LIME: High-Performance Private Inference with Lightweight Model and Batch Encryption*

HUAN-CHIH WANG and JA-LING WU, National Taiwan University, Taiwan

The rapid pace of artificial intelligence (AI) and machine learning techniques has necessitated the development of large-scale models that rely on energy-intensive data centers, thereby raising environmental sustainability. Simultaneously, the increasing significance of privacy rights has led to the emergence of Privacy-Preserving Machine Learning (PPML) technologies, which aim to ensure data confidentiality. Although homomorphic encryption (HE) facilitates computations on encrypted data, it entails considerable computational costs and challenges, which impede the effective deployment of privacy-enhancing applications with large models.

To create a more sustainable and secure AI world, we propose LIME, a pure HE-based PPML solution, by integrating two techniques: element-wise channel-to-slot packing (ECSP) and power-of-two channel pruning (PCP). ECSP leverages abundant slots to pack multiple samples within ciphertexts, facilitating batch inference. PCP prunes the channels of convolutional layers by powers of two, thereby reducing computational demands and enhancing the packing capabilities of pruned models. Additionally, we implement the ReLU-before-addition block in ResNet to mitigate accuracy degradation caused by approximations with quadratic polynomials.

We evaluated LIME using ResNet-20 on CIFAR-10, VGG-11 on CIFAR-100, and ResNet-18 on Tiny-ImageNet. Using the original models, LIME attains up to 2.1% and 8.4% accuracy improvements over the methods of Lee et al. (IEEE ACCESS'21) and AESPA (arXiv:2201.06699), which employ high- and low-degree polynomial ReLU approximations, respectively. Even with 75% parameter pruning, LIME retains higher accuracy than AESPA. Using the state-of-the-art ORION (ASPLOS '25) as the convolution backend and evaluating on the original models, LIME achieves speedups of 41.5× and 8× over ORION integrated with Lee et al. and AESPA, respectively. For models pruned by 90%, these speedups increase to 202.5× and 35.1×, respectively.

1 Introduction

Over the past decade, artificial intelligence (AI) has made significant advancements across various fields, including healthcare and finance [41]. The success of machine learning (ML) models has driven researchers to develop increasingly larger models for more complex tasks, and the largest model has expanded over one hundred billion parameters since the 2020s [9]. This growth necessitates the establishment of large, energy-intensive data centers, which have become a standard in the competitive AI landscape [7]. However, the substantial energy consumption associated with AI development raises concerns regarding environmental sustainability [22]. In response, the notion of green learning has emerged, focusing on ecological sustainability and energy efficiency. By employing innovative algorithms and architectures, green learning aims to achieve superior performance with significantly reduced computational demands and environmental impacts, thereby contributing to the development of a cleaner, energy-efficient AI ecosystem [29].

Alongside the flourishing of AI, another issue that cannot be overlooked is the protection of privacy rights. The establishment of regulations, such as the European Union's General Data Protection Regulation (GDPR), has imposed stricter standards on personal data, making the transmission or storage of data on cloud service servers potentially non-compliant [37]. As a result, most ML-based services need to adapt. In this context, Privacy-Preserving Machine Learning (PPML) technology has emerged, focusing on data analysis and machine learning while protecting user privacy, with the intention of safeguarding data confidentiality [39]. Among various methods, homomorphic encryption (HE) stands out as a promising solution, as it enables computations to be performed on encrypted data without requiring

*This is the full version of a paper accepted for publication in ACM Transactions on Multimedia Computing, Communications, and Applications (ACM TOMM), Special Issue on Green Learning for Multimedia Computing. Please cite the official ACM TOMM version [38].

Authors' Contact Information: Huan-Chih Wang, whcjimmy@cmlab.csie.ntu.edu.tw; Ja-Ling Wu, wjl@cmlab.csie.ntu.edu.tw, National Taiwan University, Taipei, Taiwan.

decryption [1]. However, the trade-off for privacy protection involves a substantial increase in computational load, rendering operations approximately four to five orders of magnitude slower than those performed on plaintext, in addition to the difficulties associated with computing nonlinear functions [33, 35]. These difficulties significantly degrade the performance of large models, making it challenging for practical applications to utilize them effectively.

To create a more sustainable and secure AI world, we approach the development of energy-efficient PPML from the perspectives of batch inference and model compression. Batch inference [36] allows for the merging and processing of large amounts of data with the same format, yielding inference results for all data simultaneously. Compared to frequently and repetitively calling model inference, batch inference saves computational time and resources for each data point [2, 30]. Additionally, smaller models have fewer parameters than large models with millions of parameters, resulting in lower computational demands and easier deployment. This, in turn, offers faster inference speeds and being more suitable for resource-constrained scenarios [9, 27].

In summary, we propose *LIME*, an HE-based PPML solution that integrates batch encryption and model compression to leverage the deployment of large-scale models for PPML applications. Our main contributions are as follows:

- The element-wise channel-to-slot packing (ECSP) is utilized in LIME for batch encryption, facilitating batch inference. It utilizes slots within ciphertexts to pack channels from multiple samples for inference. Featuring two designated packing styles and a specifically designed evaluation block, ECSP is optimized for convolutional neural network (CNN) models and can effectively handle layers with wide channels. Our microbenchmark shows that ECSP-based convolution is up to 1,350× faster than a series of packing methods, including MPCNN [23], FHELIPE [21], and ORION [10].
- The power-of-two channel pruning (PCP) in LIME is a method for pruning the channels of convolutional layers using powers of two. This procedure not only compresses the model and reduces the computational requirements for ECSP-based convolutions but also increases the packing capabilities of the pruned model.
- LIME employs a quadratic polynomial to approximate ReLU functions. To alleviate accuracy degradation, it adopts a ReLU-before-addition (RBA) block for constructing ResNet models without introducing additional computational cost. Subsequently, knowledge distillation is applied for model retraining.

We evaluate the performance using ResNet-20 on CIFAR-10, VGG-11 on CIFAR-100, and ResNet-18 on Tiny-ImageNet datasets. The experimental results are summarized below:

- LIME demonstrates superior accuracy compared to Lee et al. [24] (which employs high-degree polynomial ReLU approximations) and AESPA [32] (which uses quadratic ReLU approximations) across all evaluation scenarios. Using the original models, LIME achieves accuracy improvements of up to 2.1% and 8.4% over Lee et al. [24] and AESPA [32], respectively. When employing models compressed by PCP, LIME can prune at least 75% of the parameters while maintaining higher accuracy comparable to that of AESPA [32].
- For end-to-end evaluations, we evaluate our results with the state-of-the-art (SoTA), ORION [10]. Using the original models, LIME achieves 41.5× and 8× speedups over ORION-M (ORION [10] integrated with Lee et al. [24]) and ORION-A (ORION [10] integrated with AESPA [32]), respectively. With up to 90% of parameters pruned, LIME gains a 4× increase in batching capabilities (up to 2048 samples for ResNet-20 and 256 samples for VGG-11 and ResNet-18). It achieves speedups of up to 202.5× and 35.1× over ORION-M and ORION-A, respectively.

2 Preliminaries

This section introduces the background knowledge, including the components of the CNN model in §2.1, HE in §2.2, the HE-based inference workflow in §2.3, and our thread model in §2.4. The notations used throughout this paper are listed in Table 1.

Table 1. A summary of notations.

Notation	Description
x	Scalar
$\mathbf{x}$, $\mathbf{x}_i$	Vector and the i-th element of vector $\mathbf{x}$
$\mathbf{X}$, $\mathbf{X}_i$, $\mathbf{X}_{i:i+a,j:j+b}$	Multi-dimensional tensor, elements of $\mathbf{X}$ where the first dimension is i, and elements where the first dimension ranges from i to $i+a-1$ and the second dimension ranges from j to $j+b-1$
N, Q, S	Lattice dimension, ciphertext modulus, and number of slots of a ciphertext
$[x]$	Encrypted scalar
$\oplus$, $\otimes$	HE addition and multiplication
c_w	Number of the widest output channel of a CNN model
c_i, c_o, n_c	Number of the input, output, and intermediate channels of a convolutional layer
B	Number of samples packed in the widest channel-aware dense style packing
n_s, n_r	Number of subsets and rounds of an evaluation block

2.1 An Introduction to CNN

The CNN is an ML model inspired by the biological neural network, which consists of many layers, each containing many neurons (nodes). Each neuron is connected to some of the neurons in the previous layers, so the information in a node is the weighted sum of the nodes to which it is connected. A collection of samples designated for inference serves as the input to the CNN model, producing the corresponding predictive outcomes. The following paragraph primarily introduces the key components of a CNN model.

Convolution Layer. The convolution layer performs convolutions on a three-dimensional (3-D) data $\mathbf{X} \in \mathbb{R}^{W\times H\times c_i}$ using a 4-D kernel $\mathbf{K} \in \mathbb{R}^{r\times r\times c_i\times c_o}$, where c_i and c_o are the number of input and output channels, respectively. Usually, a bias vector $\mathbf{b} \in \mathbb{R}^{c_o}$ is also applied. Therefore, assume the stride step is l, the resulting output $\mathbf{Y}$ will be $W' \times H' \times c_o$, where $W' = \lceil (W-r)/l \rceil$ and $H' = \lceil (H-r)/l \rceil$. During a convolution, a total of $n_c = c_i c_o$ channels of feature maps will be generated, and the value of the k-th output feature map is derived from convolving each kernel across all input feature maps and summing the results at the end. That is,

$$\mathbf{Y}_{k,i,j} = \sum_{c\in C}\sum_{a\in r}\sum_{b\in r} \mathbf{X}_{c,i\times l+a,j\times l+b} \times \mathbf{K}_{a,b,c,k} + \mathbf{b}_k. \tag{1}$$

Dense Layer. The dense layer is also known as the fully-connected layer because each neuron is connected to each neuron in the previous layer. Assume the input and output of a dense layer are $\mathbf{x}$ and $\mathbf{y}$, with a weight matrix $\mathbf{W}$ and a bias vector $\mathbf{b}$, then a dense layer behaves as:

$$\mathbf{y} = \mathbf{x}\mathbf{W} + \mathbf{b}.$$

Activation Function. Activation functions usually are typically applied after linear layers to enable the model to learn with more complex patterns. ReLU is a commonly used non-linear transformation after linear layers, which is defined as:

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}.$$

Batch Normalization. Batch normalization is a technique to accelerate the training process by mitigating internal covariate shifts. It normalizes data to fit Gaussian distribution followed by an affine transformation with parameters scale γ and shift β. To conclude, given a vector of data $\mathbf{x}$, assuming the mean and variance of $\mathbf{x}$ are μ and σ^2, the batch normalization calculates each element by

$$\mathbf{x}_i^{'} = \frac{\gamma(\mathbf{x}_i - \mu)}{\sqrt{\sigma^2 + \epsilon}} + \beta,$$

where ϵ is a very small error.

