Toward Practical Privacy-Preserving Convolutional Neural Networks
Exploiting Fully Homomorphic Encryption

Jaiyoung Park*, Donghwan Kim*, Jongmin Kim*, Sangpyo Kim*, Wonkyung Jung†, Jung Hee Cheon*, and Jung Ho Ahn*
*Seoul National University, Seoul, South Korea
†Samsung Electronics, Suwon, South Korea
{jeff1273, eastflame, jongmin.kim, vn987, jhcheon, gajh}@snu.ac.kr
wk2.jung@samsung.com

Abstract
Incorporating fully homomorphic encryption (FHE) into the inference process of a convolutional neural network (CNN) draws enormous attention as a viable approach for achieving private inference (PI). FHE allows delegating the entire computation process to the server while ensuring the confidentiality of sensitive client-side data. However, practical FHE implementation of a CNN faces significant hurdles, primarily due to FHE’s substantial computational and memory overhead. To address these challenges, we propose a set of optimizations, which includes GPU/ASIC acceleration, an efficient activation function, and an optimized packing scheme. We evaluate our method using the ResNet models on the CIFAR-10 and ImageNet datasets, achieving several orders of magnitude improvement compared to prior work and reducing the latency of the encrypted CNN inference to 1.4 seconds on an NVIDIA A100 GPU. We also show that the latency drops to a mere 0.03 seconds with a custom hardware design.

1. Introduction
Privacy regulations such as GDPR [22] have propelled data confidentiality to the forefront of concerns for cloud companies offering machine learning (ML) as a service. Fully homomorphic encryption (FHE) [7], which can evaluate arbitrary functions on encrypted data called ciphertext, has garnered attention as a promising solution for achieving robust security assurance. However, FHE-based computation exhibits distinct characteristics from their unencrypted counterpart, and naively applying FHE to ML inference results in significant inefficiencies. These disparities pose challenges in developing an end-to-end ML framework with FHE, which has been perceived to be costlier than alternative security solutions [9, 19].

We unlock the potential of FHE-based solutions in the context of private inference (PI). Our focus lies on the framework for PI of convolutional neural networks (CNNs). We employ the RNS-CKKS FHE scheme [5], which supports handling real and complex numbers. The PI framework involves two parties: a client requesting CNN inference on encrypted data and a server conducting the evaluation using FHE operations. The robust security offered by RNS-CKKS guarantees the non-disclosure of sensitive information, encompassing both the client’s input data and the server’s CNN model while securely delivering the inference results back to the client.

Our contributions span various layers of computer systems, and each contribution can be utilized separately to accelerate FHE-based CNN inference in other systems. With all techniques combined, we achieve orders of magnitude higher performance for FHE-based CNN inference, reducing the encrypted inference latency of ResNet20 to a mere 1.4 seconds on a GPU and 0.03 seconds on a custom hardware design.

The following list summarizes the key contributions:
• We are unlocking new horizons in FHE-based PI by harnessing the latest GPU advancements and incorporating techniques derived from accelerator-based research, thereby introducing a new achievable milestone in this field.
• We combine AESPA [18], a novel activation function tailored to FHE-based PI, which converts ReLU into a simple quadratic function during inference through training-time specialization.
• We leverage an efficient CNN library with novel packing methods named HyPHEN [11], which reduces the computation and memory requirements of FHE-based PI of CNNs.

2. A GPU Library Supporting RNS-CKKS
Due to the high degree of parallelism in RNS-CKKS, GPUs have a great potential for performance enhancement by utilizing the massive number of computing resources in GPUs through parallelized computation. In RNS-CKKS, the unit of computation is a polynomial in \( \mathbb{Z}_q[X]/(X^N + 1) \), which is an \( L \times N \) integer matrix where \( N \) is the degree of the polynomial and \( L \) is the maximum multiplicative level of a ciphertext. A polynomial is a huge matrix; typical values for \( L \) and \( N \) are respectively around 1–60 and 2^{15}–2^{16}. By using GPUs, various computations on polynomials required for RNS-CKKS can mostly be performed in parallel. For example, two polynomials can be added by \( L \times N \) parallel element-wise additions.

