Uranus: Ultra-efficient Acceleration Architecture for the Privacy Inference of Graph Neural Networks

Xicheng Xu[1,2,3], Yinghao Yang[1,2(✉)], Fuyao Liu[1,2,3], Xiaowei Li[1,2,3], Hang Lu[1,2,3(✉)]
SKLP, Institute of Computing Technology, CAS[1];
University of Chinese Academy of Sciences[2];
Zhongguancun Laboratory[3]

***Abstract*—Graph Neural Networks (GNNs) are increasingly applied across various domains, including social media and recommendation systems. However, data privacy protection has become a critical issue for GNN applications. Fully Homomorphic Encryption (FHE), which enables computation on encrypted data, has emerged as a mainstream solution for secure GNN inference. However, due to the high computational costs of FHE-based GNNs, dedicated accelerators are necessary to make them practical. Existing CKKS-based GNN algorithms require large encryption parameters, imposing high demands on hardware resources. The state-of-the-art design PPGNN is hardware friendly on memory by combining CKKS with TFHE, but TFHE's SISD computing characteristic causes a dramatic increase in computational overhead, limiting its performance benefits. This paper proposes a novel software-hardware co-designed GNN accelerator architecture, Uranus, which integrates a hardware-friendly algorithmic framework with a matching accelerator design. The inference framework leverages CKKS and BFV schemes to enable efficient linear layer computations and SIMD-based nonlinear calculations with reduced encryption parameters suitable for hardware constraints. Additionally, the highly reconfigurable hardware accelerator maximizes utilization and performance. Key results demonstrate the following: (1) up to 92× speedup compared to CKKS accelerators; (2) achieves 3.3× to 4.2× performance improvement over the SOTA PPGNN architecture; (3) over 119× in energy efficiency improvement compared to PPGNN.**

I. Introduction

Graph Neural Networks (GNNs) are widely applied in recommendation systems, social networks, knowledge graphs, natural science, etc [10], [30], [32]. However, handling sensitive data, such as personal information in cloud services or medical records in bioinformatics, raises privacy concerns. Protecting sensitive information during GNN inference has thus become a critical research focus.

Fully Homomorphic Encryption (FHE), known for its "computable but not visible" feature, is a prominent privacy-preserving technology. This enables its wide application in scenarios requiring data privacy, including GNN inference. CryptoGCN [23] and Penguin [24] are FHE-based GNN implementations, both of which optimize the linear layer of the GNN to reduce the computational overhead. However, to improve the inference performance, they use squared activation or second-order polynomial fitting to implement the activation layer of the GNN, respectively, which leads to a drastic decrease in their accuracy and low practicality. In fact, high-precision nonlinear computations require high-order polynomial fitting implementations, such as a 343-degree polynomial for sign function approximation [19], [31], resulting in at least 12 multiplication depths per ReLU operation. Even a simple 3-layer GNN requires 42 multiplication depths. Therefore, practical high-precision privacy GNN inference requires bootstrapping to support large multiplication depths, which in turn requires large parameter settings and high overhead.

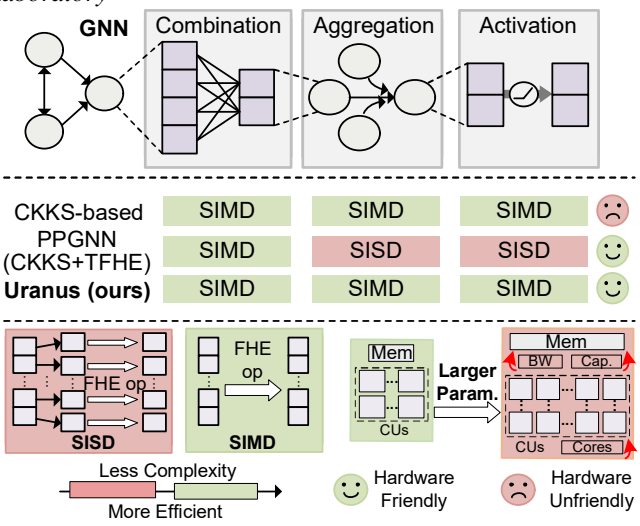


Fig. 1. Comparison of GNN acceleration accross CKKS-based, PPGNN and Uranus implementations. Single instruction multiple data (SIMD) has lower computational complexity and higher efficiency compared to single instruction single data (SISD). Smaller encryption parameters mean lower hardware resource requirements and design complexity, more hardware friendly.

