
# FHEFusion: Enabling Operator Fusion in FHE Compilers for Depth-Efficient DNN Inference

Tianxiang Sui
*Ant Research*
*Ant Group*
Shanghai, China
suitianxiang.stx@antgroup.com

JianXin Lai
*Ant Research*
*Ant Group*
Shanghai, China
laijianxin.ljx@antgroup.com

Long Li
*Ant Research*
*Ant Group*
Shanghai, China
ll398708@antgroup.com

Peng Yuan
*Ant Research*
*Ant Group*
Shanghai, China
yp398707@antgroup.com

Yan Liu
*Ant Research*
*Ant Group*
Shanghai, China
ly409648@antgroup.com

Qing Zhu
*Ant Research*
*Ant Group*
Shanghai, China
zq398709@antgroup.com

Xiaojing Zhang
*Ant Research*
*Ant Group*
Shanghai, China
zxj398711@antgroup.com

Linjie Xiao
*Ant Research*
*Ant Group*
Shanghai, China
xiaolinjie.xlj@antgroup.com

Mingzhe Zhang
*Ant Research*
*Ant Group*
Beijing, China
huayi.zmz@antgroup.com

Jingling Xue
*School of Computer Science and Engineering, Ant Research*
*UNSW, Ant Group*
Sydney, Australia
j.xue@unsw.edu.au

***Abstract*—Operator fusion is essential for accelerating FHE-based DNN inference because it reduces multiplicative depth and, in turn, lowers the cost of ciphertext operations by keeping them at lower ciphertext levels. Existing approaches either rely on manual optimizations, which miss cross-operator opportunities, or on compiler pattern matching, which lacks generality. Standard DNN graphs omit FHE-specific behaviors, while fully lowering to primitive FHE operations introduces excessive granularity and obstructs effective optimization.**

**We present FHEFUSION, a compiler framework for the CKKS scheme that enables fusion through a new IR. This IR preserves high-level DNN semantics while introducing FHE-aware operators—masking and compaction (Strided_Slice)—that are central to CKKS, thereby exposing broader fusion opportunities. Guided by algebraic rules and an FHE-aware cost model, FHEFUSION reduces multiplicative depth and identifies profitable fusions. Integrated into ANT-ACE, a state-of-the-art FHE compiler, FHEFUSION outperforms NGRAPH, the only framework with graph-level fusion, achieving up to $3.02\times$ (average $1.40\times$) speedup across seven DNNs (13 variants from different RELU approximations) on CPUs, while maintaining inference accuracy.**

***Index Terms*—FHE Compilation, Operator Fusion, CKKS**

## I. INTRODUCTION

Operator fusion is widely used for improving DNN execution efficiency [1]–[3]. Popular frameworks such as TensorFlow [1], TVM [2], and PyTorch [4] support fusion based on the *computational graph* [2], which models a DNN as tensor operations with dependency edges. By exploiting algebraic rules among operators, these transformations reduce computation and intermediate operations to enhance performance.

Beyond efficiency, deep learning also raises privacy concerns on untrusted platforms. Fully homomorphic encryption (FHE) [5] enables inference directly on encrypted data, allowing secure outsourcing of storage and computation. Among existing FHE schemes, CKKS [6] is the most practical for encrypted DNN inference due to its support for fixed-point arithmetic and SIMD parallelism.

However, FHE schemes (including CKKS) introduce unique constraints—such as ciphertext slot structure, multiplicative depth limits [7], and expensive homomorphic operations like rotations and bootstrapping—that make operator fusion fundamentally different from conventional DNNs. Despite strong privacy guarantees, FHE-based inference remains orders of magnitude slower (often over $10{,}000\times$ on CPUs) than plaintext computation, posing serious performance challenges [8].

The performance of FHE programs, including CKKS, is highly sensitive to multiplicative depth. As ciphertexts move to lower levels, homomorphic operations become cheaper; however, each homomorphic multiplication consumes one ciphertext level, and when levels are exhausted, expensive bootstrapping must be invoked to refresh them [9]. The cost of bootstrapping itself depends on the level to which a ciphertext is restored, and bootstrapping typically dominates runtime [8]. Consequently, reducing multiplicative depth—thereby both lowering per-operation cost and reducing the number and cost of bootstraps—is often more important than micro-optimizing individual ciphertext operations. This makes operator fusion,

This work was supported by National Key R&D Program of China(Grant No. 2023YFB4503204).

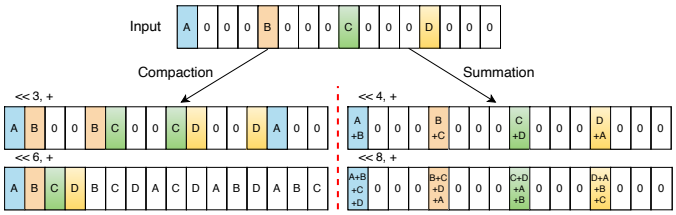


Fig. 1. Equivalence of compaction and summation in CKKS.

which directly reduces multiplicative depth, a key mechanism for accelerating encrypted inference.

**Problem Statement.** We aim to design a graph-level operator fusion framework for FHE compilers that accelerates encrypted DNN inference under CKKS. Because fusion directly reduces multiplicative depth—thereby lowering both per-operation cost and the number and cost of bootstraps—the framework must identify and apply profitable fusions before lowering to CKKS operations, enabling the compiler to generate more efficient encrypted code.

**Challenges.** In FHE, including CKKS, operator fusion must focus on reducing multiplicative depth and expensive ciphertext operations—goals that differ fundamentally from fusion in traditional DNNs. For instance, rewriting $((a * x) * x) * x$ as $(a * x) * (x * x)$ reduces depth from 3 to 2. CKKS lowering also expands operators into many fine-grained loops, obscuring high-level structure. As shown in Figure 1, compaction and summation share identical CKKS rotations and additions except for rotation indices, making them indistinguishable at the CKKS level and complicating fusion.

The core challenge is representation. Conventional DNN graphs ignore FHE-specific characteristics, offering little guidance for fusion, while expanding into primitive CKKS operators is overly fine-grained and hinders analysis. This representation gap complicates optimization and limits the effectiveness of existing compiler techniques.

**Prior Work.** Efforts to improve FHE [7], [8], [10]–[15] have addressed programmability through compilers [7], [11], [13], [15], DSLs [12], and libraries [10], as well as optimizations for data layout [8], rescaling [14], noise management [16]–[18], and operator implementations [19]–[21]. Operator fusion for depth reduction remains limited. HELIB [10] and FHE-MP-CNN [19] apply manual fusion to core operators such as Conv, with a few cross-operator cases like Conv+BN [19]. The DSL compiler Fhelipe [8] fuses consecutive GEMVs, while NGRAPH [7], the only framework with graph-level fusion, applies three simple constant-folding patterns—AvgPool Folding, Activation Folding, and BatchNorm Folding, the last already standard in PyTorch [4].

When lowering DNN operators under CKKS, the limited operation set often leaves junk data or scatters valid data across slots [8], [12], [19], [22], requiring masking to discard irrelevant data [8], [10], [12], [19] and compaction to make valid data contiguous [8]. The DSL compiler Fhelipe [8] compacts by masking gaps with 0/1 vectors, adding extra multiplications and missing further masking optimizations. Overall, existing approaches rely on manual fusion, which overlooks many opportunities, or pattern matching, which lacks generality.

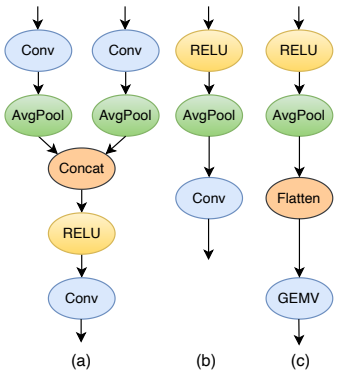


Fig. 2. Fusion cases: NGRAPH fuses (a,b) only, while FHEFUSION fuses all.

**This Work.** We present FHEFUSION, a graph-level fusion framework for FHE compilers that automates encrypted DNN inference under CKKS. It introduces FHE-aware operators (masking and compaction) alongside standard DNN operators in a new IR, exposing broader fusion opportunities through syntactic and semantic analysis. Guided by algebraic rules and an FHE-aware cost model, FHEFUSION reduces multiplicative depth and accelerates encrypted execution.

In developing FHEFUSION, we leverage three insights:

- Operators, such as Flatten, Reshape, and Concat, preserve data values, allowing scalar multiplications to propagate through them and expand fusion opportunities. As shown in Figure 2, NGRAPH [7] applies fusion only in two cases (Activation Folding in (a) and AvgPool Folding in (b)), whereas FHEFUSION handles all three. Leveraging algebraic rules rather than the three fixed patterns in NGRAPH, FHEFUSION enables additional fusions, such as RELU followed by AvgPool and Conv (Equation (9)).
- Gaps in ciphertexts typically arise from strides in DNN operators [8], [19], [22], [23]. This regularity allows stride information to propagate, enabling later operators to adjust parameters under algebraic transformations, as also noted and manually incorporated in ResNet20 [23]. For example, fusing Strided_Slice with Masking can remove an otherwise required compaction, allowing the fused output with gaps to be consumed directly by subsequent operations (Figure 4).
- CKKS-specific operations such as masking and Strided_Slice can be expressed at the same abstraction level as standard DNN operators, simplifying and broadening optimization opportunities.

Building on associative, commutative, and distributive laws, we develop algebraic rules and a simple FHE-aware cost model to guide a graph-level fusion algorithm for DNNs. It employs three strategies—constant folding, masking folding, and compaction folding (Strided_Slice). Fusion reduces multiplicative depth, lowering bootstrapping costs, while propagating and fusing Strided_Slice operations further cut CKKS

operations, especially rotations. Together, these extend the scope and effectiveness of compiler-driven fusion [7].

**Contributions.** This paper makes three contributions:

- **A New Framework.** We present FHEFUSION, the first graph-level framework to perform fusion on a dedicated IR for FHE compilers, introducing FHE-aware operators (masking and compaction) alongside standard DNN operators for encrypted DNN inference under CKKS.
- **A New Fusion Algorithm.** We design an operator fusion algorithm that applies algebraic rules, guided by an FHE-aware cost model, to reduce multiplicative depth and restructure graphs in ways that lessen costly CKKS operations (e.g., rotations and bootstrapping) once lowered.
- **A Comprehensive Evaluation.** We integrate FHEFUSION into ANT-ACE, a state-of-the-art open-source FHE compiler for DNNs, and show speedups up to $3.02\times$ (average $1.40\times$) over NGRAPH [7] across seven DNNs (13 variants from different RELU approximations) on CPUs.