Mean-pooling Function. The $m \times m$ mean-pooling function partitions the input into non-overlapping regions and computes the average of the elements within each region. Given a $W \times H$ matrix $\mathbf{X}$, the output $\mathbf{X}'$ is of the shape $\frac{W}{m} \times \frac{H}{m}$, where

$$\mathrm{X}_{i,j}^{'} = \frac{1}{m^2} \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \mathrm{X}_{i \times m+a, j \times m+b}.$$

2.2 Homomorphic Encryption

Homomorphic encryption (HE) facilitates the execution of operations directly on encrypted data. Upon decryption of the resulting encrypted outputs, the outcomes are congruent with those obtained by performing the corresponding operations on plaintext values. This subsection provides a brief introduction to the CKKS scheme [5], a public-key cryptosystem based on the ring learning-with-errors problem [28]. Notably, the CKKS scheme allows for computations involving floating-point numbers, which has led to its extensive application in PPML scenarios [3, 23, 26, 32].

Encryption. With the lattice dimension N is a power of two and the ciphertext modulus Q, the encryption function, written as $Enc_{pk}^{N,Q}(\mathbf{m})$, uses the public key pk to encrypt an S-length plaintext vector $\mathbf{m} \in \mathbb{R}^S$, where $S = N/2$. The generated ciphertext is in the form of two ring elements so that $[a] = Enc_{pk}^{N,Q}(\mathbf{m}) \in R_{N,Q}^2$, where $R_{N,Q}$ is the cyclotomic ring over $\mathbb{Z}_Q/X^N + 1$.

Decryption. The decryption function, $Dec_{sk}^{N,Q}([a])$, uses the secret key sk to approximately recover the ciphertext $[a]$ to the plaintext vector $\mathbf{m}$, which means $Dec_{sk}^{N,Q}([a]) \approx \mathbf{m}$.

Batching. Because N is a power of 2, the ring $R_{N,Q}$ in the CKKS scheme is isomorphic to S copies of $\mathbb{Z}_Q$. Consequently, a ciphertext can accommodate values across up to S slots. The CKKS scheme enables operations to be conducted simultaneously on all S slots, a process referred to as batching. Specifically, a single HE operation executes identical computations across all S values contained within the ciphertext.

Amortized Time. The amortized time measures the runtime of HE operations per unit, defined as the total runtime divided by the number of units.

Homomorphic Operations. Assume $[a] = Enc_{pk}^{N,Q}(\mathbf{m})$ and $[b] = Enc_{pk}^{N,Q}(\mathbf{n})$, where $\mathbf{m}, \mathbf{n} \in \mathbb{R}^S$. We have the following two HE operations with batching capabilities.

- **HE Addition:** This operation simply slot-wise adds values. That is:
$$[a] \oplus [b] = Enc_{pk}^{N,Q}([\mathbf{m}_0 + \mathbf{n}_0, \mathbf{m}_1 + \mathbf{n}_1, \cdots, \mathbf{m}_{S-1} + \mathbf{n}_{S-1}]).$$
- **HE Multiplication:** HE multiplication can be performed on two ciphertexts or one ciphertext and one plaintext vector, and both cases return the same result. Therefore,
$$[a] \otimes [b] = [a] \otimes \mathbf{n} = Enc_{pk}^{N,Q}([\mathbf{m}_0 \times \mathbf{n}_0, \mathbf{m}_1 \times \mathbf{n}_1, \cdots, \mathbf{m}_{S-1} \times \mathbf{n}_{S-1}]).$$

Leveled Ciphertext Modulus. In the CKKS scheme, the ciphertext modulus Q is constructed from several co-primes q_i to enhance performance and improve noise management. Specifically, for an L-level Q_L, we have $Q_L = \prod_{i=1}^{L} q_i$. The ciphertext level indicates how many HE multiplications can be performed before the noise grows to an unacceptable level. When the level reaches zero, it becomes necessary to apply bootstrapping [4] to elevate the ciphertext back to a higher level, enabling further computations. However, it is essential to recognize that bootstrapping is computationally intensive and time-consuming.

Rotate Functions. The rotation operation performs a homomorphic and cyclic shift of the slots. Let us consider the ciphertext $[a]$, which encrypts a vector $\mathbf{m}$. The left rotation function, denoted as $RotL([a], i)$, shifts the $\mathbf{m}$ i positions to the left. In contrast, $RotR([x], i)$ facilitates the shifting to the right. For illustrative purposes, when applying the $RotL$ function, the resulting output $[a']$ can be expressed as follows:
$$[a'] = RotL([a], i) = Enc_{pk}^{N,Q}([\mathbf{m}_i, \mathbf{m}_{i+1}, \cdots, \mathbf{m}_{S-1}, \mathbf{m}_0, \cdots, \mathbf{m}_{i-1}]).$$

Rotation and Sum Functions. We further explain the rotate-and-sum-left function, $RSL([a], s, t)$. This function executes the $RotL$ function for t iterations, with the iterator commencing from zero. In the i-th iteration, the ciphertext $[a]$ performs an in-place addition with its own left-rotated version with $2^i \times s$ slots. Conversely, the rotation-and-sum-right function, RSR, operates analogously but utilizes the $RotR$ function instead.

2.3 HE-based Inference Workflow

As depicted in Figure 1, the workflow encompasses both the client and the server. The client transmits the encrypted data intended for inference to the server, which subsequently offers an inference service utilizing a pre-trained model. Throughout the inference process, the server handles only the encrypted data, resulting in its access being limited to the encrypted input and the corresponding encrypted prediction outcomes. The decryption of the encrypted results is exclusively possible by the client, who possesses the private key.

2.4 Thread Model

This paper follows previous HE-based PPML solutions [3, 6, 8, 10, 12, 16, 20, 21, 23–26], assuming that the server operates as a semi-honest party. After the user uploads encrypted data to the server, the server adheres to the established protocol and performs the private inference honestly. Although the server may attempt to analyze the intermediate data, it learns nothing because the data is encrypted. Only the client possessing the secret key can decrypt the inference results, thereby ensuring data privacy from the server.

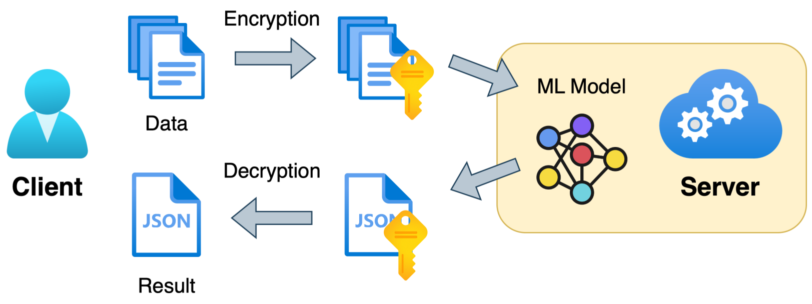


Fig. 1. An HE-based private inference workflow. Icons, along with a key, indicate the information is encrypted.

3 Related Work

This section provides a concise overview of existing inference solutions that employ pure HE, while addressing their inherent limitations.

3.1 Data Packing

In the early development of HE-based inference, CryptoNets [12], Faster CryptoNets [6], and CryptoDL [16] embedded each pixel of an image into distinct ciphertexts. Although this approach enabled batch inference by encrypting the same pixel positions from multiple inputs into the same ciphertext, it was constrained by long inference times. Gazelle [20] proposed a novel method that directly packed the entire input into a single ciphertext, significantly reducing end-to-end inference time. However, Gazelle [20] evaluated performance only on relatively small models. Following Gazelle [20], Lee et al. [26] further extended the packing method to support convolutions with a stride length of two and evaluated performance on ResNet-20. Nevertheless, the strided convolution proposed by Lee et al. [26] is inefficient due to its packing format and density loss.

After Lee et al. [26], MPCNN [23] introduced multiplexed packing-based convolution to address the inefficiency of strided convolution. Despite this, the data representation of MPCNN [23] sometimes requires rearrangement during the inference, resulting in additional overhead. To mitigate this, FHELIPE [21] proposed a novel data representation aimed at increasing packing density within ciphertexts and reducing the overhead of transforming between different representations. In contrast to MPCNN [23], which encrypts the entire input into a single ciphertext, FHELipe [21] embeds the input data on a channel-wise basis; therefore, the number of ciphertexts is proportional to the number of channels. Afterwards, ORION [10] proposed a single-shot multiplexed packing strategy that is not only suitable for arbitrary convolution settings but also reduces computational overhead, including halving the number of levels and ciphertext rotations. However, despite the advanced techniques provided by MPCNN [23], FHELIPE [21], and ORION [10], they do not utilize all of slots for encrypting data, and all of them support only single-input inference on a single thread, which limits their performance as batch size increases.

3.2 Polynomial Approximations for Non-linear Functions

To minimize the approximation error between polynomials and the original non-linearity, Lee et al. [24] approximate the $sign(x)$ using a composition of minimax-approximated polynomials, with ReLU derived through the calculation

$x \times (x + x \times sign(x))/2$. For high accuracy, polynomials of up to degree twenty-seven are employed. Subsequent studies have followed this approach [23, 25, 26].

[3, 8] focused on reducing the polynomial degree requirements through post-retraining techniques. The former study successfully reduced the necessary polynomial degree from twenty-seven to fourteen, while the latter employed mixed-degree polynomials to strike a balance between performance and accuracy. AESPA [32] utilized quadratic polynomials and retrained the models. However, a notable drawback of AESPA is its limited applicability to larger models.

