https://doi.org/10.7840/kics.2025.50.11.1669

RNS-CKKS 기반 ResNet 추론을 위한 병합 부트스트래핑 가속 기법

박 수 민◆, 이 은 상°

Accelerating ResNet Inference on RNS-CKKS with Merged Bootstrapping

Soomin Park◆, Eunsang Lee°

요 약

최근 완전동형암호 기반의 컨볼루션 신경망 연구가 활발히 진행되고 있으며, 대표적인 완전동형암호 스킴인 residue number system variant of Cheon‐Kim‐Kim‐Song(RNS-CKKS) 상에서 residual network(ResNet)을 높은 정확도로 구현한 사례가 보고되었다. 그러나 기존 방법은 암호문 슬롯을 충분히 활용하지 못해 부트스트래핑을 자주 수행해야 하고, 그만큼 계산 시간이 길어지는 한계가 있다. 이를 개선하기 위해 본 논문에서는 여러 이미지에 대해 동시에 ResNet을 수행하는 경우, 복수의 암호문을 단일 암호문으로 병합한 후 부트스트래핑을 수행하는 '병합 부트스트래핑' 방법을 제안한다. 이는 암호문의 전체 슬롯을 효율적으로 활용하여 수행 시간을 크게 감소시키는 방법이다. RNS-CKKS 스킴 라이브러리인 Lattigo에서의 실험 결과, 병합 부트스트래핑을 사용하여 2개, 4개, 8개의 Canadian Institute for Advanced Research-10(CIFAR-10) 이미지를 ResNet-20으로 분류할 경우, 기존 방법에 비해 평균 부트스트래핑 시간이 각각 39%, 55%, 59% 줄어들었음을 확인하였다. 또한, 병합 부트스트래핑을 사용한 ResNet-20에 의한 2개 이미지 분류 구현 결과 평균 수행 시간이 기존 연구에 비해 37% 감소하였다.

키워드 : 부트스트래핑, 컨볼루션 신경망, 동형 암호, 잔차신경망
Key Words : bootstrapping, convolutional neural networks, homomorphic encryption, ResNet

ABSTRACT

Recent studies on convolutional neural networks based on fully homomorphic encryption(FHE) have gained momentum, and there are reports of implementing ResNet with high accuracy on the representative FHE residue number system variant of Cheon‐Kim‐Kim‐Song(RNS-CKKS). However, existing approaches under utilize ciphertext slots, requiring frequent bootstrapping and therefore incurring substantial computational overhead. In response to this, our paper proposes a 'merged bootstrapping' method that, when performing residual network(ResNet) on multiple images simultaneously, merges multiple ciphertexts into a single ciphertext before bootstrapping. This effectively utilizes all slots of the ciphertext, significantly reducing computation time. Experimental results using the RNS-CKKS scheme library, Lattigo, confirmed that when classifying 2, 4, and 8 Canadian Institute for Advanced Research-10(CIFAR-10) images with ResNet-20 using merged bootstrapping,

※ 이 논문은 2025년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (No. 2022R1I1A1A01068284).
※ 이 논문은 과학기술정보통신부 및 정보통신기획평가원의 2025년 SW중심대학사업 지원을 받아 수행되었음(2024-0-00037).
◆ First Author : Department of Computer Engineering, Chung-Ang University, rft753@gmail.com, 정회원
° Corresponding Author : Department of Software, Sejong University, eslee3209@sejong.ac.kr, 정회원
논문번호 : 202504-091-A-RN, Received April 19, 2025; Revised May 14, 2025; Accepted May 26, 2025

the per-image computation time was reduced by 43%, 58%, and 61% respectively, compared to the previous methods. Additionally, the implementation of two-image classification using ResNet-20 with merged bootstrapping resulted in a 37% reduction in amortized runtime compared to previous work.

Ⅰ. 서 론

동형암호(homomorphic encryption)는 암호화된 상태에서도 대수 연산을 수행할 수 있게 해 주는 특수한 암호 방식이다. 초창기에는 암호화된 데이터 위에 제한된 횟수의 동형 연산만 허용되었으나, 2009년 Gentry가 무제한 연산을 지원하는 완전동형암호(fully homomorphic encryption, FHE) 체계를 제안하면서 획기적인 진전이 이뤄졌다[1]. 이후 연구들은 연산 속도와 정밀도를 지속적으로 개선해 왔으며, 현재 완전동형암호는 컴퓨터 비전, 자연어 처리, 통계 분석, 의료 데이터 등 다양한 분야에서 프라이버시 보호 도구로 활용되고 있다[2].

최근에는 완전동형암호 환경에서 컨볼루션 신경망(convolutional neural networks)을 실행하려는 연구가 활발하다[3-6]. 이 연구들이 초점을 두고 있는 클라이언트-서버 시스템에서는 사용자가 이미지를 완전동형암호로 암호화해 서버에 전송하면, 서버는 복호화 없이 이미지 추론을 수행한 뒤 암호화된 결과만 돌려준다. 덕분에 서버는 원본 데이터를 전혀 볼 수 없고, 비밀키를 가진 클라이언트만이 결과를 확인할 수 있다.

본 연구는 완전동형암호 스킴 가운데 널리 쓰이는 RNS-CKKS(residue number system variant of Cheon‐Kim‐Kim‐Song)[7,8] 기반 컨볼루션 신경망 연산에 초점을 맞춘다. RNS-CKKS는 실수형 데이터를 한 암호문에 대량으로 담아 빠르게 계산할 수 있어 각광받고 있다. 최근 RNS-CKKS를 이용해 residual network(ResNet)으로 CIFAR-10 이미지를 분류한 연구가 보고됐는데[4], 이들은 효율적인 데이터 인코딩 기법과 rectified linear unit(ReLU) 함수의 다항식 근사 덕분에 높은 정확도를 달성했다.

하지만 선행 연구[4]에서는 처리 시간이 여전히 길었으며, 이는 주로 부트스트래핑 단계 때문이었다. RNS-CKKS 스킴에서 부트스트래핑이 가장 고비용 연산이므로, 그 횟수를 줄이는 것이 필수적이다. Canadian Institute for Advanced Research-10 (CIFAR-10) 이미지 데이터를 분류하는 경우 암호문에 포함되는 데이터의 양이 전체 슬롯(slot) 개수에 비해 적어, 기존 연구에서는 암호문 전체 슬롯을 완전히 활용하지 않은 상태에서 부트스트래핑을 수행하였다. 이러한 방식은 비효율적이며, 결과적으로 더 많은 부트스트래핑을 요구하게 된다.