While our presentation focuses on CKKS, FHEFUSION applies directly to other RLWE-based schemes [24], including BGV [25] and BFV [26]), since they share the same core RLWE operations (Section II-A).

## II. BACKGROUND

We next review the essential background for this work.

### A. CKKS

CKKS [6] is an FHE scheme that supports fixed-point arithmetic on encrypted complex numbers, with RNS-CKKS [27] as its most efficient variant. We briefly review its fundamentals to highlight operation costs and the role of multiplicative depth in performance. In RNS-CKKS, a ciphertext is a degree-$N$ polynomial with coefficients modulo $Q$, where $N$ is a power of two and $Q = \prod_{i=0}^{r} Q_i$ is the product of $r+1$ co-primes. Each ciphertext provides $N/2$ slots for encoding up to $N/2$ values, and $Q$ is chosen to match the maximum multiplicative depth of the program. The number of factors $r$ defines the ciphertext level, i.e., its capacity for homomorphic multiplications.

CKKS supports a limited set of homomorphic vector operations: additions and multiplications on plaintexts or ciphertexts (element-wise on the underlying vectors in SIMD fashion) and ciphertext rotations (slot-wise shifts within a ciphertext).

**Complexity of CKKS Operations.** Table I summarizes their complexity under RNS-CKKS. Larger $N$ and $Q$ (or $r$) increase operation cost, while smaller-$r$ ciphertexts are faster. Each multiplication followed by rescaling reduces the ciphertext level $r$ by one; once $r = 0$, no further multiplications are possible and bootstrapping [9], [28] is required.

**Multiplicative Depth.** CKKS ciphertexts accumulate noise with homomorphic operations, especially multiplications; excessive noise breaks correctness. Noise-management steps [8], [14], [17], [18], [29] reduce the ciphertext level $r$, limiting the remaining multiplicative budget. Bootstrapping restores levels but is the most expensive CKKS operation, and its cost increases with the level to which the ciphertext is restored. Moreover, bootstrapping itself consumes a fixed number of levels, so the recoverable depth is bounded by the remaining modulus. Reducing multiplicative depth therefore decreases both the number of required bootstraps and the cost of each one, making it crucial for CKKS performance.

TABLE I
COMPLEXITY OF RNS-CKKS OPERATIONS.

| Operator | Operands | Complexity |
|---|---|---|
| Addition ($+$) | ⟨Cipher, Plain⟩; ⟨Cipher, Cipher⟩ | $O(N \cdot r)$ |
| Multiplication ($\cdot$) | ⟨Cipher, Scalar⟩ | $O(N \cdot r)$ |
| | ⟨Cipher, Plain⟩ | $O(N \cdot r)$ |
| | ⟨Cipher, Cipher⟩ | $O(N \cdot \log N \cdot r^2)$ |
| Rotation ($\ll, \gg$) | ⟨Cipher, Integer⟩ | $O(N \cdot \log N \cdot r^2)$ |

For security, $N$ must be large (typically $\geq 16$K) and fixed during execution. Since unused ciphertext slots do not lower costs, applications must fill these vectors to avoid wasted work. Balancing security and performance by selecting $N$ and $Q$ and designing data layouts to fully utilize the $N/2$ slots for parallelism remains a core challenge in FHE programming.

### B. Implementing DNN Operators with CKKS

DNNs operate on tensors, but CKKS supports only ciphertext vectors. Input tensors must therefore be flattened into one-dimensional vectors under an appropriate data layout before encryption. Typically, only inputs are encrypted, while model weights remain plaintext constants known at compile time. Because ciphertext operations are costly, computation is offloaded to plaintext whenever possible, a heuristic that also exposes fusion opportunities. Moreover, the limited CKKS operations (Table I) mean that implementing DNN operators—especially reductions—often introduces junk values or empty slots in ciphertext vectors, as noted in prior work [8], [12], [19], [22].

We examine common DNN operators and show how their CKKS implementations reveal graph-level fusion opportunities, even though such information is hidden at the graph level.

**RELU.** Because CKKS [6] does not support comparisons, this operator is approximated by a polynomial:

$$\mathsf{RELU}(x) = \sum_{i=0}^{n} a_i x^i \tag{1}$$

where $a_i$ are scalars and $x$ denotes a slot value in the input ciphertext. Two approaches exist:

- **Sequential High-Degree Polynomials:** compose multiple high-degree polynomials to approximate RELU; preserves pretrained weights but incurs large multiplicative depth, often requiring a bootstrap per RELU [19], [23].
- **Single Low-Degree Polynomial:** approximate RELU with one low-degree polynomial (e.g., $ax^2 + bx + c$); requires retraining but consumes only a few levels (two for quadratic), typically avoiding bootstrapping and enabling faster inference [12], [30].

Such polynomial evaluations on a ciphertext can also be fused with adjacent operations on the same ciphertext, creating further optimization opportunities.

**Pooling.** In DNNs, pooling (e.g., AvgPool) with non-unity strides produces tensors with gaps, which map to ciphertexts with gaps. Managing these at the graph level creates fusion opportunities, particularly for AvgPool.

**Reshaping.** Operators such as Flatten, Reshape, and Concat change tensor shape but not data values. This allows constant multiplications to propagate across them without affecting correctness, enabling fusion in subsequent operations.

**Conv and GEMV.** Both take a ciphertext input $X$ (image for Conv and features for GEMV) and a plaintext input $W$ (kernels or weights), producing an output $Y$. Under CKKS, they are typically implemented as repeated ciphertext rotations with ciphertext–plaintext multiplications [8], [15]:

$$Y = \sum_i \big(\text{Rotation}(X, i) \times W_i\big) \tag{2}$$

where $W_i$ is a row/column of $W$. Since these are ciphertext–plaintext multiplications, nearby scalar multiplications on $X$ or $Y$ can be fused with $W$ at the graph level.

For Conv with stride $> 1$, $Y$ has non-contiguous slots. Standard implementations compact them, but as noted in [23], compaction can be skipped if downstream operators adapt to the altered layout, enabling further fusion.

## III. Motivation

We first illustrate how high-level DNN operators are lowered to FHE operations (Section III-A), highlight fusion opportunities, including cross-operator cases (Section III-B), and discuss two key challenges (Section III-C): (1) selecting among competing optimizations, and (2) loss of operator information when DNN operators are decomposed into low-level CKKS loops, which complicates both analysis and optimization. We conclude by outlining our solution (Section III-D).

### A. A Motivating Example

Figure 3 motivates FHEFusion using AvgPool with $N = 32$ slots ($N/2 = 16$ usable slots per ciphertext).

Figure 3a applies AvgPool to a $4 \times 2 \times 2$ input ciphertext tensor $A$ with a $2 \times 2$ kernel and stride 2, producing an output tensor of shape $4 \times 1$. The operation sums four elements per channel and scales by $1/4$ to yield the final 4-element output.

Figure 3b shows a CKKS implementation in four steps ❶–❹: $B = \mathsf{Compact}(\mathsf{Masking}(\mathsf{Averaging}(\mathsf{Roll\_Sum}(A))))$. First, the kernel sums are computed (❶) and then averaged by multiplying with $0.25$ (❷), producing a ciphertext with four valid values separated by gaps (#). Masking then clears the junk slots using a 0/1 plaintext (❸), and compaction packs the valid values into the first four slots (❹), yielding the encrypted equivalent of the result in Figure 3a.

### B. Optimization Opportunities

Examining ❶ and ❷ in Figure 3b shows that the two steps commute. Interchanging Roll_Sum and Averaging gives $B = \mathsf{Compact}(\mathsf{Masking}(\mathsf{Roll\_Sum}(\mathsf{Averaging}(A))))$, with Figure 3c illustrating the alternative order. As noted in Section II-A, multiplications consume depth and reduce ciphertext levels. Applying averaging (multiplication by $0.25$) first lowers the input ciphertext's level $r$ before the two rotations and additions in Roll_Sum. Since $N$ is fixed, reducing $r$ is the only way to improve performance (Table I).

In Figure 3b, Steps ❷ and ❸ apply successive multiplications—first by $0.25$, then by $[1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0]$—consuming two multiplicative levels. By associativity, the same result can be obtained with a single multiplication by $[0.25, 0, 0, 0, 0.25, 0, 0, 0, 0.25, 0, 0, 0, 0.25, 0, 0, 0]$ through fusing Averaging and Masking: $B = \mathsf{Compact}(\mathsf{Fused\_Masking}(\mathsf{Roll\_Sum}(A)))$, as shown in Figure 3d. This transformed AvgPool uses only one multiplicative level and one ciphertext-plaintext multiplication.

After Step ❹, the last 12 slots contain junk data. If later operations access them, masking with $[1, 1, 1, 1, 0, 0, \ldots, 0]$ (ones in the first four slots) may be required, which can also be fused with subsequent multiplications.

Figure 4 illustrates a cross-operator optimization. Suppose a preceding operator produces $A_{\text{gaps}} = [A1, \#, A2, \#, A3, \#, A4, \#]$, which would normally require masking and compaction to yield $A = [A1, A2, A3, A4]$ before a GEMV with weights $W \in \mathbb{R}^{4\times 2}$. Fusing masking, compaction, and GEMV yields $\mathsf{GEMV}_{\text{gaps}}$, which operates directly on $A_{\text{gaps}}$ with $W_{\text{zeros}}$ (weights with inserted zeros). This removes explicit masking and compaction, and lowers multiplicative depth.

### C. Optimization Challenges

Developing FHEFusion faces two main challenges. The first is the choice of IR. Standard DNN graphs omit FHE-specific operators such as masking and compaction, missing fusion opportunities, while fully lowering into CKKS operations yields overly fine granularity and obscures optimization. Once lowered, rich operator-level semantics are lost, leaving only loops and basic CKKS operations (addition, multiplication, and rotation). This obfuscation makes analysis and optimization considerably harder: for example, in Figure 3b, ❶ and ❹ reduce to additions and rotations, while ❷ and ❸ become scalar and vector multiplications. It then becomes difficult to tell whether a loop implements a Compact or Roll_Sum, or whether a multiplication clears junk data (using a Masking operation) or performs a standard arithmetic task.