3.3 Model Compression

To reduce computational overhead, many PPML approaches [6, 11, 19, 40] enhance performance by simplifying model structures. For example, [11, 19, 40] focus on pruning ReLU activation functions, which are major bottlenecks in PPML inference. In contrast, pruning model weights has not been extensively investigated in pure HE-based inference systems. The only notable work in this direction is Faster CryptoNets [6], which employs dynamic network surgery [13]. However, this method requires 39.1 seconds to process a single image using a 5-layer CNN, while recent studies [23, 26] have shown superior performance, particularly for larger model architectures.

On the other hand, He et al. [15] is one of the most representative studies on channel pruning in the ML community. However, compared with our proposed PCP method in §4.3, the main differences are as follows: (1) PCP enforces the number of remaining channels to be a power of two for ResNet and VGG models, ensuring compatibility with our ECSP packing and avoiding additional CKKS rotations during channel aggregation (in §4.1.2); (2) PCP pre-groups layers before pruning, effectively reducing the search space for suitable pruned architectures; and (3) PCP employs knowledge distillation to maintain model performance, whereas He et al. [15] train the pruned models directly.

4 LIME

LIME integrates element-wise channel-to-slot packing (ECSP) and power-of-two channel pruning (PCP) to reduce the resource required by pure HE-based inference. The former technique equips LIME with batch encryption capabilities, while the latter facilitates model compression. The ECSP packing technique leverages the available slots within a ciphertext to facilitate batch inference. At the same time, the PCP serves as a model compression strategy that reduces the number of channels within convolutional layers, thereby decreasing computational complexity.

To enhance performance with ECSP, we propose an evaluation block specifically designed for processing wide-channel convolutions using ECSP data. Furthermore, LIME replaces ReLU with a quadratic polynomial, incorporates the ReLU-before-addition (RBA) residual block, and employs knowledge distillation [17] to enhance approximation accuracy. The integration of these techniques facilitates efficient batch inference with minimal performance degradation.

The ECSP is detailed in §4.1, while the evaluation block for ECSP-based data is discussed in §4.2. The PCP is introduced in §4.3, the RBA residual block is described in §4.4, and the knowledge distillation is covered in §4.5.

4.1 Element-Wise Channel-to-Slot Packing (ECSP)

The element-wise channel-to-slot packing (ECSP) is an innovative technique that significantly enhances the performance of convolutions with batch samples (input data). It effectively utilizes the abundant slots within a ciphertext to store values. For any multi-channel data $\mathbf{X}$, ECSP packs it into a 2-D encrypted array $[\mathbf{X}]$ by element-wise encrypting each element along with its channel values into separate ciphertexts. In the CKKS scheme, there are S slots available. Since the value of S is usually large enough to meet security requirements, a single ciphertext can accommodate an element with a substantial number of channels or even pack the channels of multiple samples to facilitate batch inference.

ECSP provides two distinct packing styles to organize channel values: dense and sparse. A notable feature of this technique is the ability to interchange between the two styles after channel aggregation without requiring additional post-processing. This flexibility enhances the usability of ECSP across various CNN models. In the following sections, we will first introduce the two packing styles within ECSP, followed by the construction of the ECSP-based convolution and the ECSP-based activation function.

4.1.1 The Dense and Sparse Style Packing. As illustrated in Equation 1, the convolution operation results in the generation of n_c intermediate channels. Consequently, each sample element will be assigned n_c slots within each ciphertext of $[\mathbf{X}]$ across both packing styles. Provided by $\lceil S/n_c \rceil > 1$, it is feasible to perform a single convolution across several samples. The allocations of n_c slots for each sample are sequentially concatenated until the entirety of the S slots is fully occupied.

Dense Style Packing. All channels corresponding to an element of a sample are allocated within the initial c_i positions of the n_c slots. The remaining slots are populated with zeros. Therefore, the encrypted $[\mathbf{X}]$ can be represented as:

$$[\mathbf{X}_{i,j}] = Enc_{pk}^{N,Q}([\underbrace{\overbrace{\mathbf{X}_{i,j,0}, \cdots, \mathbf{X}_{i,j,c_i-1}}^{c_i \text{ slots}}, 0, \cdots,}_{n_c \text{ slots}} \cdots]) \forall i \in [0, W-1], j \in [0, H-1]. \quad (2)$$

Sparse Style Packing. This packing style arranges c_i input channels within every c_o slot. In other words,

$$[\mathbf{X}_{i,j}] = Enc_{pk}^{N,Q}([\underbrace{\overbrace{\mathbf{X}_{i,j,0}, 0, \cdots,}^{c_o \text{ slots}} \overbrace{\mathbf{X}_{i,j,1}, 0, \cdots,}^{c_o \text{ slots}} \cdots, \overbrace{\mathbf{X}_{i,j,c_i-1}, 0, \cdots,}^{c_o \text{ slots}}}_{n_c \text{ slots}} \cdots]) \forall i \in [0, W-1], j \in [0, H-1]. \quad (3)$$

4.1.2 ECSP-based Convolution. Four primary steps are involved in conducting convolution on data organized in the ECSP format: data duplication, kernel preparation, 2-D HE convolution, and channel aggregation. It is important to note that, except for the 2-D HE convolution, each preceding step requires distinct configurations that vary according to the specific packing styles employed.

Data Duplication. This step organizes the encrypted $[\mathbf{X}]$ ready for convolution. Specifically, all c_i channels associated with each sample within the ciphertext are duplicated c_o times. Due to the CKKS batching, the subsequent function will be executed only once and will have a uniform impact all samples.

- **Dense Style Data:** The *RSR* function is applied with parameters $(s, t) = (c_i, log_2 c_o)$.
- **Sparse Style Data:** The *RSR* function is applied with parameters $(s, t) = (1, log_2 c_o)$.

Kernel Preparation. In this step, a 4-D kernel $\mathbf{K} \in \mathbb{R}^{r \times r \times c_o \times c_i}$ is organized into a 2-D array format $\hat{\mathbf{K}} \in \mathbb{R}^{r \times r \times S}$. The channel values of each kernel element occupy n_c slots, with the arrangement of these channel values determined by the data with which they will be convolved. When dealing with multiple samples, the kernel values are duplicated, just as multiple samples are packed.

- **Dense Style Data:** The feature map resulting from the kernel convolving with the i-th channel of the input to produce the j-th output channel is placed in the $(j \times c_i + i)$-th slot.

- **Sparse Style Data:** The kernel value that operates on the i-th input channel to generate the j-th output channel is located in the $(i \times c_o + j)$-th slot.

2-D HE Convolution. Regardless of the packing style employed, the 2-D HE convolution between $[\mathbf{X}]$ and $\hat{\mathbf{K}}$ follows the same processing approach. Because of the CKKS batching, a total of n_c convolutions can be performed simultaneously. We can apply the direct convolution algorithm to obtain the convoluted result, which is mathematically represented as:

$$[\mathrm{Y}_{i,j}] = \sum_{a,b=0}^{r-1} [\mathrm{X}_{i\times l+a, j\times l+b}] \otimes \hat{\mathrm{K}}_{a,b}. \tag{4}$$

Channel Aggregation. In this step, we collect n_c intermediate channel values to generate the final convolution results, resulting in c_o output values.

- **Dense Style Data:** The *RSL* function is applied with parameters $(s,t) = (1, log_2 c_i)$. The output values are then stored in the first slot of all c_o groups. In this case, the outputs are organized in a sparse representation.
- **Sparse Style Data:** The *RSL* function is again, and this time with parameters $(s,t) = (c_o, log_2 c_i)$. The resulting output values are allocated to the first c_o slots, arranged in a dense style representation.

Following channel aggregation, two styles are exchanged, resulting in no additional costs associated with reorganizing the slots. However, it is essential to note that only the slots containing the correct c_o output values hold significance for further computations. Therefore, after channel aggregation, a filtering process is employed to retain the valid values while discarding the irrelevant ones.

Since the ECSP packs the plaintext input element-wise, the computational and memory complexity of ECSP-based convolution scales proportionally with the input resolution. In conclusion, Figure 2 illustrates an example of convolution based on the ECSP for both data packing styles. Algorithm 1 outlines the complete workflow of the ECSP-based convolution algorithm.

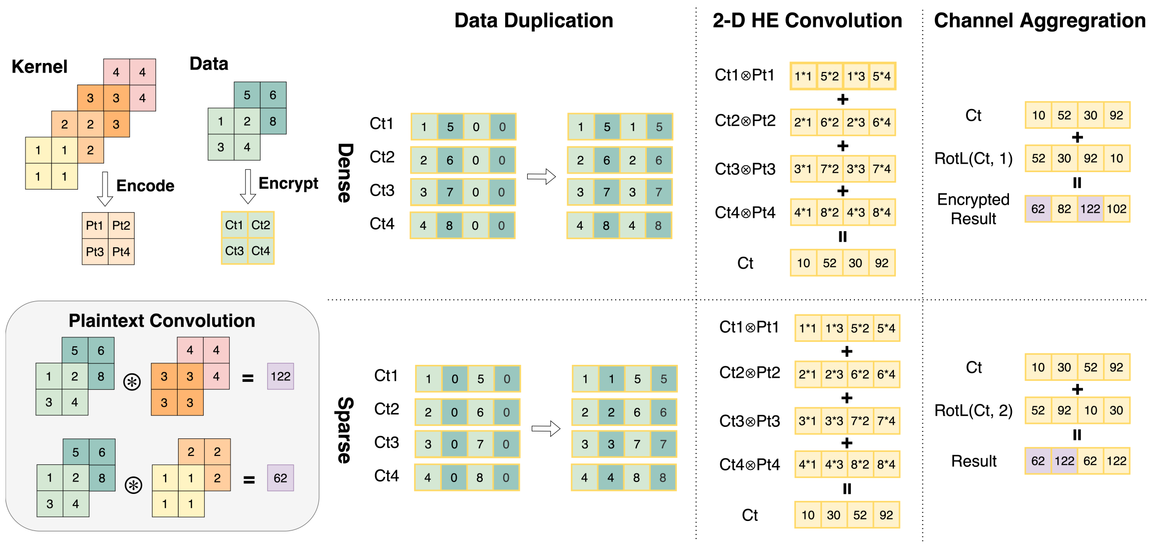


Fig. 2. An example of the ECSP-based convolution with $c_i = 2$ and $c_o = 2$ for both style data.