본 논문에서는 여러 이미지에 대해 동시에 ResNet 연산을 수행하는 경우, 암호문 전체 슬롯을 활용한 상태에서 부트스트래핑을 수행함으로써 연산 시간을 줄이는 방법을 제안한다. 구체적으로는 병합 알고리즘 $Merge(k,s)$ 및 분리 알고리즘 $Split(k,s)$을 도입한다. 부트스트래핑을 수행하기 직전에 여러 암호문들을 하나의 암호문으로 병합한 후 부트스트래핑을 수행하고, 다시 여러 암호문으로 분리하는 방식이다. 이렇게 하면 여러 암호문에 대해 수행해야 할 부트스트래핑을 하나의 부트스트래핑으로 대체함으로써 전체적인 연산 시간을 크게 줄일 수 있다.

본 연구에서는 유명한 RNS-CKKS 스킴 라이브러리인 Lattigo[9]를 활용하여 제안하는 병합 부트스트래핑 사용 시, 전체 ResNet에서 필요한 부트스트래핑 수행 시간을 분석한다. 제안하는 병합 부트스트래핑을 사용해 각각 2개, 4개, 8개의 CIFAR-10 이미지를 동시에 ResNet-20으로 분류할 경우, 분할 상환 시간(이미지 당 시간)이 각각 415초, 306초, 283초가 소요된다. 이는 기존 연구[4]에서 제시된 분할 상환 시간인 683초에 비해 각각 39%, 55%, 59%의 시간이 줄어든 것을 의미한다. 또한, 2개의 이미지를 병합 부트스트래핑을 사용하여 동시에 분류하는 경우를 실제로 구현한 결과 분할 상환 시간이 기존 연구에 비해 37% 감소하였다.

Ⅱ. 배경지식

본 장에서는 완전동형암호 RNS-CKKS에 대한 개념과 RNS-CKKS 상에서의 컨볼루션 신경망 연산 방법에 대해 소개한다.

2.1 RNS-CKKS

RNS-CKKS 스킴[7,8]은 완전동형암호를 대표하는 방식 가운데 하나로, 암호화된 상태에서도 대수 연산을 수행할 수 있게 한다. 스킴의 모든 연산은 다항식 환(polynomial ring)에서 정의되며, 다항식 차수가 N 이면 하나의 암호문에는 $n = N/2$개의 실수(또는 복소수) 요소로 이루어진 벡터가 담긴다. RNS-CKKS는 이 벡터에 대해 성분별 덧셈과 곱셈을 동형적으로 지원한다.

암호화와 복호화를 각각 Enc, Dec로 표기하면, 벡터 $\boldsymbol{a}\in\mathbb{R}^n$을 암호화한 결과는 $Enc(\boldsymbol{a})$이고 복호화를 거치면 $\boldsymbol{a}=Dec(Enc(\boldsymbol{a}))$가 성립한다. 벡터 $\boldsymbol{a}$의 각 요소는 암호문에서 '슬롯'으로 불리며 슬롯의 총 개수는 n이다.

RNS-CKKS 스킴의 동형 덧셈($\oplus$), 스칼라 곱셈($\odot$), 넌스칼라 곱셈($\otimes$) 연산은 다음을 만족한다.

$$Enc(\boldsymbol{a})\oplus Enc(\boldsymbol{b}) = Enc(\boldsymbol{a}+\boldsymbol{b})$$
$$Enc(\boldsymbol{a})\odot\boldsymbol{b}=\boldsymbol{a}\odot Enc(\boldsymbol{b}) = Enc(\boldsymbol{a}\cdot\boldsymbol{b})$$
$$Enc(\boldsymbol{a})\otimes Enc(\boldsymbol{b}) = Enc(\boldsymbol{a}\cdot\boldsymbol{b})$$

여기서 $\boldsymbol{a}\cdot\boldsymbol{b}$는 두 벡터 $\boldsymbol{a},\boldsymbol{b}$의 성분별 곱셈 결과 벡터를 나타낸다.

또한, RNS-CKKS 스킴에서는 벡터의 성분들의 위치를 이동시키는 회전 연산 $Rot(\cdot)$을 제공한다. 어떤 벡터 $\boldsymbol{a}=(a_0,a_1,\cdots,a_{n-1})\in\mathbb{R}^n$ 및 음이 아닌 정수 r에 대해 다음 식이 성립한다.

$$Rot(Enc(\boldsymbol{a});r) = Enc((a_r,a_{r+1},\cdots,a_{n-1},a_0,\cdots a_{r-1}))$$

여기서 $(a_r,a_{r+1},\cdots,a_{n-1},a_0,\cdots a_{r-1})$은 벡터 $\boldsymbol{a}$를 왼쪽으로 r만큼 순환 이동(cylic shift)시킨 벡터를 나타낸다. 만약 r이 음수라면 $Rot(Enc(\boldsymbol{a});r)$은 벡터 $\boldsymbol{a}$의 성분들을 오른쪽으로 r만큼 순환 이동시킨다.

RNS-CKKS 스킴을 활용하면 어떤 대수적 연산이든 암호화된 상태에서 수행 가능하다. 예를 들어, 어떤 실수 벡터 $\boldsymbol{x}$에 대해 다수의 성분별 덧셈, 곱셈을 사용하여 실수 벡터 $\boldsymbol{y}$를 출력하는 연산 회로 F를 생각해보자. 즉, $\boldsymbol{y}=F(\boldsymbol{x})$가 성립하는 상황이다. 이 때, 모든 덧셈을 $\oplus$로 대체하고 모든 상수곱을 $\odot$로, 모든 변수간의 곱셈을 $\otimes$로 대체하여 생성한 동형 연산 회로를 $\overline{F}$라 하자. 이 경우, 다음의 식이 성립한다.

$$\boldsymbol{y}=F(\boldsymbol{x}) = Dec(\overline{F}(Enc(\boldsymbol{x})))$$

따라서, 입력 $\boldsymbol{x}$에 대해 연산 회로 F를 수행하는 것은, 먼저 $\boldsymbol{x}$를 암호화한 다음에 암호화된 상태에서 F에 대응하는 동형 연산 회로 $\overline{F}$을 수행하고, 마지막으로 결과를 복호화하는 과정과 동일하다는 것을 알 수 있다. 이를 클라이언트-서버 시스템에 적용해본다면, 서버는 클라이언트의 암호화된 이미지 $Enc(\boldsymbol{x})$에 대해 동형 연산 회로 $\overline{F}$를 수행하는 상황에 해당된다. 이러한 방식으로, 클라이언트는 자신의 데이터를 서버에게 노출하지 않아도 이미지 추론 서비스를 이용할 수 있게 된다.

RNS-CKKS 스킴에서는 연속적인 곱셈을 수행할 수 있는 횟수가 암호문의 '레벨'로 제한된다. 곱셈을 한 번 실행할 때마다 레벨이 감소하고, 레벨이 0에 도달하면 부트스트래핑[10]을 통해 레벨을 다시 높여야 한다. 이 과정을 주기적으로 거치면 사실상 무제한으로 동형 연산을 이어갈 수 있지만, 부트스트래핑은 스킴 전체에서 가장 시간이 오래 걸리는 단계이므로 그 횟수를 최소화하는 것이 성능 개선의 핵심이다.