Another challenge lies in selecting among alternative optimization strategies, which can be mutually exclusive. Figures 3c and 3d illustrate such cases, where the choice must be guided by an FHE-aware cost model.

### D. Our Solution

FHEFusion addresses these two challenges by performing fusion on a new IR that integrates DNN operators with FHE-aware operators, preserving high-level semantics while remaining sensitive to key CKKS characteristics during lowering. At this level, algebraic rules and an FHE-aware cost model jointly guide both the legality and profitability of fusion.

## IV. Design

We first examine the operators supported in FHEFusion—standard DNN operators augmented with high-level

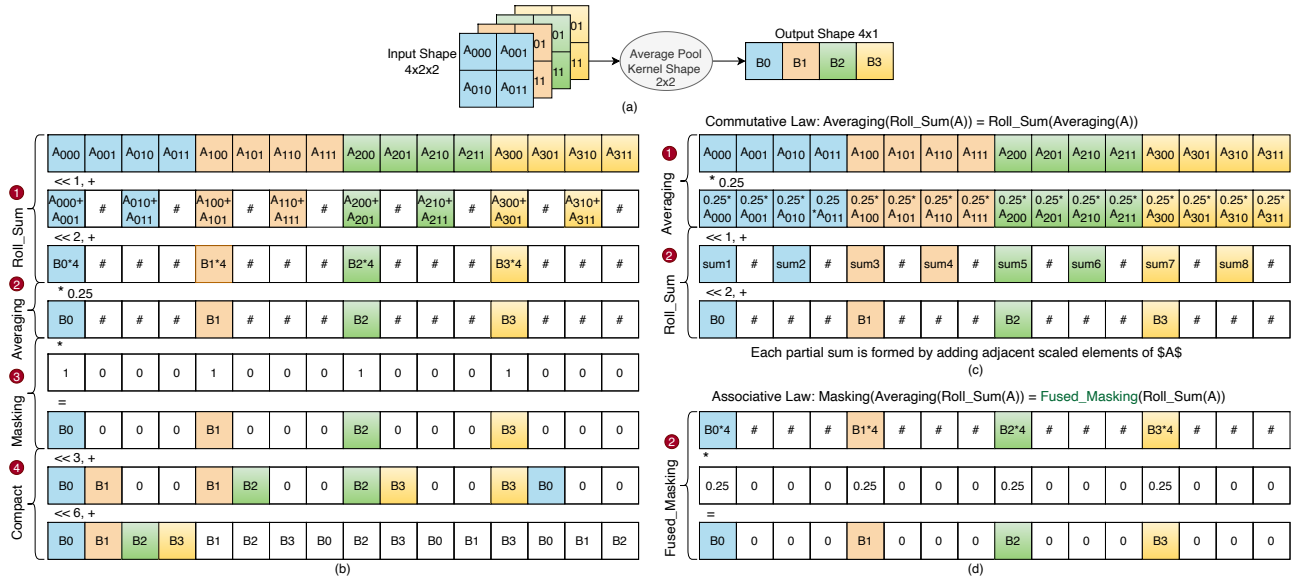


Fig. 3. Example of AvgPool with $N = 32$ ($N/2 = 16$ slots). (a) High-level operator. (b) CKKS implementation: input tensor $A$ is flattened, yielding $B = \mathsf{Compact}(\mathsf{Masking}(\mathsf{Averaging}(\mathsf{Roll\_Sum}(A))))$; # marks junk slots. (c) Interchanging Averaging and Roll_Sum via commutativity: $B = \mathsf{Compact}(\mathsf{Masking}(\mathsf{Roll\_Sum}(\mathsf{Averaging}(A))))$. (d) Fusing Averaging and Masking via associativity: $B = \mathsf{Compact}(\mathsf{Fused\_Masking}(\mathsf{Roll\_Sum}(A)))$.

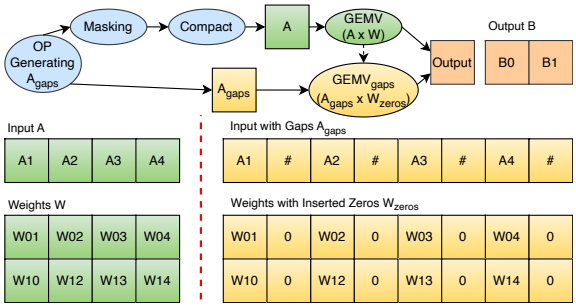


Fig. 4. Example of GEMV ($B = AW$) with gaps in the input: fusing masking, compaction, and GEMV into $\mathsf{GEMV}_{\text{gaps}}$, which operates directly on $A_{\text{gaps}}$ and $W_{\text{zeros}}$, eliminating explicit masking and compaction.

semantics and two FHE-aware operators for CKKS-specific computations (Section IV-A). We then present our IR (Section IV-B) and a set of representative algebraic rules that guide fusion optimizations (Section IV-C).

### A. FHE-Aware and Reexpressed DNN Operators

We introduce two FHE-aware operators—masking and compaction (via Strided_Slice)—and show how standard DNN operators can be reexpressed with them while preserving operator-level semantics. Fusion at this graph level exposes broader fusion opportunities while avoiding the complexity of analysis and optimization at the CKKS level (Figure 3).

*1) FHE-Aware Operators:* Currently, FHEFUSION supports two such operators, with the framework designed to accommodate more as needed.

**Masking.** This operator removes junk data from a ciphertext tensor $ct$ using a 0/1 plaintext tensor $zo\_pt$ of matching dimensions, which can be expressed as:

$$\mathsf{Masking}(ct) = ct \times zo\_pt \tag{3}$$

In CKKS, this translates to one or more ciphertext–plaintext multiplications. Modeling masking as an explicit graph-level operator makes its semantics visible to analysis and optimization, enabling fusion opportunities.

**Compaction.** Inspired by TensorFlow's Strided_Slice, this operator extracts and reorganizes contiguous valid data from a ciphertext tensor. When lowered to CKKS, its implementation depends on the underlying data layout. In simple cases, such as Step ❹ in Figure 3, it requires only ciphertext rotations and additions and therefore adds no multiplicative depth. For more complex layouts, however, an additional masking step may be needed to clear junk slots, which incurs multiplicative depth.

Given a tensor $x$, Strided_Slice extracts a sub-tensor $y$ using attributes `begin`, `end`, and `strides`. For each dimension $i$, slicing starts at `begin[`$i$`]`, ends at `end[`$i$`]` (exclusive), and steps by `strides[`$i$`]`, yielding $\left\lceil \frac{\texttt{end[i]}-\texttt{begin[i]}}{\texttt{strides[i]}} \right\rceil$ elements.

For example, for a 2-D tensor in (row, column) order:

$$x = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix} \in \mathbb{R}^{2\times 4}$$

we can extract the following sub-tensor $y$ from $x$ with $\texttt{begin} = (0, 0)$, $\texttt{end} = (2, 4)$, and $\texttt{strides} = (1, 2)$:

$$y = \mathsf{Strided\_Slice}(x) = \begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix} \in \mathbb{R}^{2\times 2}$$

*2) Reexpressing DNN Operators:* In FHEFUSION, DNN operators are reexpressed using the two FHE-aware operators.

For AvgPool illustrated in Figure 3, we can express it using masking and Strided_Slice. Pooling with a kernel of size $h \times w$ reduces the spatial dimensions by that factor. With the `divisor_override`$(d_o)$ parameter, we obtain:

$$\mathsf{AvgPool}(x) = \mathsf{Strided\_Slice}\big(\mathsf{Masking}\big(\tfrac{1}{hw}\,\mathsf{AvgPool}'(x, d_o = 1)\big)\big) \tag{4}$$

As shown in Figure 3, this reformulation separates summation from averaging; only the latter consumes multiplicative depth, enabling more flexible fusion (as discussed later in Figure 8).

We now consider Conv, noting that GEMV is similar. With bias, $\mathsf{Conv}(x, w, b) = \mathsf{Conv}(x, w) + b$ can be reexpressed as:

$$\mathsf{Conv}(x, w, b) = \mathsf{Strided\_Slice}\big(\mathsf{Masking}(\mathsf{Conv}(x, w) + b)\big) \quad (5)$$

The ciphertext output of Conv often contains junk slots, cleared by masking. For stride $> 1$, additional gaps appear between valid values [8], [23] and are removed by Strided_Slice (Figure 4); when *stride* $= 1$, Strided_Slice is unnecessary.

### B. IR

Our IR represents DNNs at the graph level, comprising standard operators, FHE-aware operators, and polynomial-approximated RELU. Standard operators include Conv, GEMV, AvgPool, RELU, Flatten, Reshape, and Concat. We currently extend these with two FHE-aware operators—Masking and Strided_Slice (compaction)—which are sufficient for the CNN models evaluated in this paper, though the framework can readily support additional operators. Under CKKS, RELU is approximated by polynomials, with two types introduced in Section II and evaluated in Section VI.

### C. Algebraic Rules

Table II lists the algebraic rules used in FHEFUSION, illustrating how distributive, associative, and commutative laws apply to DNN and FHE-aware operators at the graph level. Each rule rewrites an expression into an equivalent, semantics-preserving form and is therefore always *legal*. The first twelve rules reduce multiplicative depth, while the remainder enable additional optimizations.

Rule applicability is governed by preconditions on tensor shapes and slicing parameters. For Masking-related rules, a binary mask must be derivable from the input shape, kernel, and stride, marking valid-data positions with 1s and others with 0s to ensure correct homomorphic multiplication. For Strided_Slice-related rules, the input shape, kernel, stride, and padding must align so slice boundaries match valid output regions. Constant-folding rules, by contrast, have no such constraints and apply whenever their pattern is matched. For instance, [FUSED-CMPT-MASKING] removes a masking operation when it can be absorbed into a new Strided_Slice, and [MASKING-ADD-RELU] applies when the trailing zero-valued slots of $\mathsf{RELU}(t_2)$ are exactly those cleared by $\mathtt{Masking}(t_1)$.

Our fusion algorithm, which will be described in Section V, treats most rules as potentially profitable, except those involving Strided_Slice (compaction): while propagating gaps can unlock later optimizations, it reduces slot utilization and SIMD parallelism. Such rules are therefore applied selectively under a simple FHE-aware cost model.