Algorithm 1: The ECSP-based Convolution Algorithm. (ECSP_CONV)

```
Input: An encrypted $W \times H$ $type$-style encrypted input $[\mathbf{X}]$ and the $r \times r \times S$ kernel $\hat{\mathbf{K}}$. The stride length is $l$.
Output: An encrypted $W' \times H'$ convolution result $[\mathrm{Y}]$.
// Data Duplication
1  for $i \leftarrow 0$ to $W-1$ do
2      for $j \leftarrow 0$ to $H-1$ do
3          if $type$ == "$dense$" then $s \leftarrow c_i$ ;
4          else $s \leftarrow 1$ ;
5          $[\mathrm{X}_{i,j}] \leftarrow RSR([\mathrm{X}_{i,j}], s, log_2 c_o)$ ;
// 2-D HE Convolution
6  for $i \leftarrow 0$ to $W'-1$ do
7      for $j \leftarrow 0$ to $H'-1$ do
8          for $a \leftarrow 0$ to $r-1$ do
9              for $b \leftarrow 0$ to $r-1$ do
10                 $[\mathrm{Y}_{i,j}] \leftarrow [\mathrm{Y}_{i,j}] \oplus [\mathrm{X}_{i\times l+a, j\times l+b}] \otimes \hat{\mathbf{K}}_{a,b}$
// Channel Aggregation
11 for $i \leftarrow 0$ to $W'-1$ do
12     for $j \leftarrow 0$ to $H'-1$ do
13         if $type$ == "$dense$" then $s \leftarrow 1$ ;
14         else $s \leftarrow c_o$ ;
15         $[\mathrm{Y}_{i,j}] \leftarrow RSL([\mathrm{Y}_{i,j}], s, log_2 c_i)$;
16 return $[\mathrm{Y}]$;
```

4.1.3 ECSP-based Activation Functions with Polynomial Approximations. To minimize the computational overhead associated with activation functions, we implement the AESPA [32], which replaces the ReLU with a quadratic polynomial. Specifically, AESPA substitutes the ReLU function with a three-term Hermite expansion and integrates basis-wise normalization concatenated to each basis to mitigate the issue of gradient explosion. Consequently, the proposed activation function is defined as follows:

$$f(x) = \gamma \sum_{i=0}^{d=2} \hat{f}_i \frac{h_i(x) - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} + \beta, \tag{5}$$

where $\hat{f}_i$ is the dot product of $ReLU(x)$ and the i-th base Hermite polynomial, so

$$\hat{f}_0 = \frac{1}{\sqrt{2\pi}}, \hat{f}_1 = \frac{1}{2}, \text{ and } \hat{f}_2 = \frac{1}{\sqrt{4\pi}}.$$

The function $f(x)$ can be expressed as a second-degree polynomial, wherein the leading coefficient of each function can be amalgamated with the model weights. Thus, this activation function incurs only one multiplicative level. It is also important to emphasize that the evaluation of the activation function is performed element-wise, ensuring that the order of slots within a ciphertext remains unchanged after the activation function is applied.

4.2 ECSP-based Evaluation Block

To apply the ECSP data for CNN inference, we introduce the evaluation block, a structural component comprising two consecutive convolutional operations. The input to the evaluation block adopts a more compact dense style format known as the widest channel-aware dense (wc-dense) style, which is designed to optimize packing efficiency during batch inference. Within the wc-dense format, each sample is allocated only c_w slots, representing the maximum number of input channels permissible across the convolutional layers of the overall model architecture. Consequently, a single ciphertext can accommodate channels corresponding to up to $B = S/c_w$ samples at most.

Given that the input to an evaluation block adheres to a dense format, the input to the first convolutional layer is consequently in the dense style, and the input to the second convolutional layer is transitioned to a sparse format. Ultimately, data representation reverts to the dense style after the evaluation block. We denote the input channel of an evaluation block as c_i, whereas the output channels of the two convolutional kernels, $\hat{\mathbf{K}}^1$ and $\hat{\mathbf{K}}^2$, are c_{o_1} and c_{o_2}, respectively. Consequently, the number of slots for each sample is adjusted to $n_c = max(c_i c_{o_1}, c_{o_1} c_{o_2})$.

Each sample necessitates n_c slots due to data duplication. If $c_w > n_c$, the wc-dense data can be interpreted as conforming to the standard dense style for convolution operations. However, for in stances where $S \geq n_c \geq c_w$, duplicated slots between samples may overlap, leading to incorrect results. In more critical scenarios, when $n_c > S$, the total number of slots available in a ciphertext becomes inadequate for accommodating a single sample. Hence, the number of channels n_c required for convolution substantially influences the efficiency of the evaluation block.

To address these challenges, we partition the channels into multiple subsets, process them sequentially in rounds, and then merge the results. We propose introducing both an extraction step and a merge step positioned before and after the evaluation block. The extraction step selectively filters a subset of the total B samples for processing within the evaluation block. In contrast, the merge step consolidates the results from all subsets back into the channel-aware dense style format. In the subsequent sections of this write-up, we will analyze the functional dynamics of the extraction and merge steps, particularly the interplay between n_c and S.

4.2.1 Extraction. The extraction step is employed to sample a subset of values from the ciphertext for evaluation purposes. An HE multiplication facilitates this process with a filter $\mathbf{f}$, in which the positions designated for sampling are assigned a value of one. At the same time, all other slots are set to zero.

Case 1: $S \geq n_c \geq c_w$. The extraction step selects c_w values for every n_c slot, allowing for the selection of multiple samples. These selected samples are grouped into a subset, resulting in a total number of subsets given by $n_s = n_c/c_w$. Each subset is processed independently by the evaluation block for two consecutive convolutions. Given that $n_c \geq c_{o_1} c_{o_2}$, each subset can produce accurate results under the dense style.

Case 2: $n_c > S$. In this case, even all the S slots of the ciphertext are insufficient to provide a sample for convolution, resulting in only a single sample being available in a subset. Moreover, a single convolution pass will not yield the correct output. Consequently, the convolutions within the evaluation block must be divided into $n_r = n_c/S$ rounds, allowing for the processing of a limited number of channels at a time.

4.2.2 Merge.

Case 1: $S \geq n_c \geq c_w$. After each subset has passed through the evaluation block, the values will be in a dense format. Since each sample is allocated c_w slots in the wc-dense format, there will be no overlap of sample values throughout the entire CNN inference. To consolidate all subsets back into the wc-dense format, we can simply aggregate the subsets by summing the ciphertexts element-wise.

Case 2: $n_c > S$. In this scenario, by aggregating the results of each round element-wise, we can derive the output values for c_{o_2} output channels. Following this, as described in the first case, the aggregation process can be applied to compile all rounds back into the wc-dense format.

To summarize, Table 2 presents the required number of subsets (n_s) and rounds (n_r). Note that the two kernels are divided into n_r sub-kernels, denoted as $(\hat{\mathbf{K}}^{1(0)}, \hat{\mathbf{K}}^{1(1)}, \cdots, \hat{\mathbf{K}}^{1(n_r-1)})$ and $(\hat{\mathbf{K}}^{2(0)}, \hat{\mathbf{K}}^{2(1)}, \cdots, \hat{\mathbf{K}}^{2(n_r-1)})$. Each sub-kernel is processed in a separate round, and its elements are selected according to the corresponding input indices involved in the convolution. Figure 3 illustrates an example of the workflow of the evaluation block, and Algorithm 2 outlines the algorithm employed within the evaluation block.

Table 2. Summary of Subsets and Rounds Based on S, n_c, and c_w.

Case	$\#n_s$	$\#n_r$
$n_c > S$	B	n_c/S
$S \geq n_c \geq c_w$	n_c/c_w	1
$c_w > n_c$	1	1

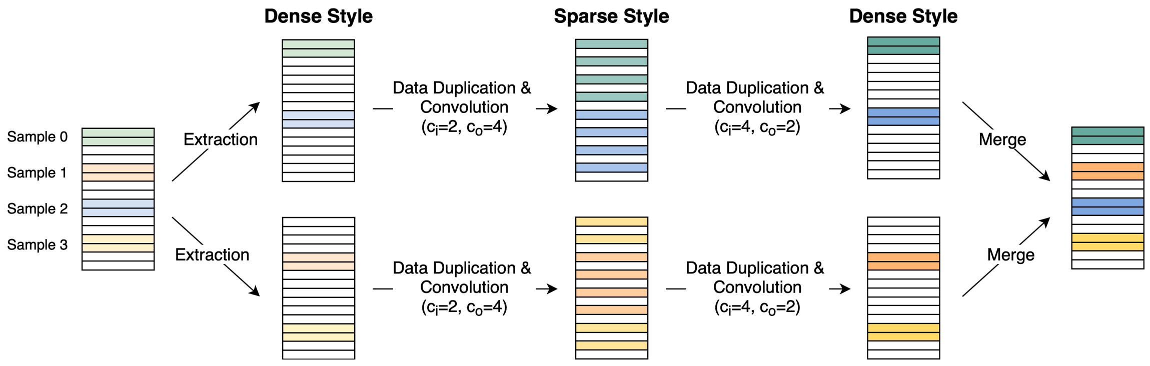

Dense Style
Sparse Style
Dense Style
Sample 0
Sample 1
Sample 2
Sample 3
Extraction
Extraction
Data Duplication & Convolution (c_i=2, c_o=4)
Data Duplication & Convolution (c_i=4, c_o=2)
Data Duplication & Convolution (c_i=2, c_o=4)
Data Duplication & Convolution (c_i=4, c_o=2)
Merge
Merge

Fig. 3. The example workflow of the evaluation block.

4.3 Power-of-Two Channel Pruning (PCP)

The necessity of n_c slots for convolution per sample limits the feasibility of processing multiple samples concurrently, particularly when utilizing convolutions with large channel sizes, as the values of n_s and n_r increase. Consequently, convolutions with wide channels impose significant burdens on computational performance.