2.2 RNS-CKKS 상에서의 컨볼루션 신경망

최근에 RNS-CKKS 상에서 암호화된 데이터로 컨볼루션 신경망을 계산하는 연구가 활발히 진행되고 있다[5,6]. 컨볼루션 신경망에서는 비대수적 활성화 함수(activation function) ReLU 함수를 포함하는 경우가 많아, 이를 RNS-CKKS 스킴에서 직접적으로 계산하는 것은 어렵다. 그래서 일반적으로 ReLU 함수를 다항식으로 근사하고, 이 근사된 다항식을 동형 연산으로 연산한다. 최근에는 ReLU 함수를 정밀하게 다항식으로 근사하는 방법[11,12]이 연구되었고, ReLU 함수를 이 근사 다항식으로 대체한 근사 컨볼루션 신경망이 기존 컨볼루션 신경망과 매우 유사한 추론 정확도를 보이는 것이 확인되었다[3].

또한, 컨볼루션 신경망은 기본적으로 3차원 텐서(tensor) 간의 복잡한 연산을 수행하는데 RNS-CKKS에서는 1차원 벡터 연산만을 지원한다. 기존 연구[4]에서는 RNS-CKKS 상에서 1차원 벡터 연산만을 활용하면서도 대표적인 컨볼루션 신경망인 ResNet을 수행하는 방법을 개발하였다[4]. 이 논문에서는 먼저 3차원 텐서 $A=(A_{i,j,k})_{0\le i<h_i,0\le j<w_i,0\le k<c_i}\in\mathbb{R}^{h_i\times w_i\times c_i}$을 1차원 벡터 $\boldsymbol{v}\in\mathbb{R}^{h_iw_ic_i}$로 맵핑하는 함수 $MultPack$를 정의한다. 3차원 텐서를 1차원 벡터에 맵핑할 때 일정한 간격(gap)을 두면서 맵핑하는데 그 간격을 k_i라 하자. 간단한 설명을 위해 $k_i^2|c_i$이라고 가정하자. 만약, c_i가 k_i^2의 배수가 아니라면 텐서에 0 값을 가진 추가적인 채널을 채널 A에 덧붙여 채널 개수가 k_i^2의 배수가 되도록 조정한다. $t_i=c_i/k_i^2$라 하자. $MultPack$ 함수를 수행하기 위해, 먼저 텐서 A를 또 다른 3차원 텐서 $A'=(A'_{i,j,k})_{0\le i<k_ih_i,0\le j<k_iw_i,0\le k<t_i}\in\mathbb{R}^{k_ih_i\times k_iw_i\times t_i}$로 맵핑하는데 이 때, $A'_{i,j,k}=A_{\lfloor i/k_i\rfloor,\lfloor j/k_i\rfloor,\ k_i^2k+k_i(i\bmod k_i)+j\bmod k_i}$가 되도록

한다. 이 과정은 k_i^2개의 채널 텐서를 하나의 큰 텐서에 멀티플렉싱(multiplexing)[4]하는 형태이다. 그 후, 텐서 A'를 1차원 벡터 $\boldsymbol{v} \in \mathbb{R}^{k_i^2 h_i w_i t_i} = \mathbb{R}^{h_i w_i c_i}$에 맵핑하는데 래스터-스캔 방식(raster-scan fashion)으로 순차적으로 맵핑한다. 구체적으로는, 첫 번째 채널부터 마지막 채널까지 순서대로 데이터를 맵핑한다. 각 채널의 데이터를 맵핑할 때, 왼쪽에서 오른쪽으로 수평적으로 값을 나열하고, 이후 아래 행으로 이동하여 이 과정을 반복한다. 이 때, 이 $\boldsymbol{v} \in \mathbb{R}^{h_i w_i c_i}$가 바로 $MultPack(A)$가 된다.

ResNet 파라미터의 경우 어떤 $n_s | n (= 2^{15})$에 대해 $h_i w_i c_i = n_s$가 된다. 이 때, $MultPack(A) \in \mathbb{R}_s^n$를 n/n_s번만큼 반복하여 얻은 벡터를 $\boldsymbol{v}' \in \mathbb{R}^n$ 이라고 할 때 $CptMultPack$ 함수는 $CptMultPack(A) = \boldsymbol{v}' \in \mathbb{R}^n$를 만족시키도록 정의된다. 즉, $\boldsymbol{v}' = [\boldsymbol{v} | \boldsymbol{v} | \cdots | \boldsymbol{v}]$를 만족한다.

기존 연구[4]에서 제시된 멀티플렉스 병렬 컨볼루션(multiplexed parallel convolution)은 $CptMultPack$ 형태로 RNS-CKKS로 암호화된 데이터에 대해 컨볼루션을 수행하더라도 결과가 동일한 $CptMultPack$ 형태로 유지되도록 설계되었다. 평문 ResNet에서의 특정 컨볼루션의 입력 텐서를 $A \in \mathbb{R}^{h_i \times w_i \times c_i}$라 하고 출력을 $A' \in \mathbb{R}^{h_o \times w_o \times c_o}$라 하자. 그러면 이에 대응하는 멀티플렉스 병렬 컨볼루션은 $Enc(CptMultPack(A))$를 입력으로 받아 $Enc(CptMultPack(A'))$를 출력하는 알고리즘이다. 기존 연구에서 구현된 RNS-CKKS 기반의 ResNet은 입력 이미지를 $CptMultPack$ 형태로 암호문에 패킹하고, 암호화된 상태에서 멀티플렉스 병렬 컨볼루션과 ReLU 등의 연산을 반복적으로 수행한다. 그리고 이 과정을 거친 후 마지막에 복호화를 통해 암호화된 상태에서의 ResNet 연산 전체를 완료한다.

Ⅲ. 병합 부트스트래핑

기존 연구[4]에서의 ResNet 구현에서 수행 시간의 70% 이상을 부트스트래핑이 차지하므로, 실행 시간을 줄이기 위해서는 부트스트래핑 과정의 시간을 최소화하는 것이 중요하다. 현재 문제점은 기존 연구에서 부트스트래핑 과정에서 암호문의 슬롯 전체를 활용하지 않아 비효율성이 발생하는 점이다. 본 연구에서는 이러한 문제를 해결하기 위해 여러 CIFAR-10 이미지에 대해 동시에 ResNet을 수행하면서 n개의 슬롯 전체를 최대한 활용하려고 한다. 이를 위해 제안하는 병합 및 분리 알고리즘을 사용하여 암호문의 모든 슬롯을 효율적으로 활용하며, 이를 통해 필요한 부트스트래핑의 횟수를 줄이고자 한다.