In most rules, such as [AVGPOOL-CMPT], $\mathsf{AvgPool}^1(\mathsf{Strided\_Slice}^1(t)) \implies \mathsf{Strided\_Slice}^2(\mathsf{AvgPool}^2(t))$ (as shown in Figure 6 and discussed shortly in Section IV-C2), operators use different superscripts to reflect updated attributes

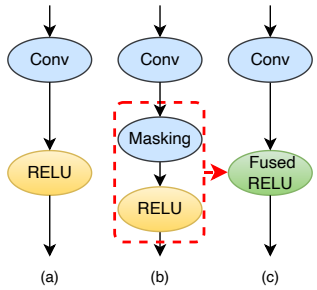


Fig. 5. Illustration of [FUSED-RELU-MASKING]. (a) Original computational graph. (b) After reexpressing Conv. (c) After fusing Masking and RELU.

after the rewrite. When propagating a Strided_Slice forward, the successor operator must inherit its slice information; for example, in [AVGPOOL-CMPT], $\mathsf{AvgPool}^2$ inherits the slice from $\mathsf{Strided\_Slice}^1$. To distinguish this inherited `strides` attribute from the conventional stride in operators such as Conv and AvgPool, we denote it as `Gap_Strides` (Figure 6).

These rules enable FHEFUSION to perform three categories of fusion: constant folding, masking folding, and compaction. We now highlight a few representative rules, explaining their legality and discussing their profitability.

*1) Masking-Related Fusion:* Figure 5 illustrates [FUSED-RELU-MASKING] on ResNet [31], the most complex DNN manually implemented under CKKS [19]. In Figure 5a, Conv is reexpressed with Masking and Strided_Slice (Figure 5b), where Strided_Slice is omitted since stride 1 is assumed in this example. Otherwise, RELU and Strided_Slice can be interchanged via [RELU-CMPT]. Applying [FUSED-RELU-MASKING] then merges Masking with RELU, yielding Figure 5c and eliminating the masking-related multiplication, thereby reducing multiplicative depth by one.

Masking removes junk data from ciphertext tensors but costs one multiplicative depth. Often inserted defensively, it is not always necessary; its use depends on subsequent operators. With careful scheduling, masking can be skipped or fused, as in Figure 5c where it merges with RELU.

**Legality Analysis.** We see why [FUSED-RELU-MASKING] is legal. Assume $N = 8$ with the mask in Equation (6) and the polynomial RELU approximation in Equation (7). Their fusion yields the equivalent form in Equation (8):

$$y = x[1, 1, 0, 0] \quad (6)$$

$$z = ay^2 + by + c \quad (7)$$

$$\begin{aligned} z &= a(x[1,1,0,0])^2 + b(x[1,1,0,0]) + c \\ &= ax^2[1,1,0,0]^2 + b[1,1,0,0]x + c \\ &= ax^2[1,1,0,0] + [b,b,0,0]x + c \\ &= [a,a,0,0]x^2 + [b,b,0,0]x + c \end{aligned} \quad (8)$$

which shows that fusing masking with RELU preserves correctness and obeys associativity, as captured in Table II.

**Profitability Analysis.** Applying [FUSED-RELU-MASKING] to Figure 5b fuses masking into the RELU in Figure 5c, reducing its multiplicative depth. Thus, when the compiler later inserts a bootstrap before the fused RELU, one fewer level needs to be restored, improving performance.

TABLE II

THE ALGEBRAIC RULES CURRENTLY USED IN FHEFUSION. HERE, "t" DENOTES A CIPHERTEXT TENSOR, AND *zo_m* IN [MASKING-SCALAR] IS THE CONSTANT VECTOR CREATED FROM THE MARK IN Masking($t$). SUPERSCRIPTS INDICATE OPERATOR VARIANTS WITH DIFFERENT OPERANDS OR ATTRIBUTES. PARAMETERS ARE OMITTED FOR BREVITY.

| Name | Rule | Before | After | Benefit |
|---|---|---|---|---|
| [FUSED-CMPT] | Assoc. | $\text{Strided\_Slice}^2(\text{Strided\_Slice}^1(t))$ | $\text{Strided\_Slice}^3(t)$ | reduces depth |
| [FUSED-GEMV-CMPT] | Assoc. | $\text{GEMV}^1(\text{Strided\_Slice}(t))$ | $\text{GEMV}^2(t)$ | reduces depth |
| [FUSED-GEMV-SCALAR] | Assoc. | $\text{GEMV}^1(t * scalar)$ | $\text{GEMV}^2(t)$ | reduces depth |
| [FUSED-SCALAR-GEMV] | Assoc. | $scalar * \text{GEMV}^1(t)$ | $\text{GEMV}^2(t)$ | reduces depth |
| [FUSED-CONV-SCALAR] | Assoc. | $\text{Conv}^1(t * scalar)$ | $\text{Conv}^2(t)$ | reduces depth |
| [FUSED-SCALAR-CONV] | Assoc. | $scalar * \text{Conv}^1(t)$ | $\text{Conv}^2(t)$ | reduces depth |
| [FUSED-SCALAR-RELU] | Assoc. | $scalar * \text{RELU}^1(t)$ | $\text{RELU}^2(t)$ | reduces depth |
| [SCALAR] | Assoc. | $(t * scalar^1) * scalar^2$ | $t * (scalar^1 * scalar^2)$ | reduces depth |
| [FUSED-RELU-MASKING] | Assoc. | $\text{RELU}^1(\text{Masking}(t))$ | $\text{RELU}^2(t)$ | reduces depth |
| [MASKING-SCALAR] | Assoc. | $\text{Masking}(t) * scalar$ | $t * (scalar * zo\_m)$ | reduces depth |
| [SCALAR-MASKING] | Assoc. | $\text{Masking}(t * scalar)$ | $t * (scalar * zo\_m)$ | reduces depth |
| [FUSED-CMPT-MASKING] | Assoc. | $\text{Strided\_Slice}(\text{Masking}(t))$ | $\text{Strided\_Slice}(t)$ | reduces depth |
| [CONCAT] | Distr. | $\text{Concat}(t_1 * scalar, t_2 * scalar)$ | $scalar * \text{Concat}(t_1, t_2)$ | enables further opt. |
| [CONCAT-CMPT] | Distr. | $\text{Concat}(\text{Strided\_Slice}(t_1), \text{Strided\_Slice}(t_2))$ | $\text{Strided\_Slice}(\text{Concat}(t_1, t_2))$ | enables further opt. |
| [CMPT-MUL] | Distr. | $\text{Strided\_Slice}(t_1) * \text{Strided\_Slice}(t_2)$ | $\text{Strided\_Slice}(t_1 * t_2)$ | enables further opt. |
| [CMPT-ADD] | Distr. | $\text{Strided\_Slice}(t_1) + \text{Strided\_Slice}(t_2)$ | $\text{Strided\_Slice}(t_1 + t_2)$ | enables further opt. |
| [MASKING-MUL] | Distr. | $\text{Masking}(t_1) * \text{Masking}(t_2)$ | $\text{Masking}(t1 * t2)$ | enables further opt. |
| [RELU-DIS] | Distr. | $a * t^2 + b * t$ | $a * (t^2 + \frac{b}{a} * t)$ | enables further opt. |
| [MASKING-ADD-RELU] | Distr. | $\text{Masking}(t_1) + \text{RELU}(t_2)$ | $\text{Masking}(t_1 + \text{RELU}(t_2))$ | enables further opt. |
| [CONV-CMPT] | Comm. | $\text{Conv}^1(\text{Strided\_Slice}^1(t))$ | $\text{Strided\_Slice}^2(\text{Conv}^2(t))$ | enables further opt. |
| [RELU-CMPT] | Comm. | $\text{RELU}(\text{Strided\_Slice}(t))$ | $\text{Strided\_Slice}(\text{RELU}(t))$ | enables further opt. |
| [AVGPOOL-CMPT] | Comm. | $\text{AvgPool}^1(\text{Strided\_Slice}^1(t))$ | $\text{Strided\_Slice}^2(\text{AvgPool}^2(t))$ | enables further opt. |
| [CMPT-SCALAR] | Comm. | $\text{Strided\_Slice}(t * scalar)$ | $scalar * \text{Strided\_Slice}(t)$ | enables further opt. |
| [SCALAR-CMPT] | Comm. | $scalar * \text{Strided\_Slice}(t)$ | $\text{Strided\_Slice}(t * scalar)$ | enables further opt. |
| [AVGPOOL-SCALAR] | Comm. | $\text{AvgPool}(t * scalar)$ | $scalar * \text{AvgPool}(t)$ | enables further opt. |
| [FLATTEN-MASKING] | Comm. | $\text{Flatten}(\text{Masking}(t))$ | $\text{Masking}(\text{Flatten}(t))$ | enables further opt. |
| [FLATTEN-CMPT] | Comm. | $\text{Flatten}^1(\text{Strided\_Slice}^1(t))$ | $\text{Strided\_Slice}^2(\text{Flatten}^2(t))$ | enables further opt. |
| [FLATTEN-SCALAR] | Comm. | $\text{Flatten}(t * scalar)$ | $scalar * (\text{Flatten}(t))$ | enables further opt. |
| [RESHAPE-CMPT] | Comm. | $\text{Reshape}^1(\text{Strided\_Slice}^1(t))$ | $\text{Strided\_Slice}^2(\text{Reshape}^2(t))$ | enables further opt. |
| [RESHAPE-SCALAR] | Comm. | $\text{Reshape}(t * scalar)$ | $scalar * (\text{Reshape}(t))$ | enables further opt. |

*2) Compaction-Related Fusion:* Compaction-related fusion is central to reducing multiplicative depth. The Strided_Slice operator removes gaps and packs valid data in ciphertext tensors. When applied to tensors already containing gaps, three outcomes are possible: (1) all gaps eliminated (e.g., GEMV in Figure 4); (2) gaps preserved (e.g., Conv with stride 1 and matching padding, or element-wise operators such as RELU); (3) gaps increased, producing sparser tensors (e.g., Conv with stride $> 1$ or AvgPool).

We next examine the legality and profitability of algebraic rules involving Strided_Slice in Table II.

**Legality Analysis.** A Strided_Slice operation preserves all valid data, changing only their placement through compaction. Swapping it with another operator merely alters the gap pattern it processes. As shown in Figure 4, fusing Strided_Slice with GEMV is semantics-preserving and thus legal, while eliminating explicit compaction.

