and approved the submitted version.

FUNDING

This work was supported in parts by the National Key Research and Development Project (2020YFA0712303), in parts by Chongqing Research Program (cstc2019yszx-jcyjX0003, cstc2020yszx-jcyjX0005, cstc2021yszx-jcyjX0004), in parts by Guizhou Science and Technology Program ([2020]4Y056) and NSFC (11771421), in parts by Youth Innovation Promotion Association of CAS (2018419), in parts by the Key Cooperation Project of Chongqing Municipal Education Commission (HZ2021017, HZ2021008), in parts by CAS "Light of West China" Program.

REFERENCES

Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., et al. (2016). "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, (Vienna), 308–318.

Albrecht, M. R., Player, R., and Scott, S. (2015). On the concrete hardness of learning with errors. *J. Math. Cryptol.* 9, 169–203. doi: 10.1515/jmc-2015-0016

Alperin-Sheriff, J., and Peikert, C. (2014). "Faster bootstrapping with polynomial error," in *Annual Cryptology Conference* (Santa Barbara, CA: Springer), 297–314.

Assistance, H. C. (2003). *Summary of the Hipaa Privacy Rule*. Office for Civil Rights (Washington, D.C.).

Bai, Y., Shi, X., Wu, W., Chen, J., and Feng, Y. (2020). seimc: a gsw-based secure and efficient integer matrix computation scheme with implementation. *IEEE Access* 8, 98383–98394. doi: 10.1109/ACCESS.2020.2996000

Deng, S., Cai, Q., Zhang, Z., and Wu, X. (2021a). User behavior analysis based on stacked autoencoder and clustering in complex power grid environment. *IEEE Trans. Intell. Transport. Syst*. doi: 10.1109/TITS.2021.3076607

Deng, S., Chen, F., Dong, X., Gao, G., and Wu, X. (2021b). Short-term load forecasting by using improved gep and abnormal load recognition. *ACM Trans. Intern. Technol.* 21, 1–28. doi: 10.1145/3447513

Dowlin, N., Giladbachrach, R., Laine, K., Lauter, K. E., Naehrig, M., and Wernsing, J. (2016). "Cryptonets: applying neural networks to encrypted data with high throughput and accuracy," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning* (New York, NY), 48, 201–210.

Duong, D. H., Mishra, P. K., and Yasuda, M. (2016). Efficient secure matrix multiplication over lwe-based homomorphic encryption. *Tatra Mountains Math. Publ.* 67, 69–83. doi: 10.1515/tmmp-2016-0031

Gentry and Craig (2009). Fully homomorphic encryption using ideal lattices. *Stoc* 9, 169–178. doi: 10.1145/1536414.1536440

He, K., Zhang, X., Ren, S., and Sun, J. (2016). "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Las Vegas), 770–778.

Hiromasa, R., Abe, M., and Okamoto, T. (2016). Packing messages and optimizing bootstrapping in gsw-fhe. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* 99, 73–82. doi: 10.1587/transfun.E99.A.73

Jiang, X., Kim, M., Lauter, K., and Song, Y. (2018). "Secure outsourced matrix computation and application to neural networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto), 1209–1222.

Juvekar, C., Vaikuntanathan, V., and Chandrakasan, A. (2018). "{GAZELLE}: A low latency framework for secure neural network inference," in *27th {USENIX} Security Symposium ({USENIX} Security 18)* (Baltimore, MD), 1651–1669.

Li, S., Xue, K., Zhu, B., Ding, C., Gao, X., Wei, D., et al. (2020). "Falcon: a fourier transform based approach for fast and secure convolutional neural network predictions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (Seattle, WA), 8705–8714.

Liu, A., Shen, X., Xie, H., Li, Z., Liu, G., Xu, J., et al. (2020). Privacy-preserving shared collaborative web services qos prediction. *J. Intell. Inform. Syst.* 54, 205–224. doi: 10.1007/s10844-018-0525-4

Liu, J., Juuti, M., Lu, Y., and Asokan, N. (2017). Oblivious neural network predictions via minionn transformations, 619–631.

Lu, W.-J., Kawasaki, S., and Sakuma, J. (2016). Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data. *Cryptol. Arch*. doi: 10.14722/ndss.2017.23119

Micciancio, D., and Peikert, C. (2012). "Trapdoors for lattices: Simpler, tighter, faster, smaller," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (Zagreb: Springer), 700–718.

Mishra, P. K., Duong, D. H., and Yasuda, M. (2017). "Enhancement for secure multiple matrix multiplications over ring-lwe homomorphic encryption," in *International Conference on Information Security Practice and Experience* (Melbourne: Springer), 320–330.

Paverd, A., Martin, A., and Brown, I. (2014). *Modelling and Automatically Analysing Privacy Properties for Honest-But-Curious Adversaries*. Univ. Oxford Tech. Rep.

Riazi, M. S., Rouani, B. D., and Koushanfar, F. (2019). Deep learning on private data. *IEEE Secur. Privacy* 17, 54–63. doi: 10.1109/MSEC.2019.2935666

Rouhani, B. D., Riazi, M. S., and Koushanfar, F. (2018). "Deepsecure: scalable provably-secure deep learning," in *Proceedings of the 55th Annual Design Automation Conference* (San Francisco), 2, 1–6.

Shamir, A. (1979). How to share a secret. *Commun. ACM* 22, 612–613. doi: 10.1145/359168.359176

Shen, D., Wu, G., and Suk, H.-I. (2017). Deep learning in medical image analysis. *Annu. Rev. Biomed. Eng.* 19, 221–248. doi: 10.1146/annurev-bioeng-071516-044442

Shi, X., He, Q., Luo, X., Bai, Y., and Shang, M. (2020). Large-scale and scalable latent factor analysis via distributed alternative stochastic gradient descent for recommender systems. *IEEE Trans. Big Data*. doi: 10.1109/TBDATA.2020.2973141

Tian, W., He, M., Guo, W., Huang, W., Shi, X., Shang, M., et al. (2018). On minimizing total energy consumption in the scheduling of virtual machine reservations. *J. Netw. Comput. Appl.* 113, 64–74. doi: 10.1016/j.jnca.2018.03.033

Wang, L., Aono, Y., and Phong, L. T. (2017). "A new secure matrix multiplication from ring-lwe," in *International Conference on Cryptology and Network Security* (Hong Kong: Springer), 93–111.

Wu, D., and Haven, J. (2012). Using homomorphic encryption for large scale statistical analysi, FHE-SI-Report. *Univ. Stanford Tech. Rep. TR-dwu4*.

Wu, D., He, Y., Luo, X., and Zhou, M. (2021a). A latent factor analysis-based approach to online sparse streaming feature selection. *IEEE Trans. Syst. Man Cybern. Syst*. doi: 10.1109/TSMC.2021.3096065

Wu, D., Luo, X., Shang, M., He, Y., Wang, G., and Wu, X. (2020). A data-characteristic-aware latent factor model for web services qos prediction. *IEEE Trans. Knowl. Data Eng*. doi: 10.1109/TKDE.2020.3014302

Wu, D., Shang, M., Luo, X., and Wang, Z. (2021b). An l1-and-l2-norm-oriented18 latent factor model for recommender systems. *IEEE Trans. Neural Netw. Learn. Syst*. doi: 10.1109/TNNLS.2021.3071392

Yao, A. C. (1986). "How to generate and exchange secrets," in *27th Annual Symposium on Foundations of Computer Science* (Toronto: IEEE), 162–167.

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.