To mitigate this challenge, we introduce the power-of-two channel pruning (PCP) technique, a model compression approach that reduces the number of channels in convolutional layers. This reduction effectively alleviates the demands associated with n_s and n_r. Furthermore, as the maximum number of channels c_w decreases, the maximum batch size feasible under the ECSP packing scheme can be increased. Thus, PCP not only reduces computational overhead and compresses the model but also enhances the efficiency of batch inference.

Algorithm 2: The workflow of an evaluation block algorithm.

```
Input: An encrypted $W \times H$ dense-style encrypted input $[\mathbf{X}]$ and two sets of the sub-kernels of $\hat{\mathbf{K}}^1$ and $\hat{\mathbf{K}}^2$. The
       number of slots is $S$, and the slots reserved for each sample are $n_c$. The number of widest channel
       throughout the model is $c_w$. The number of subsets and rounds for the evaluation block is $n_s$ and $n_r$.
Output: An encrypted $W' \times H'$ convolution result $[\mathbf{Y}]$.
// Evaluate the total batch samples into $n_s$ subsets
1  for $s \leftarrow 0$ to $n_s - 1$ do
       // Filter Definition
2      for $i \leftarrow s \times c_w$ to $S - 1$ do
3          $\mathbf{f}_i \leftarrow 1$ if $((i - s \times c_w) \mod n_c) < c_w$ else $0$;
       // Extraction
4      for $i \leftarrow 0$ to $W - 1$ do
5          for $j \leftarrow 0$ to $H - 1$ do
6              $[\mathbf{X'}_{i,j}] \leftarrow [\mathbf{X}_{i,j}] \otimes \mathbf{f}$ ;
       // Produce $c_{o_2}$ output channels over $n_r$ rounds
7      for $r \leftarrow 0$ to $n_r - 1$ do
           // ECSP-based Convolution
8          $[\mathbf{X}''] \leftarrow ECSP\_CONV([\mathbf{X}'], \hat{\mathbf{K}}^{1(r)}, dense)$ ;
9          $[\mathbf{X}'''] \leftarrow ECSP\_CONV([\mathbf{X}''], \hat{\mathbf{K}}^{2(r)}, sparse)$ ;
           // Merge
10         for $i \leftarrow 0$ to $W' - 1$ do
11             for $j \leftarrow 0$ to $H' - 1$ do
12                 $[\mathbf{Y}_{i,j}] \leftarrow [\mathbf{Y}_{i,j}] \oplus [\mathbf{X'''}_{i,j}]$ ;
13 return $[\mathbf{Y}]$;
```

Because channel aggregation in ECSP-based convolutions moves slots in powers of two, the most effective way to reduce n_s and n_r is to prune channels by powers of two. For instance, with an initial count of 64 channels in a convolutional layer, applying half-channel pruning would reduce the number of channels to 32 or fewer.

Since contemporary models are primarily developed in a deep architecture, the number of pruned model configurations can increase significantly if each layer is pruned independently. To constrain the search space for pruned models, PCP groups several consecutive layers and applies a uniform pruning ratio across the layers within the same group, as illustrated in Figure 4.

Given an M-layer CNN model, the layers of the model are denoted as $\mathbf{c} = [0, \mathbf{c}_1, \cdots, \mathbf{c}_M]$, where $\mathbf{c}_i$ represents the i-th layer, and layer index counts start from one. PCP segments several consecutive layers into a total of T groups. The j-th group, denoted as $\mathcal{G}_j$, consists of b_j consecutive layers and is defined as follows:

$$\mathcal{G}_j = \{i | i \in [a_j, a_j + b_j - 1]\},$$

where $a_1 = 1$, $a_j = a_{j-1} + b_{j-1}$, and $a_T \leq M$.

For the group j, a pruning ratio $\mathbf{r}_j = 2^{-k_j}$ is assigned, where $k_j \in \mathbb{Z}_{\geq 0}$. As all layers within the same group share the same pruning ratio, the result of pruning for layers in group j is described by the following equation:

$$\mathbf{c}_i^{'} = \mathbf{c}_i \mathbf{r}_j \ \forall \mathbf{c}_i \in \mathcal{G}_j,$$

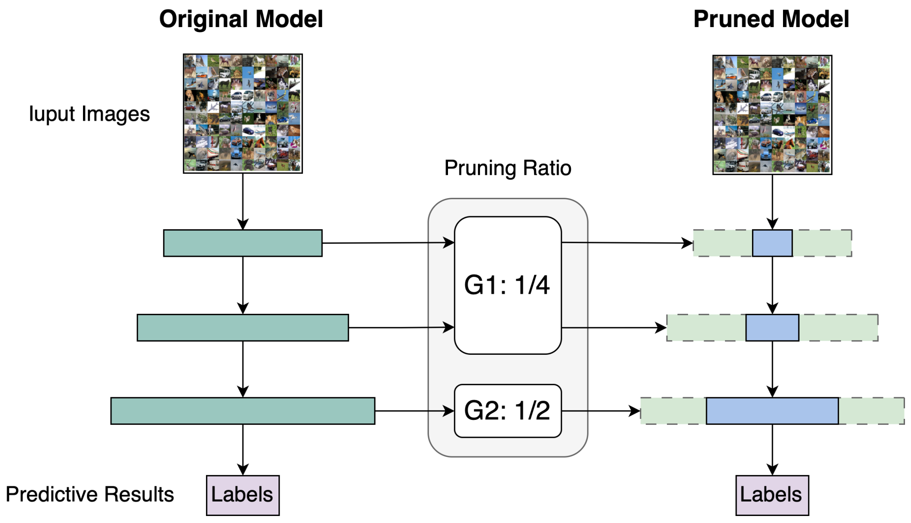


Fig. 4. A demonstration of PCP.

and all $\mathbf{r}_j$ values are set to one, indicating that no pruning occurs for any of the groups initially.

A function designated as *CNP* is employed to calculate the total number of parameters in the model $\mathbf{c}$ based on the specified pruning ratio $\mathbf{r}$:

$$CNP(\mathbf{c}, \mathbf{r}) = \sum_{i=1}^{M-1} \left(\mathbf{c}_i \mathbf{r}_{GI(i)}\right) \times \left(\mathbf{c}_{i+1} \mathbf{r}_{GI(i+1)}\right), \tag{6}$$

where $GI(i)$ returns the group index of the i-th layer:

$$GI(i) = j \text{ if } i \in \mathcal{G}_j.$$

PCP is governed by two parameters: the minimum required batch size, denoted as β, and the maximum request compression ratio, represented by τ, defined as the ratio of the reduced number of parameters to that of the original model. Before the search algorithm begins, PCP calculates the new widest channel among the CNN model, $c'_w = S/\beta$, subsequently updating the pruning ratio $\mathbf{r}$ according to the following conditions:

$$\mathbf{r}_{GI(i)} = \begin{cases} c'_w / c_w & \text{if } \mathbf{c}_i = c_w \\ 1 & \text{otherwise} \end{cases}.$$

PCP explores for all qualified models $\{\mathbf{c}', \mathbf{r}'\}$ such that the compression ratio C_r is smaller than τ. Therefore,

$$C_r = 1 - \frac{CNP(\mathbf{c}', \mathbf{r}')}{CNP(\mathbf{c}, \mathbf{r})} < \tau. \tag{7}$$

Moreover, PCP guarantees that the structure of the pruned model aligns with that of the original model, specifically by preserving the shape order of layers. The requirement can be expressed as:

$$\begin{cases} \mathbf{c}_i^{'} \leq \mathbf{c}_{i+1}^{'} \text{ if } \mathbf{c}_i \leq \mathbf{c}_{i+1} \\ \mathbf{c}_i^{'} > \mathbf{c}_{i+1}^{'} \text{ if } \mathbf{c}_i > \mathbf{c}_{i+1} \end{cases}.$$

To conclude, Algorithm 3 summarizes the PCP algorithm, which depends on β and τ.

Algorithm 3: The PCP Algorithm.

```
Input: An $M$-layer model $\mathbf{c}$, the batch size requirement $\beta$, and the pruning threshold $\tau$. The group index
       mapping function $GI$ divides $M$ layers into $T$ groups, and the number of parameters calculator $CNP$. The
       number of slots is $S$ and the widest channel in the original model is $c_w$.
Output: All qualified models $\mathcal{L}$.
1  $\mathbf{r} \leftarrow [1$ for $i$ in range$(1, T+1)]$ ;
2  for $i \leftarrow 1$ to $M$ do
3      if $\mathbf{c}_i$ is $c_w$ then
4          $\mathbf{r}_{GI(i)} \leftarrow S/(\beta \times c_w)$ ; // Prune the widest channel by $c'_w/c_w$
5  $\mathcal{L} \leftarrow \{[\mathbf{c}_i \mathbf{r}_{GI(i)}$ for $i$ in range$(1, M+1)]\}$ ; // Save the qualified compressed models
6  $Q \leftarrow deque(\mathbf{r})$ ;
7  while $Q$ is not empty do
8      $\mathbf{a} \leftarrow Q.popleft()$ ;
9      for $t \leftarrow T$ to $1$ do
10         $\mathbf{s}_t \leftarrow \mathbf{a}_t$ // 2 ;
           // Verify whether the pruned layers maintain their order-preserving properties
11         if sum([$\mathbf{c}_i \geq \mathbf{c}_{i-1}$ and $\mathbf{c}_i \mathbf{s}_{GI(i)} \geq \mathbf{c}_{i-1} \mathbf{s}_{GI(i-1)}$ for $i$ in range$(1, M+1)$]) is not $M-1$ then
12             continue ;
13         if $CNP(\mathbf{c}, \mathbf{s})/CNP(\mathbf{c}, \mathbf{r}) < \tau$ and $\mathbf{s}$ not in $\mathcal{L}$ then
14             $\mathcal{L} \leftarrow \{[\mathbf{c}_i \mathbf{s}_{GI(i)}$ for $i$ in range$(1, M+1)]\}$ ; // Save the qualified compressed model
15             $Q.append(\mathbf{s})$ ;
16 return $\mathcal{L}$
```

4.4 RBA-style Residual Block

While AESPA [32] has demonstrated the potential of substituting the ReLU activation function with a quadratic polynomial, the research conducted by [3] has identified several limitations inherent to AESPA [32]. These limitations include challenges related to convergence and a deterioration in accuracy performance when applied to larger models. To address these challenges, we propose implementing the ReLU-before-addition (RBA) residual block, which aims to mitigate the issues associated with AESPA [32]. Detailed comparisons of the residual blocks are presented in Figure 5.