We now examine [FUSED-CMPT], which merges two adjacent Strided_Slice operations in FHEFUSION (Figure 6). This example highlights the role of compaction and the adjustments required to operator attributes after applying a rewrite rule. For clarity, only key attributes are shown.

Before applying the rule, $x4 = \text{Strided\_Slice}^2(\text{AvgPool}^1(\text{Strided\_Slice}^1(x1)))$, with $x2 = \text{Strided\_Slice}^1(x1)$, $x3 = \text{AvgPool}^1(x2)$, and $x4 = \text{Strided\_Slice}^2(x3)$. By interchanging $\text{Strided\_Slice}^1$ and $\text{AvgPool}^1$ via [AVGPOOL-CMPT], we obtain $x4 = \text{Strided\_Slice}^2(\text{Strided\_Slice}^{1'}(\text{AvgPool}^2(x1)))$. Finally, fusing $\text{Strided\_Slice}^{1'}$ (not shown) and $\text{Strided\_Slice}^2$ into $\text{Strided\_Slice}^3$ (indicated by the two dashed red arrows)

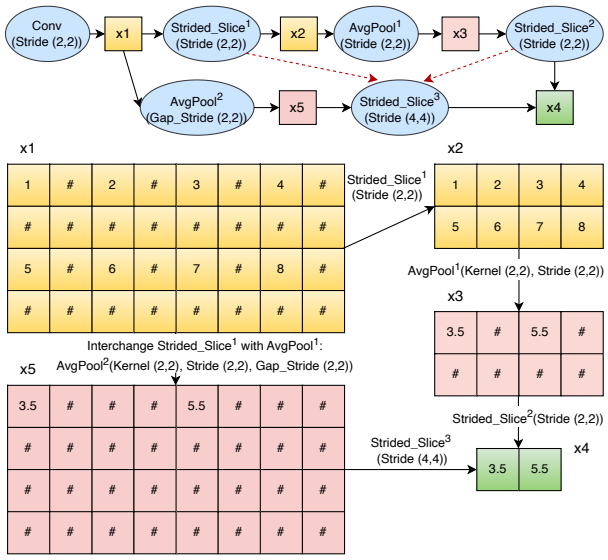


Fig. 6. Fusion of Adjacent Strided_Slice Operators.

simplifies this to $x4 = \mathsf{Strided\_Slice}^3(\mathsf{AvgPool}^2(x1))$.

**Profitability Analysis.** The cost of Strided_Slice depends on the number of gaps: more gaps reduce its local cost but hurt slot utilization and SIMD parallelism when lowered to CKKS. This may force a larger polynomial degree $N$, which degrades performance unless ciphertext tensors are sharded. The main benefit of Strided_Slice lies in consolidating multiple such operations into one. Thus, optimizations aim to merge it with other operators, avoiding growth in $N$ while keeping overhead low, as guided by our fusion algorithm (Section V).

*3) Constant-Folding-Related Rules:* Unlike traditional compilation, constant folding here primarily reduces multiplicative depth. A common DNN subgraph is $\mathsf{RELU} \to \mathsf{AvgPool} \to \mathsf{Conv}$ (Figure 2). With $\mathsf{RELU}(x) \approx ax^2 + bx + c$ for a ciphertext data $x$, $W$ the plaintext weight tensor of Conv, and $scalar = 1/(h \times w)$ for AvgPool, constants are folded as:

$$\begin{aligned} y &= \mathsf{Conv}(\mathsf{AvgPool}(\mathsf{RELU}(x)), W) \\ &= \mathsf{Conv}(\mathsf{AvgPool}(ax^2 + bx + c), W) \\ &= \mathsf{Conv}((a\ Scalar)\ \mathsf{AvgPool}'(x^2 + (b/a)x + c/a, d_o = 1), W) \\ &= \mathsf{Conv}(\mathsf{AvgPool}'(x^2 + (b/a)x + c/a, d_o = 1),\ a\ Scalar\ W) \end{aligned} \tag{9}$$

Here $d_o$ is the `divisor_override` attribute (Equation (4)).

**Legality Analysis:** Since scalar multiplication commutes with both AvgPool and Strided_Slice ([CMPT-SCALAR] and [AVGPOOL-SCALAR]), applying the scalar before or after these operators yields equivalent results.

**Profitability Analysis.** Folding constants $a$, $scalar$, and $W$ at compile time eliminates the required ciphertext–plaintext multiplications, reducing multiplicative depth by two.

## V. FHEFUSION: A FUSION FRAMEWORK FOR CKKS

Figure 7 overviews FHEFUSION. The input ONNX graph is lowered into a unified set of DNN and FHE-aware operators, forming a directed acyclic graph DAG. Algebraic rules (Table II) and an FHE-aware cost model (described below) then guide legal and profitable optimizations on this graph.

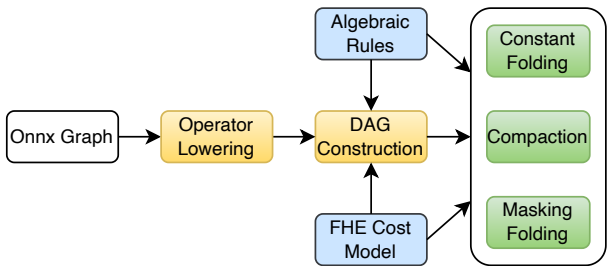


Fig. 7. The FHEFUSION fusion framework.

FHEFUSION, outlined in Algorithm 1, transforms an FHE program DAG $G$ into an optimized DAG $G'$ through a sequence of fusion steps. First, distributive rules are applied to expose additional optimization opportunities (line 1). As illustrated in Figures 3c and 3d, different application orders can lead to different fusion outcomes. Such reordering may enable or miss certain opportunities, much like pass ordering in traditional compilers. Our current ordering was chosen empirically for good coverage and efficiency, but it can be extended with profitability-driven or heuristic strategies.

**Algorithm 1:** FHEFUSION: Fusion Pipeline

```
Input: G: DAG of an FHE program
Output: G': optimized DAG
1 G' ← apply distributive rules from Table II to G
2 while there exists a fusion source s in G' do
3     targets ← FindFusionTargets(s)
4     if Profitable(s, targets) then
5         Fuse(s, targets) w.r.t. Table II
6         update G' with the fusion
7 return G'
```

Next, FHEFUSION iteratively processes all fusable sources (e.g., Masking and Strided_Slice) (lines 2–6). For each source $s$, FINDFUSIONTARGETS() (Algorithm 2) identifies candidate fusion targets (line 3). If merging $s$ with its targets is profitable (discussed shortly), the fusion is applied (line 5) and the DAG $G'$ is updated accordingly (line 6).

As illustrated in Figure 5b, Conv with stride 1 is lowered into a variant of Conv followed by Masking. With Masking as the fusion source and RELU as the target, fusing them eliminates the masking-related multiplication, thereby reducing multiplicative depth (Figure 5c).

Note that normalizing the IR before DAG construction also simplifies optimizations; for instance, constant folding is easier on $a(x^2 + \frac{b}{a}x + \frac{c}{a})$ than on $ax^2 + bx + c$ (Equation (9)).

FHEFUSION proceeds in three steps conceptually:

**Step 1. Finding Source-Specific Fusion Targets.** `FindFusionTargets`($s$) (Algorithm 2) identifies fusion targets for a source node $s$. The search first explores successors (line 2) and, if none are found, predecessors (line 4), following the rules in Table II. In `FindNeighbors`($s$, `"succ"`), an associative node $t$ is added to $targets$ (line 9); if $t$ is commutative, the search recurses (lines 11–12). Otherwise, the search terminates (line 14).

Fusion of Strided_Slice and Masking does not require examining predecessors, since they are always inserted after gap-generating operators; their fusion targets lie only among successors. By contrast, constant folding requires forward traversal to locate targets and may also involve predecessors. Rules needing both directions are applied iteratively.

A key issue in constant folding is deciding where to absorb constants. For Masking, they should be low-priority targets, whereas for operators like Conv and GEMV (Equation (2)) they must be prioritized, since their constants cannot otherwise be removed. As shown in Figure 8, after rewriting AvgPool as $\mathsf{AvgPool}'$ (Equation (4)) and commuting Flatten, the scalar factor (e.g., $\frac{1}{hw}$) is best folded into the GEMV weights (from (c) to (d)) rather than into Masking, which would hinder further fusion. The guiding principle is to fold constants into operators that inherently use constant vectors, reducing ciphertext–plaintext multiplications when lowered.

**Step 2. Profitability Analysis.** Given a source $s$ and fusion targets $targets$, profitability is checked (line 4, Al-

**Algorithm 2:** FindFusionTargets

```
Input  : s: source node
Output: targets: fusion targets of s
1  targets ← ∅
2  targets ← FindNeighbors(s, "succ")
3  if targets = ∅ then
4      targets ← FindNeighbors(s, "pred")
5  return targets
6  Function FindNeighbors(s, dir):
7  foreach t ∈ (dir = "succ" ? succ(s) : pred(s)) do
8      if t is associative with s then
9          targets ← targets ∪ {t}
10     else if t is commutative with s then
11         subtargets ← FindNeighbors(t, dir)
12         targets ← targets ∪ subtargets
13     else
14         return targets
```

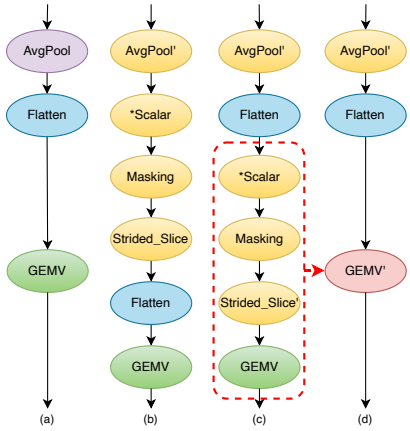


Fig. 8. Example of fusing Strided_Slice, masking, and scalar multiplication. (a) Original graph. (b) After lowering AvgPool. (c) After applying commutativity. (d) Final fusion into GEMV via associativity.

gorithm 1). At this stage, the IR includes DNN, Masking, Strided_Slice, and polynomial RELU operators, but not CKKS details. All rules in Table II are applied aggressively except those involving Strided_Slice. Propagating Strided_Slice may create tensors with gaps, reducing slot utilization and SIMD parallelism once lowered to CKKS, and possibly forcing a larger polynomial degree $N$ unless sharding is used. We therefore check gap sparsity: propagation is allowed when gaps are absent or minimal (e.g., from padding); for stride-induced gaps, it is allowed for AvgPool (Figure 6), but for Conv or GEMV only when it does not increase $N$. These checks define a simple quantitative cost model based on gap density and the resulting impact on $N$.