To improve the inference efficiency, FHE-based GNN requires specialized accelerators to improve its efficiency and practicality. As shown in Fig. 1, although general-purpose FHE accelerators like CraterLake [26] and SHARP [16] can support CKKS-based implementations, they require huge scratchpads (180MB-512MB) to meet the memory access requirements for large parameters, significantly increasing the area and power consumption of the accelerators. PPGNN [31], the first hardware-software co-designed GNN accelerator, improves accuracy and inference speed using a combination of CKKS and TFHE schemes. PPGNN's co-design reduces parameter demands, offering a memory advantage in hardware accelerating. However, PPGNN can only support GNN models without edge weights, and its nonlinear layer implementation based on TFHE's Programmable Bootstrapping (PBS) essentially extracts packed data and computes them in SISD manner, which introduces a dramatic increase in computational complexity. For instance, processing a single iteration of a simple 3-layer

Yinghao Yang and Hang Lu are corresponding authors. This work was supported in part by the National Natural Science Foundation of China under Grant 62172387; in part by the CCF Phytium Fund 2023; in part by the Open Research Fund of the State Key Laboratory of Blockchain and Data Security, Zhejiang University.

TABLE I
NOTATIONS, PARAMETERS AND KEY OPERATIONS.

Notation	Description
$\mathbf{m}$	Polynomial plaintext.
N	Degree of the polynomial.
Q	Ciphertext modulus.
$\mathcal{R}_Q$	Cyclotomic polynomial ring, $\mathbb{Z}_Q/(X^N+1)$
$[[\mathbf{m}]]$	RLWE ciphertext of $\mathbf{m}$, $[[\mathbf{m}]] = (a(X), b(X)) \in \mathcal{R}_Q^2$
m	Scalar plaintext.
n	Length of the vector.
$[[m]]$	LWE ciphertext of m, $[[m]] = (\vec{a}, b) \in \mathbb{Z}_Q^{n+1}$
HAdd	Homomorphic addition of two ciphertexts
PMult	Homomorphic multiplication of plaintext and ciphertext
CMult	Homomorphic multiplication of two ciphertexts
Extract	Extract LWE ciphertexts from RLWE ciphertext
Repack	Pack LWE ciphertexts to RLWE ciphertext

GNN on the Pubmed dataset involves hundreds of thousands of PBS.

Therefore, an efficient FHE-based GNN inference acceleration system needs to consider both algorithms and hardware, designing a hardware-friendly GNN inference framework while ensuring high efficiency. We propose Uranus, a hardware-software co-designed GNN accelerator. Uranus features a hardware-friendly and efficient inference framework based on hybrid FHE scheme (CKKS for linear layers and BFV for nonlinear layers) and a specialized hardware architecture to accelerate inference speed. The contributions of this paper are as follows:

- We propose a high-accuracy and hardware friendly Uranus framework for the GNN inference under FHE. This framework is based on the CKKS and BFV implementations, and designs efficient matrix multiplication for combination and aggregation in GNN and an accurate SIMD activation method supporting small parameters with high accuracy.
- We propose a hardware accelerator architecture that matches the Uranus framework, which includes a shifter unit for FHE scheme conversion and a highly reconfigurable HMU based on a shared basic computing unit capable of supporting all modulo-related computations in the Uranus framework, achieving excellent hardware utilisation and performance.
- We evaluate the performance of Uranus based on RTL implementation at the 7nm technology node. We highlight the following results: (1) achieves 9.0×-92× and 3.3×-4.2× speedup compared to SOTA CKKS accelerator and GNN-oriented accelerating architecture PPGNN. (2) over 119× efficiency (EDAP) improvement compared to SOTA PPGNN.

II. PRELIMINARY

A. FHE schemes and operations

State-of-the-art HE schemes can be broadly categorized into RLWE-based schemes, such as CKKS [6] and BFV [11], and LWE based schemes, such as TFHE [7], depending on the underlying hardness assumptions. Table I summarizes the key notations, parameters and operations of these schemes.