The RBA block is initially introduced in the foundational ResNet paper [14]. It is defined by the repositioning of the ReLU within a residual block, placing it before the addition with the shortcut connection. Our empirical analysis reveals that it significantly addresses the challenges identified in AESPA [32], ultimately enhancing accuracy. Notably, the RBA block maintains computational efficiency comparable to the original evaluation block used in AESPA [32], as it only rearranges the order of computations within the residual block without introducing any additional operations. Last but not least, the RBA block requires only one bootstrapping operation, the same as AESPA [32].

4.5 Knowledge Distillation

To achieve optimal accuracy, we employed knowledge distillation [17] to train models incorporating the RBA block. In this setup, the student model serves as the target network architecture, which includes the RBA block. In contrast, the teacher model has the same architecture but uses the original residual block with ReLU activation functions. If PCP prunes the student model, the teacher model is pruned accordingly.

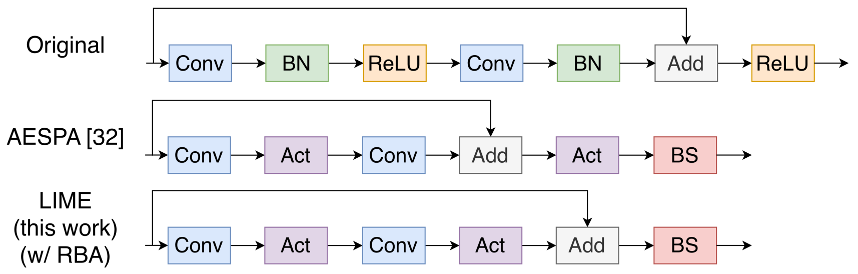


Fig. 5. Residual block with bootstrapping for HE-based private inference. “Conv”, “BN”, and “Act” denote the convolution, batch normalization, and the activation function, respectively, while “BS” stands for bootstrapping.

Knowledge distillation involves three additional parameters: (1) the temperature parameter, which facilitates knowledge transfer from the teacher to the student model; (2) the distillation loss, defined as the difference between the outputs (or logits) of the teacher and student models; and (3) the balancing parameter, which regulates the relative importance of the distillation loss and the standard cross-entropy loss during training.

5 Evaluations

Cryptosystem Configuration. To assess the LIME’s performance within the ciphertext domain, we implement private inference utilizing the HEaaN library, which offers high-performance computations based on the CKKS scheme. We have chosen to use the FGb parameter configuration, which ensures a security level of 128 bits. The parameters are defined with a ciphertext modulus of $N = 65536$ and a polynomial degree of $S = 32768$. Under these parameter settings, nine computational levels are available for utilization.

Datasets. We evaluate the performance of our proposed methods using the CIFAR-10, CIFAR-100, and Tiny-ImageNet datasets. CIFAR-10/100 datasets contain 50,000 training and 10,000 testing samples, each comprising a 32×32 RGB image. The primary difference between the two datasets lies in their classification schemes: the CIFAR-10 dataset categorizes samples into ten distinct classes, whereas the CIFAR-100 dataset encompasses 100 predictive classes. The Tiny-ImageNet dataset is more complex, consisting of 100,000 training samples and 10,000 testing samples, each represented as a 64×64 RGB image, comprising 200 classes.

Models. We implement ResNet-20 for CIFAR-10, VGG-11 for CIFAR-100, and ResNet-18 for Tiny-ImageNet. Table 3 presents the structural configurations of these models. Given the maximum channel widths c_w of width 64, 512, and 512 respectively, the corresponding batch sizes B for these architectures are 512, 64, and 64 samples, respectively.

PCP Settings. The layers classified under the same “group-i” in Table 3 are aggregated into a single pruning group. To maximize the potential of all pruned models, we set the batch size requirement β equal to the B of the original model. Regarding the maximum pruning rate, τ, is set at 90%.

Training Parameters. The training batch size is set to 128, with a learning rate of 0.1. ResNet-20 and VGG-11 models were trained for 200 epochs, whereas the ResNet-18 model was trained for 40 epochs using pretrained weights from the ImageNet dataset. Cross-entropy loss is employed for model training, while the distillation loss is measured by mean squared error. Both losses are equally weighted during training. The temperature parameter for knowledge distillation is fixed at one.

Table 3. Architectures of ResNet-20, VGG-11, and ResNet-18 models. "C-[number]" denotes a convolutional layer with the specified output channels. Each pair of convolutional layers forms an evaluation block, corresponding to a residual block in ResNet architectures. The number of repetitions of the residual block is indicated in the layer description column. Evaluation blocks with identical dimensions are grouped for pruning in PCP. A ReLU activation follows each convolution.

Model	Group Name	Input Size	Layer Description	$\#n_s$	$\#n_r$
ResNet-20	conv-1	$32 \times 32 \times 3$	C-16	1	1
	group-1	$32 \times 32 \times 16$	(C-16, C-16)×3	4	1
	group-2	$32 \times 32 \times 16$	(C-32, C-32)×3	16	1
	group-3	$16 \times 16 \times 32$	(C-64, C-64)×3	64	1
VGG-11	conv-1	$32 \times 32 \times 3$	C-64	1	1
	conv-2	$16 \times 16 \times 64$	C-128	1	1
	group-1	$8 \times 8 \times 128$	(C-256, C-256)	64	2
	group-2	$4 \times 4 \times 256$	(C-512, C-512)	64	8
	group-3	$2 \times 2 \times 512$	(C-512, C-512)	64	8
ResNet-18	conv-1	$64 \times 64 \times 3$	C-64	1	1
	group-1	$32 \times 32 \times 64$	(C-64, C-64)×2	64	2
	group-2	$32 \times 32 \times 128$	(C-128, C-128)×2	64	8
	group-3	$16 \times 16 \times 256$	(C-256, C-256)×2	64	8
	group-4	$8 \times 8 \times 512$	(C-512, C-512)×2	64	8

Experimental Setup. Evaluations of private computations were performed on a system equipped with an AMD 7975WX CPU and 512 GB of RAM. Model training utilized an Nvidia GTX 4090 GPU. All programs were executed in parallel using OpenMP [31].

Baselines. We select Lee et al. [24] and AESPA [32] as baselines to evaluate the accuracy of applying the RBA block and knowledge distillation. Specifically, Lee et al. [24] approximate teacher models with a security parameter of 13, using three polynomials of degrees 15, 15, and 27 to approximate the ReLU activation. To assess the ECSP packing performance, we compare our method with MPCNN [23], FHELIPE [21], and ORION [10]. For end-to-end evaluation of LIME, comparisons are made with the SoTA ORION [10], combined with Lee et al. [24] and with AESPA [32].

5.1 Evaluations of the RBA Block and the ECSP Packing

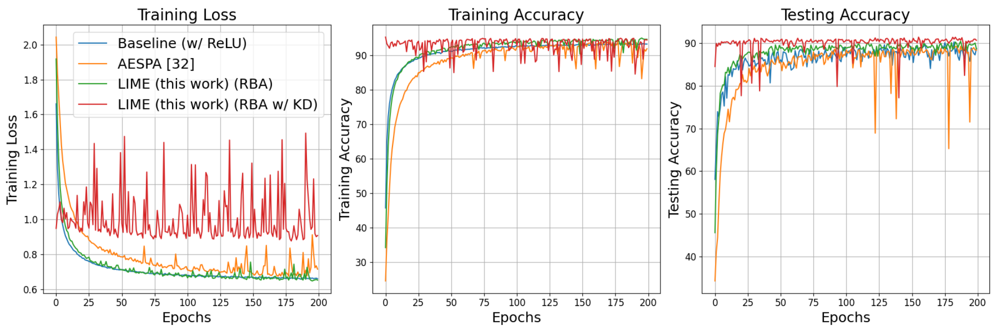
We employ the original, uncompressed models to evaluate the accuracy of using the RBA block and to measure the runtime with ECSP packing.

5.1.1 Accuracy. As shown in Table 4, Lee et al. [24] employ high-degree polynomials, resulting in a slight degradation in accuracy compared to the ReLU activation. AESPA [32] exhibits an accuracy decline of 3% to 8% across all test scenarios. With the RBA block, the performance of LIME on ResNet models is comparable to that of Lee et al. [24]. Note that since VGG-11 lacks residual blocks, its accuracy is the same as that of AESPA [32]. After applying knowledge distillation (KD), LIME achieves the best results among all other methods.

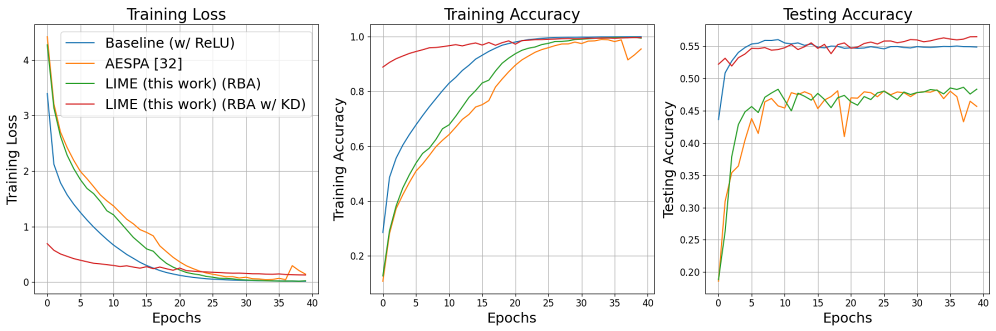
Figure 6 shows that the training loss curve of the RBA block exhibits greater stability than that of AESPA [32] and closely matches the original model employing ReLU. Consequently, for both ResNet-20 and ResNet-18, LIME with the RBA block outperforms AESPA [32] in terms of training and testing accuracy. When KD is applied, the final accuracy surpasses that of the baseline method using the ReLU activation.