**Step 3. Fusion.** Applying a fusion operation (lines 5–6, Algorithm 1) is generally straightforward. The main exception is when Strided_Slice is involved, as traversed operators must be updated to account for gaps in tensors [23].

## VI. Evaluation

We integrated FHEFusion into ANT-ACE [15], an open-source FHE compiler for ONNX DNNs under CKKS on single-core CPUs. As FHE compilation is still nascent, prior work has mostly focused on foundational support for

TABLE III
FUSION COUNTS OF NGRAPH, AND FHEFUSION ACROSS 13 MODEL VARIANTS. FOR FHEFUSION, FOUR FUSION VARIANTS ARE CONSIDERED: CF (CONSTANT FOLDING), MF (MASKING), SF (COMPACTION VIA Strided_Slice), AND ALL (COMBINING ALL).

| Model | NGRAPH | FHEFUSION | | | |
|---|---|---|---|---|---|
| | | CF | MF | SF | ALL |
| LeNet-H | 1 | 2 | 4 | 4 | 10 |
| ResNet20-H | 0 | 1 | 19 | 4 | 24 |
| VGG11-H | 4 | 5 | 6 | 3 | 14 |
| MobileNet-H | 0 | 1 | 12 | 3 | 16 |
| SqueezeNet-H | 0 | 3 | 2 | 1 | 6 |
| AlexNet-H | 2 | 3 | 6 | 3 | 12 |
| CryptoNet-L | 0 | 0 | 1 | 1 | 2 |
| LeNet-L | 3 | 6 | 1 | 4 | 11 |
| ResNet20-L | 9 | 22 | 2 | 4 | 28 |
| VGG11-L | 7 | 13 | 3 | 3 | 19 |
| MobileNet-L | 26 | 28 | 1 | 3 | 32 |
| SqueezeNet-L | 9 | 27 | 1 | 1 | 29 |
| AlexNet-L | 6 | 10 | 1 | 3 | 14 |

single-core CPUs [8], [11], [12], [14]–[16]. Even without our optimizations, ANT-ACE attains a 2.24× speedup on ResNet20 [31] over the best handwritten implementation [19], whereas Fhelipe [8] only matches that baseline. Our 4000-line C++ implementation has been validated for correctness, and with fusion enabled, ANT-ACE preserves inference accuracy within 1.0%, consistent with prior reports [8], [19]. This small variation reflects CKKS's approximate arithmetic and polynomial RELU approximations; fusion only reorganizes operator execution and data flow without altering semantics.

To integrate FHEFusion, we aligned its IR with ANT-ACE's five-level stack (NN, VECTOR, SIHE, CKKS, and POLY). Algebraic rules are applied at the NN level, while those involving polynomially approximated RELU are handled at the SIHE level, where RELU is lowered in ANT-ACE.

We evaluate FHEFusion against NGraph [7], the only prior work on graph-level fusion, which applies three constant-folding patterns—AvgPool, Activation, and BatchNorm Folding—to reduce multiplicative depth (Section I). Because NGraph lacks bootstrapping and supports only small models, we reimplemented these patterns in ANT-ACE; hereafter, NGraph refers to this version. All three approaches use the same minimal-level bootstrapping strategy from [16].

**Models and Datasets.** We evaluate FHEFusion on seven DNNs widely used in FHE: CryptoNet [32], LeNet [33], ResNet20 [31], VGG11 [34], MobileNet [35], AlexNet [36], and SqueezeNet [37]. CryptoNet, which uses an $x^2$ activation, is evaluated on MNIST [33]; the others use RELU on CIFAR–10 [38] and require bootstrapping [9]. For RELU, we adopt two standard approximations (Section II-B): a multi-term polynomial [19] ($m$-H) and a quadratic $ax^2+bx+c$ [12] ($m$-L). The choice of RELU approximation is orthogonal to fusion; to ensure fairness, both approximations are shared across ANT-ACE, NGraph, and FHEFusion. Since CryptoNet only supports its native $x^2$ activation (treated as the $m$-L variant), we evaluate 13 model variants in total.

**Experimental Setting.** Experiments were done in Docker 25.0.1 on a Linux server with an Intel Xeon Platinum 8369B @2.70 GHz and 1024 GB RAM. Models were compiled into FHE form with scale factor $2^{59}$ (for fixed-point

TABLE IV
MULTIPLICATIVE DEPTHS OF ANT-ACE, NGRAPH, AND FHEFUSION.

| Model | ANT-ACE | NGRAPH | FHEFUSION | | | |
|---|---|---|---|---|---|---|
| | | | CF | MF | SF | ALL |
| LeNet-H | 66 | 65 | 64 | 61 | 57 | 50 |
| ResNet20-H | 259 | 259 | 258 | 239 | 254 | 233 |
| VGG11-H | 125 | 121 | 120 | 116 | 118 | 104 |
| MobileNet-H | 356 | 356 | 355 | 340 | 351 | 332 |
| SqueezeNet-H | 240 | 240 | 237 | 235 | 238 | 230 |
| AlexNet-H | 106 | 104 | 103 | 99 | 99 | 89 |
| CryptoNet-L | 11 | 11 | 11 | 10 | 8 | 7 |
| LeNet-L | 30 | 27 | 24 | 25 | 21 | 13 |
| ResNet20-L | 88 | 79 | 68 | 68 | 83 | 60 |
| VGG11-L | 53 | 46 | 40 | 44 | 46 | 29 |
| MobileNet-L | 113 | 87 | 85 | 97 | 108 | 75 |
| SqueezeNet-L | 78 | 69 | 57 | 73 | 76 | 51 |
| AlexNet-L | 43 | 37 | 33 | 36 | 36 | 24 |

TABLE V
IMPACT OF FUSION ON BOOTSTRAPS REQUIRED.

| Model | ANT-ACE | NGRAPH | FHEFUSION |
|---|---|---|---|
| LeNet-H | 4 | 4 | 3 |
| ResNet20-H | 18 | 18 | 18 |
| VGG11-H | 9 | 9 | 7 |
| MobileNet-H | 27 | 27 | 26 |
| SqueezeNet-H | 19 | 19 | 17 |
| AlexNet-H | 7 | 7 | 6 |
| CryptoNet-L | 0 | 0 | 0 |
| LeNet-L | 1 | 1 | 0 |
| ResNet20-L | 8 | 8 | 5 |
| VGG11-L | 6 | 5 | 3 |
| MobileNet-L | 9 | 7 | 5 |
| SqueezeNet-L | 8 | 7 | 6 |
| AlexNet-L | 3 | 3 | 2 |

arithmetic) and output precision $Q_0 = 2^{60}$, with each input image encrypted into a single ciphertext. Implementations used ACElib's APIs [39], compiled with GCC 10.2.1. The polynomial degree $N$ was set to $2^{13}$ (CryptoNet), $2^{15}$ (LeNet), $2^{16}$ (ResNet20), and $2^{17}$ (AlexNet, MobileNet, SqueezeNet, VGG11). The modulus $Q$ was chosen based on the multiplicative depth of each model. All results are reported as averages over three runs.

Our evaluation addresses three research questions:

- **RQ1:** Does FHEFUSION reduce multiplicative depth more effectively than NGRAPH?
- **RQ2:** Does it improve encrypted inference performance compared to NGRAPH?
- **RQ3:** Does it add only small compilation overhead?

### A. RQ1. Multiplicative Depth

Table III shows that FHEFUSION performs many more fusions than NGRAPH, which supports only CF, across 13 model variants. NGRAPH is most effective on quadratic ($m$-L) variants, as its CF rules target $\mathsf{RELU}(x) \approx ax^2 + bx + c$. We categorize fusions into three types—CF , MF, and SF—and also report ALL, the combined total.

Table IV shows the effect of applying each fusion type individually and in combination on reducing ANT-ACE's multiplicative depth across 13 model variants. NGRAPH offers only minor improvements, mainly on quadratic ($m$-L) models, while FHEFUSION achieves much larger reductions.

Reducing multiplicative depth lowers the cost of ciphertext operations by keeping them at lower levels (Table I), an effect reflected indirectly in the inference speedups reported below.

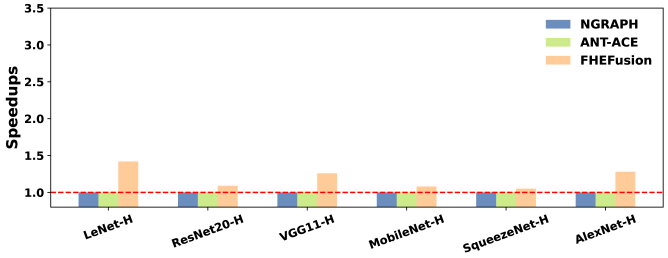


Fig. 9. Speedups of NGRAPH, ANT-ACE, and FHEFUSION (normalized to NGRAPH) for the six $m$-H models with high-degree polynomial RELU.

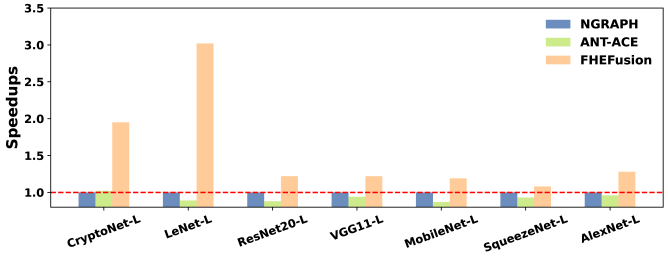


Fig. 10. Speedups of NGRAPH, ANT-ACE, and FHEFUSION (normalized to NGRAPH) for the seven $m$-L models with quadratic RELU approximations.