3.1 병합 알고리즘

기존 연구[4]에서 사용된 $CptMultPack$ 패킹 방법은 데이터를 n/n_s만큼 중복하여 암호문에 담는다. 이는 실제 데이터 개수 n_s가 암호문의 전체 슬롯 개수 n보다 작기 때문이며, 전체-슬롯 부트스트래핑 대신 희소-슬롯 부트스트래핑[10]을 사용하여 수행 시간을 줄이기 위한 것이다. 본 논문에서는 이러한 중복을 최소화하고, 데이터 전체 슬롯을 보다 효율적으로 활용하여 부트스트래핑의 횟수를 줄이기 위한 암호문의 병합 과정을 제안한다.

2의 거듭제곱인 k와 s가 주어졌다고 하자. k개의 암호문 $ct_0, ct_1, \cdots, ct_{k-1}$ 각각에는 n/ks개의 슬롯에만 실제 데이터가 들어 있고, 나머지 슬롯들은 0으로 채워져 있다. 이 암호문들을 하나로 합쳐, n개 모든 슬롯에 데이터가 가득한 단일 암호문을 만들 수 있다. 구체적으로는 ct_0부터 ct_{k-1}까지 포함된 총 n/s개의 데이터를 순서대로 이어 붙인 뒤, 이 배열을 s번 반복해 n개의 데이터를 채운다. $s > 1$인 경우 동일 데이터를 여러 번 복제해 두는 이유는 희소-슬롯 부트스트래핑 기법을 활용하기 위해서다. 알고리즘1은 이렇게 k개의 암호문을 하나로 합치는 $Merge(k,s)$ 절차를 설명하며, 여기서 ct_{zero}는 영벡터를 암호화한 암호문을 뜻한다.

여기서 주목해야 할 점은 기존 연구[4]에서 멀티플렉스-병렬 컨볼루션을 수행한 후의 암호문이 텐서 $A \in \mathbb{R}^{h_i \times w_i \times c_i}$에 $CptMultPack$ 함수를 적용하여 얻은 $CptMultPack(A) \in \mathbb{R}^n$ 꼴이며 이는 어떤 $n' | n$을 만족하는 n'에 대해 n'개의 데이터가 n/n'번 반복된 형태라는 점이다. 이는 알고리즘 1의 입력 암호문 형태, 즉, 반복이 안 된 형태와는 다르다. 알고리즘1에서의 입력 암호문은 n/ks개의 슬롯에만 $MultPack(A)$ 형태의 데이터가 있고 나머지 슬롯은 0으로 채워져 있다. 그러나 기존 연구에서의 멀티플렉스-병렬 컨볼루션 과정을 살펴보면 먼저 $MultPack(A)$ 꼴을 만들고 이를 회전 및 덧셈 연산을 사용하여 반복함으로써 $CptMultPack(A)$를 얻는다. 이러한 점을 고려하면, 마지막 회전/덧셈 연산 과정을 생략하고 $MultPack(A)$ 꼴을 얻은 상태에서 바로 $Merge(k,s)$ 알고리즘을 사용할 수 있다. 이후에 설명할 $Split(k,s)$ 알고리즘을 통해 다시 $CptMultPack(A)$ 꼴을 얻게 되므로, 후속 멀티플렉스-병렬 컨볼루션 수행에는 문제가 없다.

Input: Ciphertexts $ct_0, ct_1, \cdots, ct_{k-1}$
Output: Merged ciphertext ct_{merge}

1. $ct_{merge} \leftarrow ct_{zero}$
2. for $i = 0$ to $s-1$
3. for $j = 0$ to $k-1$
4. $temp \leftarrow Rot(ct_j; -n(ki+j)/ks)$
5. $ct_{merge} \leftarrow ct_{merge} \oplus temp$
6. Return ct_{merge}

알고리즘 1. $Merge(k,s)$ 알고리즘
Algorithm 1. $Merge(k,s)$ algorithm

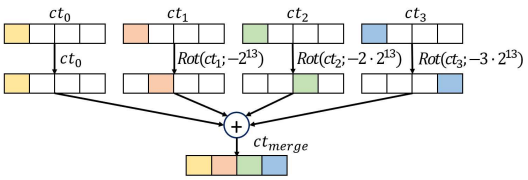


그림 1. $Merge(4,1)$ 알고리즘의 동작 과정
Fig 1. Process of $Merge(4,1)$ algorithm

그림 1은 $k=4$인 경우 $Merge(4,1)$ 의 동작과정을 시각화하며, 각 암호문들을 복호화했을 때 얻을 수 있는 데이터를 표현하고 있다. 이 그림에서 색칠된 사각형은 데이터가 존재하는 부분을 나타내며, 이는 $2^{15}/4 = 2^{13}$ 개의 숫자로 구성되어 있다. 반면, 흰색 사각형은 0으로 전부 채워진 부분을 나타낸다. 4개의 암호문에 각각 적절히 회전 연산 Rot을 적용하여 회전시킨 후 합치면, 모든 슬롯에 데이터가 존재하는 병합된 암호문 ct_{merge}를 얻을 수 있다.

이렇게 4개의 암호문을 병합한 후 부트스트래핑을 수행하면, 단 한 번의 전체-슬롯 부트스트래핑만으로 충분하다. 이를 "병합 부트스트래핑"이라 부르며, 이는 연산 효율성을 크게 향상시킨다. 이와 대조적으로, 병합을 하지 않는 경우, 총 4번의 희소-슬롯 부트스트래핑이 필요하게 된다. 희소-슬롯 부트스트래핑은 전체-슬롯 부트스트래핑에 비해 시간이 약간 덜 소요되지만, 이를 4번 반복하게 되면 전체-슬롯 부트스트래핑보다 많은 시간이 소요된다.

3.2 분리 알고리즘

$Merge(k,s)$ 알고리즘으로 k개의 암호문을 하나로 합친 뒤 부트스트래핑을 적용하면, 본래 k번 수행해야 했던 (희소-슬롯) 부트스트래핑을 단 한 번으로 줄일 수 있다. 다만 합친 암호문에 부트스트래핑을 끝낸 뒤에는, 이후 멀티플레스 병렬 컨볼루션을 위해 다시 여러 암호문으로 분리하는 과정이 필요하다.

가령 n/ks개의 슬롯에만 데이터가 있고 나머지는 0으로 채워진 k개의 암호문을 합쳐 암호문 ct를 얻었다고 하자. ct는 n개의 슬롯 모두에 데이터가 존재하며, 원래 k개 암호문에 있던 값이 슬롯마다 s회씩 반복 저장돼 있다. 이제 ct를 n/ks개의 데이터를 ks회 반복한 형태로 구성된 k개의 암호문 $ct_{split}^{(0)}, ct_{split}^{(1)}, \cdots, ct_{split}^{(k-1)}$로 되돌리고자 한다. 알고리즘2는 이와 같이 병합된 ct를 다시 k개의 암호문으로 분할하는 $Split(k,s)$ 절차를 설명한다.