Table 4. Accuracy of different methods. Best private accuracy values are in bold.

Method	**ResNet-20-CIFAR-10**	**VGG-11-CIFAR-100**	**ResNet-18-Tiny-ImageNet**
Baseline (w/ ReLU)	90.2%	65.6%	56.0%
Lee et al. [24]	90.1%	65.4%	55.8%
AESPA [32]	87.1%	59.1%	48.3%
LIME (this work) (RBA)	90.6%	59.1%	48.7%
LIME (this work) (RBA+KD)	**91.6%**	**67.5%**	**56.5%**



(a)



(b)

Fig. 6. Training Curves of (a) ResNet-20 on CIFAR-10 and (b) ResNet-18 on Tiny-ImageNet.

5.1.2 Microbenchmarks of ECSP Packing. Table 5 demonstrates that the ECSP method substantially outperforms all baselines across configurations. While runtime generally increases with larger input and output channels of all solutions, ECSP's runtime remains relatively stable. Notably, ECSP is especially effective for smaller channel combinations. For an input size of 32×32 with $(c_i, c_o) = (3, 16)$, ECSP achieves up to 1350× speedup over MPCNN [23]. Relative to ORION [10], ECSP attains a 42× speedup under the same setting and maintains a 4.4× even for $(c_i, c_o) = (512, 512)$. Furthermore, ECSP continues to exhibit substantial gains with larger inputs; for a 64×64 input and $(c_i, c_o) = (3, 64)$, it achieves a 71× speedup over ORION [10].

Previous approaches, including MPCNN [23], FHELIPE [21], and ORION [10], process a single input on a single thread, limiting runtime optimization due to the absence of effective parallelism. In contrast, when $n_c < S$, ECSP packs multiple samples for parallel execution, thereby achieving improved amortized time. A smaller n_c allows ECSP to pack more samples, as each convolution can accommodate S/n_c samples. Moreover, since the channel counts and batch sizes in above settings are powers of two, ECSP fully utilizes available slots during packing, further enhancing runtime efficiency. This explains ECSP's superior performance when input and output channels are smaller.

Table 5. Single-sample runtime (s) for a convolution operation.

Input Size	Input Channel	Output Channel	MPCNN [23]	FHELIPE [21]	ORION [10]	LIME (this work)
64×64	3	64	149.76	137.88	9.92	**0.14**
32×32	3	16	13.49	12.42	0.42	**0.01**
32×32	16	16	23.14	21.31	0.39	**0.05**
16×16	32	32	26.41	23.73	0.78	**0.06**
8×8	64	64	33.36	27.34	1.21	**0.08**
8×8	256	256	565.28	463.27	5.93	**1.51**
4×4	512	512	888.24	727.94	10.37	**2.37**

5.2 Study of PCP Compressed Models

As shown in Table 6 and Table 7, with a setting of $\tau = 0.9$, twelve pruned ResNet models and nine pruned VGG models satisfy the PCP requirements. Moreover, the batch size, B, increases by a factor of 4 for all three models. Therefore, ResNet-20 can accommodate up to 2048 samples, and both VGG-11 and ResNet-18 can handle up to 256 samples.

Although the accuracy decay is inevitable with model compression, we can observe that the VGG model is more resilience to accuracy degradation than ResNet models. When 90% of parameters are pruned, ResNet-20 and ResNet-18 experience 7% and 11.6% of accuracy loss, respectively, while VGG-11 has only 3% degradation.

Table 6. The compression results for ResNet-20 on CIFAR-10 and ResNet-18 on Tiny-ImageNet. Model names indicate the number of pruned channels per group. According to Table 3, the output channels of the three pruning groups in ResNet-20 are 16, 32, and 64; thus, the original ResNet-20 is denoted as "16-32-64." Consequently, "8-16-64" indicates that the channels in the first and second groups are pruned by a factor of 2, while the third group remains unpruned.

ResNet-20-CIFAR-10					ResNet-18-Tiny-ImageNet				
Model Name	Batch Size B	#Params	Compression Ratio C_r	Accuracy	Model Name	Batch Size B	#Params	Compression Ratio C_r	Accuracy
16-32-64	512	271K	0%	91.6%	64-128-256-512	64	11.3M	0%	56.5%
8-32-64	512	258K	4%	91.2%	64-64-256-512	64	10.7M	5%	53.0%
16-16-64	512	223K	16%	90.8%	64-128-128-512	64	9.1M	20%	54.0%
8-16-64	512	212K	19%	90.3%	64-64-128-512	64	8.7M	23%	50.0%
8-8-64	512	197K	24%	89.3%	64-64-64-512	64	7.9M	30%	48.4%
16-32-32	1024	122K	56%	90.6%	64-128-256-256	128	5.3M	53%	54.6%
8-32-32	1024	109K	60%	89.2%	64-64-256-256	128	4.7M	58%	49.1%
16-16-32	1024	80K	71%	89.0%	64-128-128-256	128	3.4M	69%	51.4%
8-16-32	1024	68K	75%	88.2%	64-64-128-256	128	3.0M	73%	47.0%
8-8-32	1024	56K	79%	86.6%	64-64-64-256	128	2.4M	78%	47.3%
16-16-16	2048	42K	85%	86.5%	64-128-128-128	256	1.9M	83%	49.0%
8-16-16	2048	30K	89%	84.7%	64-64-128-128	256	1.5M	87%	44.9%

Table 7. The pruning results for the VGG-11 on the CIFAR-100. The model names follow the same naming convention as described in Table 6.

Model Name	**Batch Size** B	**#Parameters**	**Compression Ratio** $\mathbf{C_r}$	**Accuracy**
256-512-512	64	9.3M	0%	67.5%
128-512-512	64	8.1M	13%	66.6%
256-256-512	64	5.7M	38%	67.4%
128-256-512	64	4.8M	48%	66.0%
128-128-512	64	3.6M	60%	64.6%
256-256-256	128	3.3M	64%	67.0%
128-256-256	128	2.4M	74%	65.5%
128-128-256	128	1.6M	83%	64.5%
128-128-128	256	0.9M	89%	64.7%

5.3 End-to-end Evaluations with ORION [10]

We conduct end-to-end inference experiments comparing LIME, including its pruned models, with ORION [10] to assess overall performance. While ORION [10] provides a framework for private inference, it features efficient convolution operations and supports integration with various activation functions. Accordingly, we evaluate ORION with two nonlinear activation schemes proposed by Lee et al. [24] and AESPA [32], denoted as ORION-M and ORION-A, respectively. Specifically, ORION-M approximates ReLU using high-degree polynomials, whereas ORION-A employs quadratic polynomials.

5.3.1 Evaluations with Original Models. As shown in Table 8, the accuracies of ORION-M and ORION-A are the same as those of Lee et al. [24] and AESPA [32], respectively, as presented in Table 4. Across various testing scenarios, LIME achieves its largest improvements on ResNet-20 with the CIFAR-10 dataset, outperforming ORION-M and ORION-A by up to 41.5× and 8×, respectively. For VGG-11 on CIFAR-100 and ResNet-18 on Tiny-ImageNet, LIME attains a 5.3× to 6.6× speedup over ORION-M and a 2.1× to 2.3× improvement over ORION-A.

Table 8. Accuracy and amortized runtime of HE-based approaches. Best values in bold.

Model-Dataset	**Work**	**Accuracy**	**Runtime**			
			Linear (s)	**Activation (s)**	**Bootstrapping (s)**	**Total Runtime (s)**
ResNet-20-C10	ORION-M	90.1%	15.9	34.8	389.4	440.1
	ORION-A	87.1%	44.2	2.1	38.2	84.5
	LIME (this work)	**91.6%**	**5.2**	**0.3**	**5.1**	**10.6**
VGG-11-C100	ORION-M	65.4%	51.6	14.4	189.5	255.5
	ORION-A	59.1%	59.4	2.7	19.4	81.5
	LIME (this work)	**67.5%**	**35.1**	**2.3**	**1.1**	**38.5**
ResNet-18-TIN	ORION-M	55.8%	170.5	45.6	685.0	901.1
	ORION-A	48.3%	202.6	5.2	179.7	387.5
	LIME (this work)	**56.5%**	**137.1**	**4.2**	**28.6**	**169.9**

5.3.2 Evaluations with Compressed Models. As shown in Figure 7, LIME substantially decreases the per-sample runtime. When 90% of the parameters are pruned, ResNet-20 and VGG-11 exhibit a speed up to 5×, while ResNet-18 achieves a 3.3× improvement. In contrast, the runtime of ORION [10] remains unaffected mainly by pruning. Consequently, on CIFAR-10, LIME surpasses ORION-M and ORION-A by 202.5× and 35.1×, respectively, when using ResNet-20. For VGG-11, the performance improvements reach 31.1× and 8.4×, and for ResNet-18, they are 15.2× and 3.3×, relative to ORION-M and ORION-A, respectively. Since LIME and ORION [10] share identical model architectures, their accuracies remain the same, as summarized in Table 6 and Table 7.