It also reduces bootstrap cost—because fewer levels must be restored—and can lower or even eliminate bootstraps entirely. As shown in Table V, both NGRAPH and FHEFUSION reduce bootstraps relative to ANT-ACE, with FHEFUSION achieving larger savings due to its greater depth reductions (Table IV); for example, bootstraps decrease for LeNet-H and ResNet20-L, and fall to zero for LeNet-L. The Appendix further shows that FHEFUSION also reduces the restored levels of inserted bootstraps compared with NGRAPH and ANT-ACE.

### B. RQ2. Inference Efficiency

Figure 9 reports the speedups of NGRAPH, ANT-ACE, and FHEFUSION (normalized to NGRAPH) on the six $m$-H models using high-degree polynomial RELU. FHEFUSION achieves the largest gains on LeNet-H, VGG11-H, and AlexNet-H, driven by their reduced multiplicative depth (Table IV) and fewer bootstraps (Table V).

Figure 10 shows results for the seven $m$-L models with quadratic RELU. FHEFUSION achieves the largest speedup on LeNet-L, where depth reduction (Table IV) eliminates bootstrapping (Table V), and the second-best on CryptoNet-L. For small models like CryptoNet-L, which do not bootstrap, lowering multiplicative depth still improves performance, as CKKS operations run faster at lower levels (Table I).

Overall, FHEFUSION's speedups mirror its reductions in multiplicative depth and bootstrapping. Across all 13 variants in Figures 9 and 10, NGRAPH is only marginally faster than ANT-ACE, while FHEFUSION reaches up to $3.02\times$ (average $1.40\times$). Since RELU is often approximated by simple polynomials, FHEFUSION 's advantages are especially pronounced.

Figure 11 presents an ablation study on the six $m$-H models in Figure 9, showing the contributions of the three fusion strategies. Figure 12 gives the corresponding results for the seven $m$-L models in Figure 10. For AlexNet-L, both SF

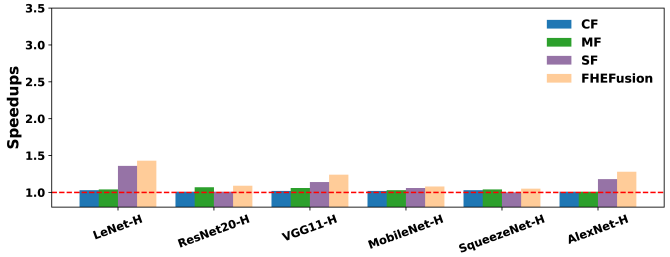


Fig. 11. Ablation of FHEFUSION (speedups vs. ANT-ACE) on the six $m$-H models in Figure 9.

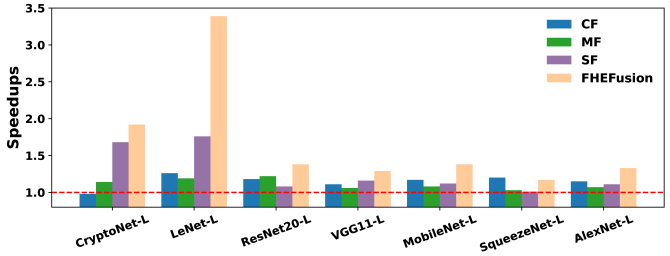


Fig. 12. Ablation of FHEFUSION (speedups vs. ANT-ACE) on the seven $m$-L models in Figure 10.

and MF reduce multiplicative depth by 7 (Table IV), but SF delivers greater performance gains by also lowering CKKS rotations, additions, and multiplications.

Additional comparisons of NGRAPH and FHEFUSION on operator-level subgraphs are provided in the Appendix.

### C. RQ3. Compile Times

Table VI reports the compile times of ANT-ACE, NGRAPH and FHEFUSION across 13 model variants. Among the three fusion types applied separately, SF is usually the slowest due to handling gapped tensors. Overall, analysis and optimization operate on the model DAG, where fusion targets partition it into subgraphs and algebraic rules consider only local neighbors, keeping overhead low.

## VII. RELATED WORK

The first lattice-based FHE scheme was proposed by Gentry in 2009 [5]. Subsequent schemes, including BGV [25], BFV [26], GSW [40], FHEW [41], TFHE [42], and CKKS [6], are based on LWE [43] or RLWE [24]. BGV, BFV, and CKKS support SIMD-style batching, and RNS-CKKS [27], which enables efficient fixed-point arithmetic, is the most widely used for encrypted machine learning inference.

Recent work on FHE [7], [8], [10]–[15], [21] has emphasized programmability through compilers [7], [11], [13], [15], DSLs [12], [14], and libraries [10], along with optimizations for data layout [8], rescaling [14], noise management [16]–[18], and operator implementations [19], [20]. Operator fusion, however, remains limited: FHE-MP-NN [19] manually implements ResNet [31] under CKKS, fusing Conv and a few cross-operator cases such as Conv+BN, while the NGRAPH compiler [7] provides only three constant-folding patterns. In contrast, FHEFUSION is more general, leveraging algebraic properties to enable fusion beyond constant folding.

TABLE VI
COMPILE TIMES OF ANT-ACE, NGRAPH AND FHEFUSION (SECONDS).

| Model | ANT-ACE | NGRAPH | FHEFUSION | | | |
|---|---|---|---|---|---|---|
| | | | CF | MF | SF | ALL |
| LeNet-H | 0.25 | 0.25 | 0.25 | 0.24 | 0.29 | 0.28 |
| ResNet20-H | 1.46 | 1.49 | 1.50 | 1.39 | 2.25 | 2.33 |
| VGG11-H | 3.39 | 3.73 | 3.71 | 3.32 | 7.16 | 7.91 |
| MobileNet-H | 1.92 | 2.06 | 2.05 | 1.88 | 2.90 | 2.86 |
| SqueezeNet-H | 3.63 | 3.90 | 3.88 | 3.60 | 3.63 | 3.86 |
| AlexNet-H | 2.21 | 2.45 | 2.52 | 2.22 | 7.27 | 9.04 |
| CryptoNet-L | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 |
| LeNet-L | 0.10 | 0.10 | 0.09 | 0.09 | 0.14 | 0.15 |
| ResNet20-L | 0.85 | 0.89 | 0.88 | 0.83 | 1.60 | 1.68 |
| VGG11-L | 3.18 | 3.39 | 3.34 | 3.02 | 6.83 | 7.68 |
| MobileNet-L | 1.07 | 1.13 | 1.10 | 1.04 | 1.88 | 2.00 |
| SqueezeNet-L | 3.17 | 3.48 | 3.43 | 3.01 | 3.16 | 3.39 |
| AlexNet-L | 2.02 | 2.49 | 2.52 | 1.99 | 7.45 | 9.62 |

Eliminating gaps in ciphertexts and avoiding unnecessary masking are critical for FHE performance, yet prior work has given these issues little attention. Qiwu [22] uses a DSL-based approach to manipulate gaps and increase SIMD parallelism, while FHE-MP-CNN [19] and CHET [12] note the problem without solutions. Fhelipe [8] supports CKKS operations on gapped ciphertexts only across two consecutive GEMV layers, but simplifies by assuming gap data are zero instead of handling masking explicitly. To our knowledge, this is the first work to propose fusion optimizations that jointly address masking and compaction within a general framework.

## VIII. CONCLUSION

We introduced FHEFUSION, an operator fusion framework for accelerating DNN inference under CKKS. FHEFUSION builds on a new IR that integrates standard DNN operators with FHE-aware operators, preserving high-level semantics while exposing CKKS-specific behaviors essential for fusion. Guided by algebraic rules and an FHE-aware cost model, it performs constant folding, masking folding, and compaction to reduce multiplicative depth and broaden optimization opportunities. Our evaluation demonstrates substantial speedups over state-of-the-art systems, establishing FHEFUSION as a solid foundation for future FHE compiler development.

Looking ahead, we plan to extend FHEFUSION along two directions. First, leveraging its hardware-agnostic design, we will broaden support beyond single-core CPUs to multi-core architectures, GPUs [44]–[47], and custom accelerators, enabling scalable high-performance FHE. Because the fusion rules and algorithms make no hardware assumptions, they can be ported to GPU backends as FHE GPU frameworks mature. Second, to meet rising privacy demands in modern AI, we will explore applying FHEFUSION's techniques to privacy-preserving LLM inference. Its two FHE-aware primitives—masking and Strided_Slice —capture ciphertext-level data selection and reorganization patterns also found in transformers, allowing operators such as attention and projection to be expressed with minimal extensions.

## DATA-AVAILABILITY STATEMENT

All data and scripts are available on Zenodo [48].

## ARTIFACT

### A. Abstract

FHEFUSION has been implemented in the open-source ANT-ACE FHE compiler [15] and is available at https://github.com/ant-research/ace-compiler/tree/fhefusion. It supports reproducing our results on 13 model variants of 7 popular DNNs (MNIST/CIFAR-10) and compares against NGRAPH for fusion counts, depths reduction, bootstraps reduction, inference speed, and compilation overhead. Experiments require Docker on Linux, Intel Xeon Platinum, 512GB memory and about 50 hours to complete.

### B. Artifact check-list (meta-information)

- **Binary:** Source code and scripts included to regenerate binaries.
- **Model:** CryptoNet, LeNet, ResNet20, VGG11, MobileNet, SqueezeNet, AlexNet(The last six models each have two variants from different RELU approximations).
- **Data set:** MNIST, CIFAR-10.
- **Run-time environment:** Docker container version 25.0.1, dependencies detailed in https://github.com/ant-research/ace-compiler/blob/fhefusion/Dockerfile.
- **Output:** Table3.pdf, Table4.pdf, Table5.pdf, Table6.pdf, Figure9.pdf, Figure10.pdf, Figure11.pdf and Figure12.pdf.
- **Experiments:** Detailed in https://github.com/ant-research/ace-compiler/blob/fhefusion/README.md.
- **How much disk space required (approximately)?:** 50 GB.
- **How much time is needed to complete experiments (approximately)?:** 50 hours.
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** Apache-2.0 WITH LLVM-exception.

### C. Description

#### 1) How to Access:

- Source code: https://github.com/ant-research/ace-compiler/tree/fhefusion
- Docker Hub: opencc/ace:fusion

#### 2) Hardware Dependencies:

- To match configurations in this paper: Intel Xeon Platinum 8369B CPU @ 2.70 GHz with 512 GB memory.

#### 3) Software Dependencies:

- Docker container version 25.0.1, other dependencies detailed in https://github.com/ant-research/ace-compiler/blob/fhefusion/Dockerfile.