알고리즘 2에서 $S_{k,s,i} \in \mathbb{R}^n$는 원소가 1과 0으로만 이루어진 일차원 벡터이다. 각각의 $0 \le j < s$에 대해 $[(kj+i)n/ks, (kj+i+1)n/ks - 1)$ 범위 안에 있는 인덱스 l에 대해 $S_{k,s,i}[l] = 1$이 되며, 그 외의 l에 대해서는 $S_{k,s,i}[l] = 0$이다. 이 $S_{k,s,i}$는 모든 슬롯에 데이터가 들어있는 암호문에서 선택된 데이터에만 1을 곱하고 나머지 데이터에는 0을 곱하여 제거하는 기능을 수행한다.

그림 2는 $Split(4,1)$의 동작과정을 시각적으로 보여준다. 4개의 암호문이 병합된 암호문 ct가 주어져있을 때 $S_{k,s,i}$ 벡터를 곱하여 원하는 부분만 추출하고 나머지 값들은 0으로 만든다. 그 후 동형 회전 및 덧셈 연산을 사용하여 동일한 데이터가 4번 반복되어 모든 슬롯에 데이터가 차게 한다. 이는 어떤 텐서 $A \in \mathbb{R}^{h_i \times w_i \times c_i}$에 $CptMultPack$ 함수를 적용하여 얻은 $CptMultPack$

Input: Ciphertexts ct
Output: Splitted ciphertexts $ct_{split}^{(0)}, ct_{split}^{(1)}, \cdots, ct_{split}^{(k-1)}$

1. for $i = 0$ to $k-1$
2. $sum \leftarrow ct_{zero}$
3. $temp \leftarrow ct \odot S_{k,s,i}$
4. for $j = 0$ to $k-1$
5. $sum \leftarrow sum \oplus Rot(temp; jn/ks)$
6. $ct_{split}^{(i)} \leftarrow sum$

알고리즘 2. $Split(k,s)$ 알고리즘
Algorithm 2. $Split(k,s)$ algorithm

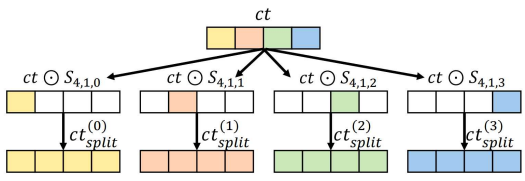


그림 2. $Split(4,1)$ 알고리즘의 동작 과정
Fig. 2. Process of $Split(4,1)$ algorithm

$(A)\in\mathbb{R}^n$ 꼴이며, 이는 이후 멀티플렉스 병렬 컨볼루션의 수행을 가능하게 한다.

3.3 병합 부트스트래핑을 사용한 ResNet-20

기존 연구[4]에서는 RNS-CKKS 상에서 ResNet이 CIFAR-10 이미지들을 독립적으로 분류하는 방식을 구현하였다. ResNet을 수행하는 과정에서 암호문에 포함된 데이터의 수는 $2^{14}\rightarrow 2^{13}\rightarrow 2^{12}$ 순서대로 감소하며, ResNet-20의 경우 각 데이터의 수에 대해 암호문에 대한 부트스트래핑을 6번씩 수행한다. 즉, $n_s=2^{12},2^{13},2^{14}$ 각각에 대해 6번의 희소-슬롯 부트스트래핑을 수행한다.

이 때, 병합 부트스트래핑을 활용하여 2개의 CIFAR-10 이미지를 동시에 분류하는 경우를 고려해보자. $n_s=2^{14},2^{13},2^{12}$ 만큼의 데이터가 포함되어 있는 두 암호문(서로 다른 이미지에 대한)을 각각 $Merge(2,1)$, $Merge(2,2)$, $Merge(2,4)$ 알고리즘을 통해 병합한다. 이렇게 하면 병합된 암호문은 각각 $n_s=2^{15},2^{14},2^{13}$개의 데이터가 1,2,4번씩 반복된 형태가 될 것이다. 즉, $n_s=2^{13},2^{14}$ 각각에 대해 6번의 희소-슬롯 부트스트래핑 및 $n_s=2^{15}$에 대한 6번의 전체-슬롯 부트스트래핑을 수행하면 된다. 그 후에는 $Split(2,1)$, $Split(2,2)$, $Split(2,4)$ 알고리즘을 사용하여 병합된 암호문을 분리하면, 이후에 후속 ResNet 레이어를 계속 수행할 수 있다.

4개의 CIFAR-10 이미지를 동시에 분류하는 경우를 생각해보자. 데이터 크기가 $n_s=2^{13},2^{12}$인 경우 네 이미지에 대한 암호문을 $Merge(4,1)$, $Merge(4,2)$ 알고리즘을 사용하여 병합하면 된다. 그런데 $n_s=2^{14}$인 경우 최대 2개의 암호문만 $Merge(2,1)$알고리즘을 사용하여 병합할 수 있다. 따라서, 총 $n_s=2^{14}$에 대한 희소-슬롯 부트스트래핑 6번, $n_s=2^{15}$에 대한 전체-슬롯 부트스트래핑 18번이 필요하게 된다.

마지막으로 8개의 CIFAR-10 이미지를 동시에 분류하는 경우를 살펴보자. 데이터 크기가 $n_s=2^{12}$인 경우 $Merge(8,1)$ 알고리즘을 이용하여 암호문을 병합하면 된다. $n_s=2^{13}$인 경우 최대 4개의 암호문을 $Merge(4,1)$ 알고리즘을 통해 병합할 수 있으며, $n_s=2^{14}$인 경우 최대 2개의 암호문을 $Merge(2,1)$ 알고리즘을 통해 병합할 수 있다. 결과적으로 총 42번의 전체-슬롯 부트스트래핑이 필요하게 된다.

이렇게 8개의 이미지가 병합되는 것이 평균 부트스트래핑 시간을 줄일 수 있는 최대 암호문 개수라 할 수 있다. 이는 ResNet에서 데이터 양이 가장 적은 경우인 $n_s=2^{12}$에서 최대 8개의 암호문을 병합할 수 있기 때문이다. 만약 데이터가 2^{12}보다 더 적은 다른 신경망을 사용하는 경우, 8보다 큰 k에 대하여 $Merge(k,s)$ 및 $Split(k,s)$ 알고리즘을 적용할 수 있을 것이다.

Ⅳ. 실험 결과

이 장에서는 병합 부트스트래핑을 활용한 RNS-CKKS 기반의 ResNet-20 성능에 대한 실험 결과를 제시한다.

4.1 실험 환경

이 실험은 유명한 오픈소스 RNS-CKKS 스킴 라이브러리인 Lattigo[9]를 기반으로 진행되었다. 실험에 사용된 컴퓨터는 2.096 GHz의 AMD Ryzen Threadripper PRO 3995WX 프로세서와 512 GB의 RAM을 탑재하였으며, Ubuntu 22.04를 운영체제로 사용한다. 본 실험에서는 1개의 CPU 코어를 사용한다.