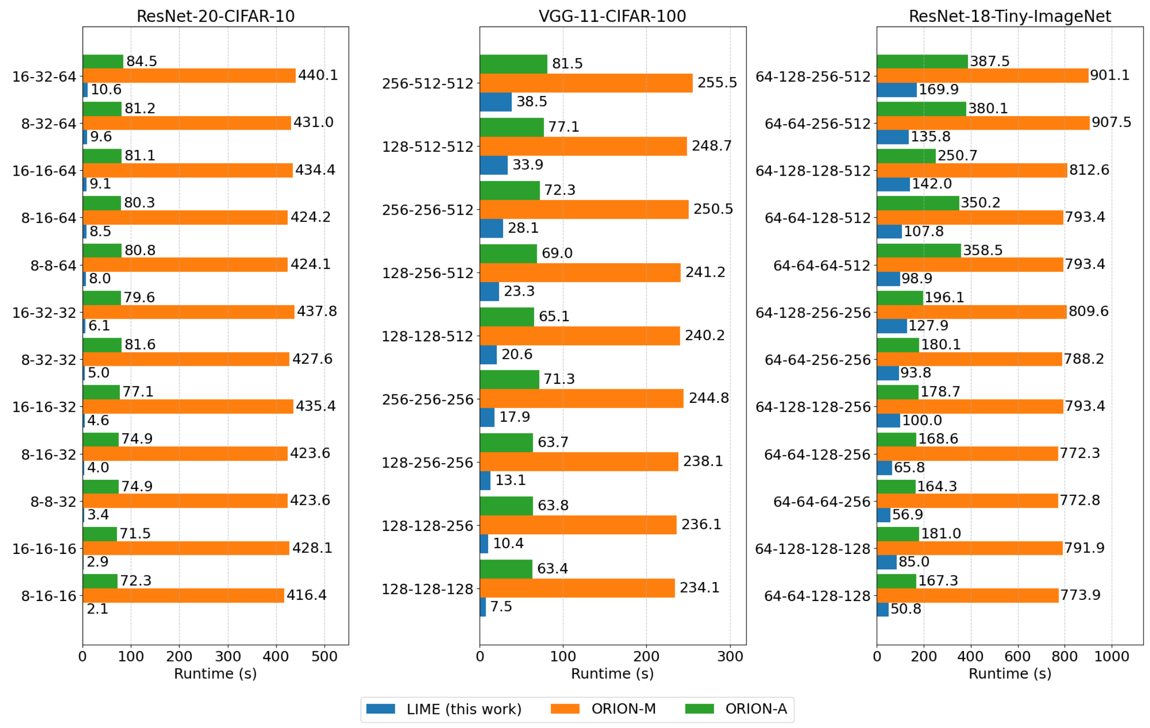


Fig. 7. Amortized time of different compressed models, sorted by compression ratio.

5.3.3 Memory Analysis. On average, the memory consumption of LIME ranges from 220 GB to 290 GB across all three evaluation models, whereas ORION [10] exhibits a broader range of memory usage. For the simpler model, ResNet-20, ORION [10] requires around 125 GB of memory, while for a more complex model, it requires up to 440 GB. However, it should be noted that the memory consumption in LIME is shared across batch samples, whereas the memory consumption in ORION [10] corresponds to a single sample.

6 Conclusion and Future Work

This paper presents LIME, a high-performance solution for private inference, based on homomorphic encryption. LIME employs element-wise channel-to-slot packing (ECSP) and power-of-two channel pruning to facilitate batch encryption and model compression, enabling the deployment of lightweight versions of large-scale models that optimize resource utilization through batch processing. By employing the ReLU-before-addition block, LIME reduces the accuracy loss associated with ReLU approximations using quadratic polynomials.

By employing innovative algorithms and architectures, LIME attains superior accuracy and runtime performance while substantially reducing computational costs and environmental impact. We contend that LIME offers a sustainable framework for advancing a cleaner, energy-efficient, and safer AI ecosystem. Leveraging the availability and efficacy of ECSP-based convolution, LIME can be extended to accommodate a broader range of model architectures with sequential convolutions, such as MobileNet [18] and YOLO [34], for future object detection tasks.

Acknowledgments

This project received funding from the National Science and Technology Council in Taiwan (111-2221-E-002-134-MY3).

References

[1] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, et al. 2021. Homomorphic Encryption Standard. *Protecting privacy through homomorphic encryption* (2021).

[2] Amazon. 2025. Getting Recommendations from Amazon Personalize. https://docs.aws.amazon.com/personalize/latest/dg/getting-recommendations.html.

[3] Wei Ao and Vishnu Naresh Boddeti. 2024. {AutoFHE}: Automated Adaption of {CNNs} for Efficient Evaluation over {FHE}. In *USENIX Security.*

[4] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. Bootstrapping for Approximate Homomorphic Encryption. In *EUROCRYPT.*

[5] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *ASIACRYPT.*

[6] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. 2018. Faster Cryptonets: Leveraging Sparsity for Real-World Encrypted Inference. *arXiv:1811.09953* (2018).

[7] McKinsey & Company. 2024. AI Power: Expanding Data Center Capacity to Meet Growing Demand. https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/ai-power-expanding-data-center-capacity-to-meet-growing-demand.

[8] Jingtian Dang, Jianming Tong, Anupam Golder, Cong Hao, Arijit Raychowdhury, Tushar Krishna, et al. 2024. Accurate Low-Degree Polynomial Approximation of Non-Polynomial Operators for Fast Private Inference in Homomorphic Encryption. *MLSys* (2024).

[9] Pierre Vilar Dantas, Waldir Sabino da Silva Jr, Lucas Carvalho Cordeiro, and Celso Barbosa Carvalho. 2024. A Comprehensive Review of Model Compression Techniques in Machine Learning. *Applied Intelligence* (2024).

[10] Austin Ebel, Karthik Garimella, and Brandon Reagen. 2025. Orion: A Fully Homomorphic Encryption Framework for Deep Learning. In *ASPLOS.*

[11] Zahra Ghodsi, Akshaj Kumar Veldanda, Brandon Reagen, and Siddharth Garg. 2020. Cryptonas: Private inference on a ReLU Budget. *Advances in Neural Information Processing Systems* (2020).

[12] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *ICML.*

[13] Yiwen Guo, Anbang Yao, and Yurong Chen. 2016. Dynamic Network Surgery for Efficient DNNs. *NIPS* (2016).

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity Mappings in Deep Residual Networks. In *ECCV.*

[15] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel Pruning for Accelerating Very Deep Neural Networks. In *ICCV.*

[16] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. 2017. Cryptodl: Deep Neural Networks over Encrypted Data. *arXiv:1711.05189* (2017).

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the Knowledge in a Neural Network. *arXiv:1503.02531* (2015).

[18] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861* (2017).

[19] Nandan Kumar Jha, Zahra Ghodsi, Siddharth Garg, and Brandon Reagen. 2021. Deepreduce: ReLU Reduction for Fast Private Inference. In *ICML.*

[20] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. {GAZELLE}: A Low Latency Framework for Secure Neural Network Inference. In *USENIX security.*

[21] Aleksandar Krastev, Nikola Samardzic, Simon Langowski, Srinivas Devadas, and Daniel Sanchez. 2024. A Tensor Compiler with Automatic Data Packing for Simple and Efficient Fully Homomorphic Encryption. *PACMPL* (2024).

[22] C-C Jay Kuo and Azad M Madni. 2023. Green Learning: Introduction, Examples and Outlook. *Journal of Visual Communication and Image Representation* (2023).

[23] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. 2022. Low-Complexity Deep Convolutional Neural Networks on Fully Homomorphic Encryption Using Multiplexed Parallel Convolutions. In *ICML.*

[24] Eunsang Lee, Joon-Woo Lee, Jong-Seon No, and Young-Sik Kim. 2021. Minimax Approximation of Sign Function by Composite Polynomial for Homomorphic Comparison. *IEEE Transactions on Dependable and Secure Computing* (2021).

[25] Junghyun Lee, Eunsang Lee, Joon-Woo Lee, Yongjune Kim, Young-Sik Kim, and Jong-Seon No. 2023. Precise Approximation of Convolutional Neural Networks for Homomorphically Encrypted Data. *IEEE Access* (2023).

[26] Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, et al. 2022. Privacy-Preserving Machine Learning with Fully Homomorphic Encryption for Deep Neural Network. *IEEE Access* (2022).

[27] Zhuo Li, Hengyi Li, and Lin Meng. 2023. Model compression for deep neural networks: A survey. *Computers* (2023).

[28] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On Ideal Lattices and Learning with Errors Over Rings. In *EUROCRYPT*.

[29] Yukta Mehta, Rui Xu, Benjamin Lim, Jane Wu, and Jerry Gao. 2023. A Review for Green Energy Machine Learning and AI Services. *Energies* (2023).

[30] OpenAI. 2025. Batch API. https://platform.openai.com/docs/guides/batch.

[31] OpenMP Architecture Review Board. 2021. OpenMP Application Program Interface. https://www.openmp.org.

[32] Jaiyoung Park, Michael Jaemin Kim, Wonkyung Jung, and Jung Ho Ahn. 2022. AESPA: Accuracy Preserving Low-Degree Polynomial Activation for Fast Private Inference. *arXiv:2201.06699* (2022).

[33] Robert Podschwadt, Daniel Takabi, and Peizhao Hu. 2021. Sok: Privacy-preserving deep learning with homomorphic encryption. *arXiv:2112.12855* (2021).

[34] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. In *CVPR*. 779–788.

[35] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald Dreslinski, Christopher Peikert, and Daniel Sanchez. 2021. F1: A Fast and Programmable Accelerator for Fully Homomorphic Encryption. In *MICRO*.

[36] Sapien. 2024. Batch Inference. https://www.sapien.io/glossary/definition/batch-inference.

[37] Samuel Sousa and Roman Kern. 2023. How to Keep Text Private? A Systematic Review of Deep Learning Methods for Privacy-Preserving Natural Language Processing. *Artificial Intelligence Review* (2023).

[38] Huan-Chih Wang and Ja-Ling Wu. 2026. LIME: High-Performance Private Inference with Lightweight Model and Batch Encryption. *ACM Transactions on Multimedia Computing, Communications, and Applications* (2026). doi:10.1145/3779221

[39] Runhua Xu, Nathalie Baracaldo, and James Joshi. 2021. Privacy-preserving Machine Learning: Methods, Challenges and Directions. *arXiv:2108.04417* (2021).

[40] Wenxuan Zeng, Meng Li, Haichuan Yang, Wen-jie Lu, Runsheng Wang, and Ru Huang. 2023. Copriv: Network/Protocol Co-Optimization for Communication-Efficient Private Inference. *Advances in Neural Information Processing Systems* (2023).

[41] Daochen Zha, Zaid Pervaiz Bhat, Kwei-Herng Lai, Fan Yang, Zhimeng Jiang, Shaochen Zhong, and Xia Hu. 2025. Data-Centric Artificial Intelligence: A Survey. *Comput. Surveys* (2025).