However, the large size of the computational granularity also results in a memory bottleneck because faster memory units (e.g., L2 cache) in GPUs do not have enough capacity to accommodate it. To overcome the bottleneck, we applied memory-centric optimizations for major computation kernels, such as NTT and base conversion, in prior work [10, 14]. We have fused multiple GPU kernels to perform multiple jobs at once for each memory load and identified eligible RNS-CKKS parameter sets for GPUs (P2 and P3 in Table 1 vs. P1).
Table 1: Execution time (ms) comparison of our GPU library vs. state-of-the-art RNS-CKKS GPU acceleration studies, 100 × [10] and TensorFHE [6]. P1, P2, P3: fused, fused2, and fused4 parameters in Table 2 of [10]. P4, P5: parameters in Table 3 of [10].

<table>
<thead>
<tr>
<th>Impl.</th>
<th>100×</th>
<th>TensorFHE</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU Word size</td>
<td>V100</td>
<td>A100-SXM</td>
<td>A100-PCIe</td>
</tr>
<tr>
<td>64 bits</td>
<td>17.40</td>
<td>6.65</td>
<td>11.30</td>
</tr>
<tr>
<td>32 bits</td>
<td>2.96</td>
<td>-</td>
<td>2.59</td>
</tr>
<tr>
<td>64 bits</td>
<td>7.96</td>
<td>-</td>
<td>5.47</td>
</tr>
<tr>
<td>HMult (P1)</td>
<td>328.25</td>
<td>250.45</td>
<td>171.27</td>
</tr>
<tr>
<td>HMult (P2)</td>
<td>526.96</td>
<td>-</td>
<td>355.84</td>
</tr>
<tr>
<td>HMult (P3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boot (P4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boot (P5)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

† TensorFHE execution time is divided by 128 as it batches 128 operations.


ML frameworks such as PyTorch offer multiple memory formats dictating the data order of tensors (e.g., channel-last memory format). Each format requires a distinct kernel to process the same operation. This concept applies similarly in the context of FHE, where the memory format is analogous to packing in FHE, although packing has more significant performance implications. Numerous studies [2, 4, 8] have proposed different packing methods for FHE-based CNN, but they did not account for bootstrapping. More recently, [16] demonstrated an FHE-based CNN implementation with bootstrapping operations. The packing method introduced in [16] utilizes a single dense packing, which can effectively minimize the number of bootstrapping. However, their method suffers from the high cost of frequent homomorphic rotation operations, which are required to maintain the fixed packing method of [16], whereas input and output ciphertexts have different data orientations in FHE-based convolution layers; rotations account for more than 83% of the total convolution time in [16].

To mitigate this inefficiency, we designed HyPHEN, an FHE-based CNN library incorporating multiple packing methods. By offering flexibility in the packing methods, we minimize the cost for rotations by choosing different packing methods before and after each convolution layer. Figure 1 depicts our construction of ResNet20. Each basic block consists of two different convolution layers (each consuming two multiplicative levels) arranged in an interleaved manner with AESPA (each consuming one multiplicative level) activation functions. We could minimize the cost of bootstrapping by placing the operation prior to the second activation function in a basic block, where the number of ciphertexts is small. Please refer to [11] for a more detailed description of AESPA.
Due to the limited GPU memory capacity, we could not run ResNet18 directly on the GPU with [16]. To estimate its performance, we assumed that the GPU can contain the entire working set, ignoring the host-to-GPU data movement cost. Meanwhile, AESPA + HyPHEN is free from this problem due to optimizations that effectively reduce the working set size.

5. Evaluation and Discussion

We trained ResNet20 on the CIFAR10 dataset and ResNet18 on the ImageNet dataset using AESPA within PyTorch for both models. CNN models trained with AESPA show comparable classification accuracies to the original networks with ReLU. ResNet20 achieves an accuracy of 92.18%, whereas the original accuracy of ResNet20 is 92.15%. Similarly, ResNet18 achieves an accuracy of 69.78%, whereas the original model’s accuracy is 69.75%.

The latency of CNN inference for the two models is shown in Table 2. We reduce the inference latency of ResNet20 to mere 1.4 seconds, 1,622 × faster than reported in [16]. Our GPU library reduces the execution time by 21.0 × compared to that of our 64-thread CPU implementation. Applying AESPA and HyPHEN results in an additional 6.12 × speedup. Meanwhile, our optimized ResNet18 inference takes 14.7 seconds.

There still exists a vast room for further improvement in FHE-based CNN performance. In particular, numerous specialized hardware accelerator designs [12, 13, 15, 20, 21] have been proposed recently. We estimated the performance of our CNN implementation on a state-of-the-art hardware accelerator proposal, SHARP [12], through simulation (see Table 2). Simulation results show that specialized hardware enables real-time FHE-based CNN inference by reducing the inference time to as low as 30 milliseconds, 75,700 × faster than the original single-threaded CPU implementation.

References