4.2 파라미터

이 실험에서는 기존 연구[4]에서 사용한 RNS-CKKS 기반의 ResNet-20 구현을 대상으로 하므로 비슷한 RNS-CKKS 파라미터를 채택하였다. 먼저, 다항식 차수 파라미터는 $N=2^{16}$로 설정하였고, 따라서 암호화된 벡터의 성분 개수는 $n=2^{15}$가 된다.

RNS-CKKS 스킴의 비밀키의 해밍 무게(Hamming weight)는 192로 설정하였다. 특수 모듈러스(special modulus)와 기초 모듈러스(base modulus)로는 51 비트 소수를 사용하였으며, 그 외의 모듈러스로는 46 비트 소수를 사용하였다. 본 논문에서는 연산 속도를 향상시키기 위해 기존 연구[4]에서 사용한 1개 대신 2개의 특수 모듈러스를 사용하였다[13]. 기존 연구에서 사용한 SEAL 라이브러리는 여러 개의 특수 모듈러스 사용 기법을 지원하지 않아, 당시에는 1개의 특수 모듈러스만 사용할 수 있었다. 스케일링 팩터(scaling factor)는 2^{46}으로 설정하였다.

4.3 희소-슬롯 부트스트래핑 시간

기존 연구[4]에서 RNS-CKKS 상에서 ResNet을 수행하는데 필요한 희소-슬롯 부트스트래핑에서의 슬롯 개수 n_s은 $2^{12},2^{13},2^{14}$로 설정되었다. 본 논문에서 제안하는 병합 부트스트래핑을 사용할 경우에는 $n_s=2^{15}$ 의

경우도 필요하게 된다. 결과적으로, 총 $n_s = 2^{12}, 2^{13}, 2^{14}, 2^{15}$ 슬롯에 대한 부트스트래핑을 수행하게 되며 이에 대한 부트스트래핑 시간을 비교하고자 한다. 이 때, 병합 부트스트래핑을 사용하는 경우 레벨을 하나 더 필요로 하게 된다. 구체적으로 $Split(k,s)$ 알고리즘에서 상수벡터와의 곱셈에서 레벨을 하나 추가로 소모하며, 기존 연구[4]에서는 레벨 30 (부트스트래핑을 위한 레벨 포함)을 사용하였는데 병합 부트스트래핑을 쓰면 레벨 31을 사용하게 된다. 병합 부트스트래핑 사용으로 인한 레벨 추가는 약간의 부트스트래핑 시간 증가로 이어지지만 그럼에도 필요한 부트스트래핑 횟수가 줄어들기 때문에 결과적으로 평균 부트스트래핑 시간을 줄일 수 있다.

표 1는 슬롯 개수 및 레벨에 따른 부트스트래핑 시간을 나타낸다. 표 1의 결과는 4.4절에서 병합 부트스트래핑의 수행 시간을 구하기 위해 사용된다.

표 1. 슬롯 개수 및 레벨에 따른 (희소-슬롯) 부트스트래핑 시간
Table 1. Runtime of (Sparse-slot) bootstrapping depending on the number of slots and level

#slots(n_s)		2^{12}	2^{13}	2^{14}	2^{15}
(sparse-slot) bootstrapping runtime (s)	level 30	36.1	38.9	38.9	47.1
	level 31	39.5	42.2	42.2	53.9

4.4 병합 부트스트래핑을 사용한 ResNet-20 수행 시간

표 2는 ResNet-20을 이용한 CIFAR-10 이미지 분류에 소요되는 총 부트스트래핑 시간을 보여준다. 이는 기존 방법을 사용하여 한 번에 1개의 이미지를 분류하는 경우와, 본 연구에서 제안하는 병합 부트스트래핑 기법을 사용하여 한 번에 2개, 4개, 8개의 이미지를 동시에 분류하는 경우를 비교한다. 실험 결과, 병합 부트스트래핑 기법을 통해 2개, 4개, 8개의 이미지를 동시에 ResNet-20을 이용하여 분류하는 경우, 기존 방법에 비해 부트스트래핑의 분할 상환 시간이 각각 39%, 55%, 59% 줄어든 것을 확인할 수 있다.

표 2. 병합 부트스트래핑을 활용한 ResNet-20 실행에 대한 총 부트스트래핑 시간
Table 2. Total bootstrapping time for ResNet-20 execution utilizing merged bootstrapping

	previous	proposed		
	1 image	2 images	4 images	8 images
runtime (s)	683	830	1223	2264
amortized runtime (s)	**683**	**415**	**306**	**283**

또한, 본 연구에서는 2개의 이미지를 병합 부트스트래핑을 사용하여 동시에 분류하는 경우를 실제로 구현하고 각 구성요소 별 수행시간을 분석한다. 그림 3은 병합 부트스트래핑 기법을 활용하여 ResNet-20으로 두 이미지를 동시에 분류하는 아키텍처를 보여준다.

여기서 컨볼루션과 배치 정규화(batch normalization)를 합한 알고리즘인 ConvBN'는 기존 연구[4]의 알고리즘을 거의 그대로 사용하되 마지막 부분에서 동일한 데이터를 여러 번 반복하여 나열하는 부분만 생략한 것이며 구분을 위해 프라임(') 기호를 사용하였다. 다만, 가장 먼저 사용되는 ConvBN 알고리즘은 여러 번 반복하여 나열하는 부분까지 그대로 포함한다. 구체적인 ConvBN 알고리즘이 필요하면 기존 논문[4]의 ConvBN 알고리즘과 공개 소스코드를 참고할 수 있다.

알고리즘 3은 위 아키텍처에서 사용되는 $Select(s)$ 알고리즘을 보여준다.

먼저 $S'^{(s)} \in \mathbb{R}^n$을 다음과 같이 정의한다.

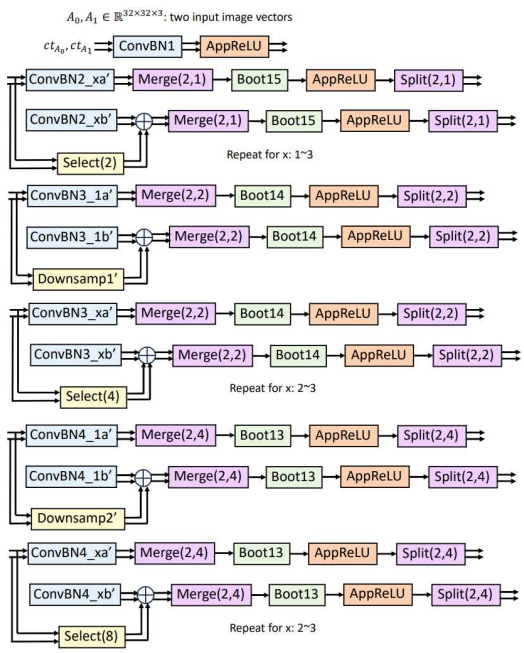


그림 3. 병합 부트스트래핑 기법을 활용하여 ResNet-20으로 두 이미지를 동시에 분류하는 아키텍처
Fig. 3. Architecture for simultaneous classification of two images using ResNet-20 with merged bootstrapping technique