Significant differences exist between them in terms of ciphertext structure and homomorphic operations. RLWE-based schemes encrypt a plaintext polynomial $\mathbf{m}$ into an RLWE ciphertext consisting of two polynomials, $[[\mathbf{m}]] = (a(X), b(X)) \in \mathcal{R}_Q^2$, while LWE-based schemes encrypt a single plaintext message m into an LWE ciphertext composed of a vector and a scalar, $[[m]] = (\vec{a}, b) \in \mathbb{Z}_Q^{n+1}$. These differences in ciphertext formats inherently lead to variations in the supported homomorphic operations.

Despite their structural and operational differences, various conversion techniques have been proposed and widely adopted [4], [7], [21]. Among them, Extraction enables the extraction of an LWE ciphertext corresponding to a specific message from an RLWE ciphertext, while Repacking allows multiple LWE ciphertexts to be packed into a single RLWE ciphertext in a prescribed order. In Uranus, scheme conversion is primarily employed to rearrange data positions within the polynomial.

B. FHE-based GNN Inference

GNN computations typically comprise three key stages: combination, aggregation, and activation. These stages can be mathematically expressed in matrix form as:

$$\mathbf{V}^{(k)} = \sigma\big(\mathbf{A} \cdot \mathbf{V}^{(k-1)} \cdot \mathbf{W}^{(k)}\big), \tag{1}$$

where $\mathbf{A}$ is the adjacency matrix of the graph, $\mathbf{V}^{(k)}$ is the feature matrix at layer k, $\mathbf{W}^{(k)}$ is the trainable weight matrix, and σ is the activation function.

Under FHE, the combination ($\mathbf{V}^{(k-1)} \cdot \mathbf{W}^{(k)}$, as in (1)) is performed via ciphertext-plaintext matrix multiplication. The aggregation ($\mathbf{A} \cdot \mathbf{V}^{(k-1)} \cdot \mathbf{W}^{(k)}$) involves ciphertext-ciphertext matrix multiplication, while the activation stage is typically implemented using polynomial approximations. While these computations enable GNN inference under FHE, they also deplete the ciphertext's multiplicative depth.

C. Threat Model

Following prior privacy-preserving deep learning approaches [3], [8], [13], [18], Uranus focuses on cloud-based GNN inference services. In this system, the client retains the confidential graph data (i.e., $\mathbf{A}$ and $\mathbf{V}^{(0)}$), while the server manages the GNN model parameters(i.e., $\mathbf{W}^k$). The server is assumed to be semi-honest, meaning it performs inference correctly but may try to infer or leak the client's data. During inference, the client encrypts graph data using FHE. The server performs GNN inference on the encrypted data and sends the results back to the client, who decrypts them to obtain the final output. Since Uranus operates on the server side, the client's graph data remain encrypted, while the model weights are in plaintext.

III. MOTIVATION

FHE-based GNN inference incurs substantial computational overhead, necessitating specialized accelerators for practical

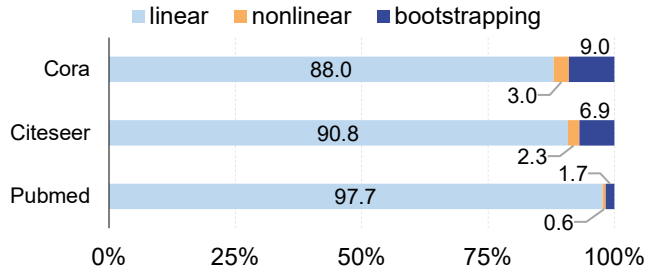


Fig. 2. Latency breakdown of GNN inference. We implement GNN on three datasets (Cora,Citeseer, and Pubmed) on the open source FHE library SEAL [27] based on the CryptoGCN [31] design method and combined with high precision polynomial fitting [19].

deployment. Achieving true efficiency requires joint consideration of both software frameworks and hardware architectures.