### D. Installation

There are two options to setup the artifact environments which are described on https://github.com/ant-research/ace-compiler/tree/fhefusion. It is recommended to pull the pre-built docker image (opencc/ace:fusion) from Docker Hub:

```
cd [YOUR_DIR_TO_DO_AE]
mkdir -p ae_result
docker pull opencc/ace:fusion
docker run -it --name fusion -v "$(pwd)"/ae_result:/app/
    ae_result --privileged opencc/ace:fusion bash
```

Alternatively, if you encounter issues pulling the pre-built image, you can build the image from the Dockerfile:

```
cd [YOUR_DIR_TO_DO_AE]
git clone https://github.com/ant-research/ace-compiler.git
cd ace-compiler
git checkout fhefusion
mkdir -p ae_result
docker build -t ace:fusion .
docker run -it --name fhefusion -v "$(pwd)"/ae_result:/app/
    ae_result --privileged ace:fusion bash
```

A local directory "ae_result" is created and mounted in the docker container to collect the generated figures and tables.

### E. Experiment workflow

To reproduce Tables 3-6 and Figures 9-12, execute the following command in the /app directory of the container:

```
./scripts/run_full.sh > fusion.log 2>&1 &
```

This process will take about 50 hours to complete on the recommended computing platform.

### F. Evaluation and Expected Results

During artifact evaluation, the newly generated Table 6 may differ slightly from those in our paper due to variations in workload and I/O on the hardware. Similarly, the generated Figures 9, 10, 11 and 12 may exhibit minor differences compared to the one in our paper. However, the generated Tables 3, 4 and 5 should match exactly with the published version.

## REFERENCES


[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "TensorFlow: A system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[2] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Q. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "TVM: An automated end-to-end optimizing compiler for deep learning," in *USENIX Symposium on Operating Systems Design and Implementation*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:52939079

[3] W. Niu, J. Guan, Y. Wang, G. Agrawal, and B. Ren, "DNNFusion: accelerating deep neural networks execution with advanced operator fusion," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, ser. PLDI 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 883–898. [Online]. Available: https://doi.org/10.1145/3453483.3454083

[4] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *PyTorch: an imperative style, high-performance deep learning library*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[5] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009.

[6] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology – ASIACRYPT 2017*. Cham: Springer International Publishing, 2017, pp. 409–437.

[7] F. Boemer, Y. Lao, and C. Wierzynski, "nGraph-HE: A graph compiler for deep learning on homomorphically encrypted data," *CoRR*, vol. abs/1810.10121, 2018. [Online]. Available: http://arxiv.org/abs/1810.10121

[8] A. Krastev, N. Samardzic, S. Langoski, S. Devadas, and D. Sanchez, "A tensor compiler with automatic data packing for simple and efficient fully homomorphic encryption," in *Proc. ACM Program. Lang. 8, PLDI, Article 152, 25 pages*, 2024.

[9] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for approximate homomorphic encryption," in *In Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, Part I 37*, 2018, pp. 360–384.

[10] S. Halevi and V. Shoup, "Algorithms in HElib," in *Advances in Cryptology – CRYPTO 2014*, J. A. Garay and R. Gennaro, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 554–571.

[11] A. Ebel, K. Garimella, and B. Reagen, "Orion: A fully homomorphic encryption framework for deep learning," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 734–749. [Online]. Available: https://doi.org/10.1145/3676641.3716008

[12] 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 *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 142–156.

[13] A. Viand, P. Jattke, M. Haller, and A. Hithnawi, "HECO: Fully homomorphic encryption compiler," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 4715–4732. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/viand

[14] R. Dathathri, B. Kostova, O. Saarikivi, W. Dai, K. Laine, and M. Musuvathi, "EVA: An encrypted vector arithmetic language and compiler for efficient homomorphic computation," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020, pp. 546–561.

[15] L. Li, J. Lai, P. Yuan, T. Sui, Y. Liu, Q. Zhu, X. Zhang, L. Xiao, W. Chen, and J. Xue, "ANT-ACE: An fhe compiler framework for automating neural network inference," in *Proceedings of the 23rd ACM/IEEE International Symposium on Code Generation and Optimization*, ser. CGO '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 193–208. [Online]. Available: https://doi.org/10.1145/3696443.3708924

[16] Y. Liu, J. Lai, L. Li, T. Sui, L. Xiao, P. Yuan, X. Zhang, Q. Zhu, W. Chen, and J. Xue, "ReSBM: Region-based scale and minimal-level bootstrapping management for FHE via min-cut," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ser. ASPLOS '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 924–939. [Online]. Available: https://doi.org/10.1145/3669940.3707276

[17] S. Cheon, Y. Lee, D. Kim, J. M. Lee, S. Jung, T. Kim, D. Lee, and H. Kim, "DaCapo: Automatic bootstrapping management for efficient fully homomorphic encryption," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 6993–7010. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/cheon

[18] Y. Lee, S. Cheon, D. Kim, D. Lee, and H. Kim, "ELASM: Error-latency-aware scale management for fully homomorphic encryption," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4697–4714.

[19] E. Lee, J.-W. Lee, J. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and W. Choi, "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," Cryptology ePrint Archive, Paper 2021/1688, 2021. [Online]. Available: https://eprint.iacr.org/2021/1688

[20] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 1651–1669. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar

[21] P. Yuan, Y. Liu, J. Lai, L. Li, T. Sui, L. Xiao, X. Zhang, Q. Zhu, and J. Xue, "Metakernel: Enabling efficient encrypted neural network inference through unified mvm and convolution," *Proc. ACM Program. Lang.*, vol. 9, no. OOPSLA2, Oct. 2025. [Online]. Available: https://doi.org/10.1145/3763095

[22] Z. Zhang, Y. Liu, Y. Zhang, Z. Chen, J. Zhao, X. Feng, H. Cui, and J. Xue, "Qiwu: Exploiting ciphertext-level SIMD parallelism in homomorphic encryption programs," in *Proceedings of the 23rd ACM/IEEE International Symposium on Code Generation and Optimization*, ser. CGO '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 523–537. [Online]. Available: https://doi.org/10.1145/3696443.3708917

[23] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim, and J.-S. No, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *IEEE Access*, vol. 10, pp. 30 039–30 054, 2022.

[24] L. Vadim, P. Chris, and R. Oded, "On ideal lattices and learning with errors over rings," *Journal of the Association for Computing Machinery*, vol. 60, no. 6, pp. 43.1–43.35, 2013.

[25] B. Zvika, G. Craig, and V. Vinod, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.

[26] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Paper 2012/144, 2012. [Online]. Available: https://eprint.iacr.org/2012/144

[27] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Selected Areas in Cryptography – SAC 2018*, C. Cid and M. J. Jacobson Jr., Eds. Cham: Springer International Publishing, 2019, pp. 347–368.

[28] H. Chen, I. Chillotti, and Y. Song, "Improved bootstrapping for approximate homomorphic encryption," in *In Annual International Conference on the Theory and Applications of Cryptographic Techniques. Cham: Springer International Publishing.*, 2019, pp. 34–54.

[29] L. Yongwoo, H. Seonyeong, C. Seonyoung, J. Shinnung, K. Changsu, K. Eunkyung, L. Dongyoon, and K. Hanjun, "HECATE: Performance-aware scale optimization for homomorphic encryption compiler," in *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2022, pp. 193–204.

[30] J. Park, M. J. Kim, W. Jung, and J. H. Ahn, "AESPA: accuracy preserving low-degree polynomial activation for fast private inference," *CoRR*, vol. abs/2201.06699, 2022. [Online]. Available: https://arxiv.org/abs/2201.06699

[31] H. Kaiming, Z. Xiangyu, R. Shaoqing, and S. Jian, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[32] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 201–210. [Online]. Available: https://proceedings.mlr.press/v48/gilad-bachrach16.html

[33] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[34] S. Karen and Z. Andrew, "Very deep convolutional networks for large-scale image recognition," *Computer Science*, 2014.

[35] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[36] K. Alex, S. I., and H. G., "ImageNet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, no. 2, 2012.

[37] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360

[38] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Toronto, Ontario, Tech. Rep. 0, 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[39] A. Compiler, https://github.com/ace-compiler/ace-compiler, 2024.

[40] G. Craig, S. Amit, and W. Brent, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Advances in Cryptology – CRYPTO 2013*, C. Ran and G. J. A., Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 75–92.

[41] D. Léo and M. Daniele, "FHEW: bootstrapping homomorphic encryption in less than a second," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2015, pp. 617–640.

[42] C. Ilaria, G. Nicolas, G. Mariya, and I. Malika, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22*. Springer, 2016, pp. 3–33.

[43] R. Oded, "On lattices and learning with errors and random linear codes and cryptography," *Proceedings of the Annual ACM Symposium on Theory of Computing*, vol. 56, no. 6, pp. 84–93, 2009.

[44] S. Fan, Z. Wang, W. Xu, R. Hou, D. Meng, and M. Zhang, "Tensorfhe: Achieving practical computation on encrypted data using gpgpu," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. Los Alamitos, CA, USA: IEEE Computer Society, mar 2023, pp. 922–934. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/HPCA56546.2023.10071017

[45] F. Zheng, G. Fan, W. Tang, Y. Song, T. Zhou, Y. Zhao, J. Dong, J. Lin, S. Yan, and J. Jing, "Gif-fhe: A comprehensive implementation and evaluation of gpu-accelerated fhe with integer and floating-point computing power," *IEEE Transactions on Parallel and Distributed Systems*, vol. 36, no. 8, pp. 1524–1541, 2025.

[46] G. Fan, M. Zhang, F. Zheng, S. Fan, T. Zhou, X. Deng, W. Tang, L. Kong, Y. Song, and S. Yan, "Warpdrive: Gpu-based fully homomorphic encryption acceleration leveraging tensor and cuda cores," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2025, pp. 1187–1200.

[47] D. Jiao, X. Deng, Z. Wang, S. Fan, Y. Chen, D. Meng, R. Hou, and M. Zhang, "Neo: Towards efficient fully homomorphic encryption acceleration using tensor core," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ser. ISCA '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 107–121. [Online]. Available: https://doi.org/10.1145/3695053.3731408

[48] T. Sui, "Reproduction package for article FHEFusion," Zenodo, 2025. [Online]. Available: https://doi.org/10.5281/zenodo.17630291