Input: Ciphertexts ct
Output: Splitted ciphertexts $ct_{split}^{(0)}, ct_{split}^{(1)}, \cdots, ct_{split}^{(k-1)}$

1. for $i=0$ to $k-1$
2. $\quad sum \leftarrow ct_{zero}$
3. $\quad temp \leftarrow ct \odot S_{k,s,j}$
4. $\quad$ for $j=0$ to $k-1$
5. $\quad\quad sum \leftarrow sum \oplus Rot(temp; jn/ks)$
6. $\quad ct_{split}^{(i)} \leftarrow sum$

알고리즘 3. $Select(s)$ 알고리즘
Algorithm 3. $Select(s)$ algorithm

$$S'^{(s)}(i) = \begin{cases} 1 & 0 \le i < n/s \\ 0 & otherwise. \end{cases}$$

이 때, $Select(s)$은 단순히 입력 암호문 ct에 대해 $ct \odot S'^{(s)}$를 반환한다. 이는 동일한 데이터가 s번 반복되어있는 형태인 입력 암호문에서 첫 n/s개의 데이터만 남겨두고 나머지는 0으로 만듦으로써 반복을 없애는 역할을 한다.

표 3은 그림 3의 아키텍처를 사용하여 병합 부트스트래핑을 활용한 ResNet-20에 의한 두 이미지 분류를 수행할 때의 수행 시간을 나타낸다.

2개의 이미지를 동시에 분류할 때 컨볼루션, ReLU, 다운샘플링, 평균-풀링(average-pooling), 완전연결 레이어(fully-connected layer)의 경우 더 많은 횟수 수행이 필요하므로 시간이 더 오래 걸린다. 부트스트래핑의 경우 횟수 자체는 동일하지만 파라미터의 레벨이 하나 더 크고 슬롯 개수를 더 많이 필요로 하므로 (희소-슬롯) 부트스트래핑 시간이 증가한다. 따라서 전체 수행 시간은 증가하지만 분할 상환 시간을 계산하기 위해 2로 나누어야 한다. 그 결과 분할 상환 시간은 기존의 888s에서 558s로 감소하게 되며 이는 37% 감소에 해당한다.

표 3. 병합 부트스트래핑을 활용한 ResNet-20에 의한 두 이미지 분류의 수행 시간
Table 3. Runtime for classifying two images using ResNet-20 with merged bootstrapping

	previous (one image)	proposed (two images)	
	runtime (amortized runtime) (s)	runtime (s)	amortized runtime (s)
ConvBN	59	119	59
Boot	684	827	414
ReLU	143	162	81
Merge	-	0.63	0.31
Split	-	1.56	0.78
Downsamp	1.07	1.83	0.91
Avg-Pool	0.57	1.13	0.57
FC-layer	1.37	2.74	1.37
total	**888**	**1115**	**558**

4.5 병합·분리 알고리즘의 정확도·수행시간 영향과 확장성 분석

RNS-CKKS 기반의 ResNet-20 추론 과정에서는, 일반적인(평문 상의) ResNet-20 대비 다소의 정확도 하락이 발생할 수 있다. 이러한 정확도 저하는 주로 동형 덧셈, 스칼라 곱셈, 회전 등의 RNS-CKKS의 근사 연산, 부트스트래핑, 그리고 근사 ReLU에 기인한다. 그러나 충분히 큰 스케일링 팩터를 사용하는 경우, RNS-CKKS 연산 자체의 오차는 상대적으로 무시할 수 있을 정도이며, 전체 정확도 저하는 대부분 부트스트래핑과 근사 ReLU의 근사화에 의해 결정된다. 실제로 본 연구에서 사용한 환경에서는, 부트스트래핑과 근사 ReLU의 평균 오차가 약 $2^{-17} \sim 2^{-15}$ 수준인 반면, 일반적인 RNS-CKKS 연산의 평균 오차는 $2^{-36} \sim 2^{-35}$ 수준으로 측정되어, 후자의 영향은 매우 작다.

제안하는 병합 부트스트래핑 기법은 기존의 RNS-CKKS 기반 ResNet-20 추론 구조를 그대로 유지하면서, 병합 및 분리 알고리즘을 추가한 형태이다. 이 알고리즘들은 동형 덧셈, 스칼라 곱셈, 회전 연산만을 사용하며, 이들 연산의 오차는 앞서 설명한 바와 같이 전체 정확도에 거의 영향을 미치지 않을 것으로 예상된다.

또한 병합·분리 알고리즘은 매우 낮은 연산 레벨(레벨 1~3)에서 수행되며, 연산 복잡도 역시 낮기 때문에 전체 추론 시간에 미치는 영향이 제한적이다. 실제 표 3에 따르면 병합 알고리즘과 분리 알고리즘의 전체 수행시간은 0.63초, 1.56초로 부트스트래핑 연산 827초에 비해 매우 미미한 수준이다. 다만, 분리 연산은 추가적인 레벨 소모로 인해 부트스트래핑 수행 시간이 소폭 증가하며, 이는 표1, 2를 통해 확인할 수 있다. 그러나 이러한 오버헤드는 부트스트래핑 횟수 감소 효과에 비해 작기 때문에, 결과적으로 분할 상환 시간이 감소하는 이점이 있다. 추가적인 레벨 소모를 최소화하기 위해 향후 분리 알고리즘을 컨볼루션 등의 연산과 적절히 통합하여 레벨을 공유하는 전략은 향후 연구 과제로 고려될 수 있다.

제안된 병합·분리 알고리즘은 부트스트래핑 사용으로 인해 슬롯 크기는 크지만 실제 데이터가 적어 슬롯이

비효율적으로 사용되는 상황에 효과적이다. 특히 한 클라이언트가 여러 이미지의 추론을 요청할 때 유용하다. 이 알고리즘은 FHE 기반 ResNet-20과 유사한 조건을 가진 상황은 물론, 컨볼루션 신경망 외의 다른 종류의 모델에도 활용될 수 있을 것으로 예상된다.

본 연구에서는 RNS-CKKS를 중심으로 분석하였으나, 제안한 알고리즘은 벡터 패킹 및 슬롯 회전을 지원하는 Brakerski-Gentry- Vaikuntanathan(BGV)[14], Brakerski- Fan-Vercauteren(BFV)[15] 등 다른 Ring Learning With Errors(RLWE) 기반 FHE 스킴에도 적용 가능하다. 또한, 본 기술은 순수 FHE만을 사용하는 환경에 적합하며, FHE와 다자간 계산을 결합한 하이브리드 접근법과 비교하면 수행 시간이 상대적으로 길지만 통신량이 적다는 장점이 있다.

V. 결 론

본 연구는 암호문의 슬롯을 모두 활용해 여러 이미지를 동시 처리하도록 ResNet을 실행함으로써 부트스트래핑 시간의 대폭 절감을 실현하는 방안을 제시한다. 제안한 병합 - 분리 절차를 적용하면, 원래 각 암호문마다 수행해야 했던 부트스트래핑을 단 한 번의 작업으로 대체할 수 있다. 이 방법을 사용하여 2개, 4개, 8개의 CIFAR-10 이미지를 동시에 ResNet-20으로 분류한 경우, 부트스트래핑의 분할 상환 시간(amortized runtime; 이미지 당 시간)이 기존 방법에 비해 각각 39%, 55%, 59% 줄어들었다. 또한, 2개의 이미지를 병합 부트스트래핑을 사용하여 동시에 ResNet-20으로 분류하는 경우를 전체를 구현한 결과 분할 상환 시간이 기존 연구에 비해 37% 감소한 것을 확인하였다. 이 결과는 본 연구에서 제안한 방법이 암호화된 딥러닝 연산의 효율성을 크게 향상시킬 수 있음을 보여준다.

제안한 병합-분리 알고리즘은 정확도에 미치는 영향이 거의 없다. 다만, 이 알고리즘은 추가적인 암호문 레벨을 하나 소모하여 부트스트래핑 수행 시간이 소폭 증가하지만, 부트스트래핑 횟수 감소로 인해 전체적인 분할 상환 시간은 감소한다. 향후 연구에서는 레벨 소모에 따른 오버헤드까지 제거할 수 있는 방안을 모색할 필요가 있다.

본 연구에서 제안한 기법은 부트스트래핑을 사용해야 하는 구조로 인해 슬롯은 충분하지만 실제 데이터는 적어 슬롯을 다 활용하지 못하는 비효율적인 상황에 특히 효과적이다. 이러한 상황을 가지는 다양한 암호화 딥러닝 응용에 적용 가능하며, RNS-CKKS 뿐만 아니라 BGV, BFV 등 RLWE 기반의 다른 완전동형암호 스킴에도 유사한 방식으로 적용할 수 있을 것으로 기대된다.

References

[1] C. Gentry, “Fully homomorphic encryption using ideal lattices,” in *Proc. Forty-First Annual ACM Symp. Theory of Computing 2009*, pp. 169-178, Bethesda, MD, USA, May 2009. (https://doi.org/10.1145/1536414.1536440)

[2] H. Oh, “Recent studies on privacy-preserving machine learning using homomorphic encryption,” *J. KIISE*, vol. 41, no. 1, pp. 21-29, 2023.

[3] J. Lee, E. Lee, J. Lee, Y. Kim, Y. Kim, and J. No, “Precise approximation of convolutional neural networks for homomorphically encrypted data.”, *J. IEEE Access,* Vol 11, pp. 62062-62076, 2023. (https://doi.org/10.1109/ACCESS.2023.3287564)

[4] E. Lee, J. Lee, J. Lee, Y. Kim, Y. Kim, J. No, and W. Choi, “Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions,” in *Proc. ICML 2022*, *PMLR*, pp. 12403-12422, Baltimore, MD, USA, Jul. 2022.

[5] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz, “CHET: An optimizing compiler for fully-homomorphic neural-network inferencing,” in *Proc. 40th ACM SIGPLAN Conf. Programming Language Design and Implementation 2019*, pp. 142-156, Phoenix, AZ, USA, Jun. 2019. (https://doi.org/10.1145/3314221.3314628)

[6] D. Kim and C. Guyot, “Optimized privacy-preserving CNN inference with fully homomorphic encryption,” *IEEE Trans. Inf. Forensics and Secur.*, vol. 18, pp. 2175-2187, 2023. (https://doi.org/10.1109/TIFS.2023.3263631)

[7] J. Cheon, A. Kim, M. Kim, and Y. Song, “Homomorphic encryption for arithmetic of

approximate numbers," in *Proc. ASIACRYPT 2017, LNCS*, pp. 409-437, Hong Kong, China, Dec. 2017.

[8] J. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Proc. Int. Conf. Selected Areas in Cryptography 2018, LNCS*, pp. 347-368, Calgary, Alberta, Canada, Aug. 2018. (https://doi.org/10.1007/978-3-030-10970-7_16)

[9] Lattigo v3, Online: https://github.com/tuneinsight/lattigo, Aug. 2022, EPFL-LDS, Tune Insight SA.

[10] J. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for approximate homomorphic encryption," in *Proc. EUROCRYPT 2018, LNCS*, pp. 360-384, Tel Aviv, Israel, Apr. 2018. (https://doi.org/10.1007/978-3-319-78381-9_14)

[11] E. Lee, J. Lee, J. No, and Y. Kim, "Minimax approximation of sign function by composite polynomial for homomorphic comparison," *IEEE Trans. Dependable and Secure Computing*, vol. 19, no. 6, pp. 3711-3727, 2021. (https://doi.org/10.1109/TDSC.2021.3105111)

[12] E. Lee, J. Lee, Y. Kim, and J. No, "Optimization of homomorphic comparison algorithm on RNS-CKKS scheme," *IEEE Access*, vol. 10, pp. 26163-26176, 2022. (https://doi.org/10.1109/ACCESS.2022.3155882)

[13] K. Han and D. Ki, "Better bootstrapping for approximate homomorphic encryption," in *Proc. Cryptographers' Track at the RSA Conf. 2020*, pp. 364-390, San Francisco, CA, USA, Feb. 2020. (https://doi.org/10.1007/978-3-030-40186-3_16)

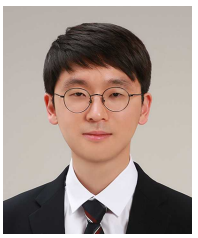
[14] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Computation Theory*, vol. 6, no. 3, pp. 1-36, 2014. (https://doi.org/10.1145/2633600)

[15] J. Fan and F. Vercauteren, "*Somewhat practical fully homomorphic encryption* (2012)," Retrieved Jun, 30, 2024, from https://eprint.iacr.org/2012/144

박 수 민 (Soomin Park)

2025년 2월 : 세종대학교 소프트웨어학과 학사 졸업
2025년 3월~현재 : 중앙대학교 컴퓨터공학과 석사
<관심분야> 암호, 프라이버시-보호 머신러닝, 인공지능
[ORCID:0009-0008-8543-273X]

이 은 상 (Eunsang Lee)



2014월 8월 : 서울대학교 전기정보공학부 졸업
2020년 8월 : 서울대학교 전기·정보공학부 석박사통합과정 졸업
2020년 9월~2022년 8월 : 서울대학교 전기·정보공학부 박사후연구원
2022년 9월~현재 : 세종대학교 소프트웨어학과 교수
<관심분야> 암호, 프라이버시-보호 머신러닝, 포스트 양자 암호
[ORCID:0000-0002-5270-2405]