A. Discrete Frameworks and Accelerators

Current optimizations for FHE-based GNN inference algorithms and FHE accelerators are largely discrete. Algorithms focus on reducing computational complexity without adapting to hardware accelerators, while general-purpose FHE accelerators lack designs tailored for GNNs, hindering their integration for efficient performance. As shown in Fig. 2, the linear layer in FHE-based GNN involves computationally expensive ciphertext matrix multiplications, accounting for a significant portion of the total computational overhead (over 88%), while the nonlinear layer and bootstrapping overhead are minimal (less than 12%). Theoretically, the linear layer requires only a multiplication depth of 2, achievable with small encryption parameters. However, bootstrapping for nonlinear functions in GNNs demands significantly higher encryption parameters (polynomial length 65536, ciphertext modulus 1500-bit). This results in the linear layer also only operating at a polynomial length of 65536, thus having a high overhead. However, since the number of features of GNNs shrinks drastically as the linear layer is computed (e.g., 2708 to 32 in Cora and and 19717 to 32 in Pubmed), the source of large parameter requirements, i.e. bootstrapping, instead has a small overhead due to the fact that only a small number of feature data ciphertexts need to be refreshed. Therefore, accelerating FHE-based GNN inference by existing accelerator designs, i.e., ARK [17], CraterLake [26], and SHARP [16], will result in very low hardware utilization, which stems from the deployment of multiple HBMs and large capacity scratchpads (180MB-512MB) primarily for bootstrapping, which accounts for less than 10% of the GNN compute load.

B. Unbalanced Software-hardware Co-design for GNN

PPGNN [31] is the first and only hardware-software co-design GNN acceleration work that focuses on minimizing the FHE encryption parameters for GNNs. It combines arithmetic and logic FHE schemes, employing CKKS for ciphertext-plaintext matrix multiplication and TFHE for ciphertext matrix multiplication and nonlinear layer evaluation, reducing encryption parameters to a polynomial length of 4096. However, it has two limitations: (1) The aggregation method supports only special cases with additive aggregation, restricting its applicability to diverse GNN types. (2) To reduce encryption parameters, PPGNN uses TFHE's PBS for nonlinear functions in GNNs, eliminating the need for CKKS bootstrapping with large encryption parameters. However, the non-SIMD nature of PBS limits it to activating a single data point, requiring numerous operations and causing a computational complexity explosion. For instance, for dataset Pubmed, PPGNN requires hundreds of thousands of PBS operations in just one iteration, specifically 19717 (vertices) $\times$ 32 (features). Thus, while PPGNN reduces memory overhead on the accelerator by optimizing parameters through software, its poor performance from computational complexity explosion demands extensive hardware resources to compensate. This unbalance significantly limits the benefits of its software-hardware co-design. An efficient privacy-preserving GNN acceleration system must adopt a balanced co-design with hardware-friendly parameters and reduced computational complexity.

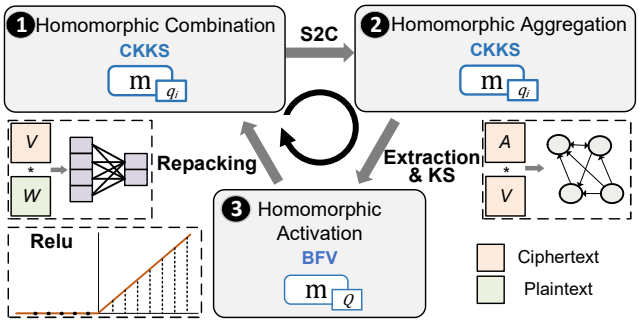


Fig. 3. Uranus framework. The three-step loop consecutively handles the key stages of GNN computation, i.e., combination, aggregation, and activation. Corresponding transition operations are required between each of the three steps to ensure the connection of execution flow.

IV. Uranus

A. Framework

As illustrated in Fig. 3, the GNN inference process in the Uranus framework involves three closely integrated steps: Homomorphic Combination, Homomorphic Aggregation, and Homomorphic Activation. These steps enable ciphertext-plaintext matrix multiplication (PMult) of node features and model weights, ciphertext-ciphertext matrix multiplication (CMult) of adjacency matrix and node features, and SIMD activation of node features. Each step corresponds to a specific FHE scheme with an associated ciphertext modulus. The CKKS scheme features a decreasing ciphertext modulus (q_i) as computation progresses, while the BFV scheme maintains a constant modulus (Q).

B. Procedures

1) Homomorphic Combination: The homomorphic combination operation (Step ① in Fig. 3) in GNN involves matrix multiplication between ciphertext features and plaintext weights. Unlike PPGNN, which employs coefficient encoding to avoid extensive rotations during matrix multiplication, we adopt a straightforward slot-based encoding approach combined with an optimized BSGS method [5] to perform this operation. Coefficient encoding distributes valid results across

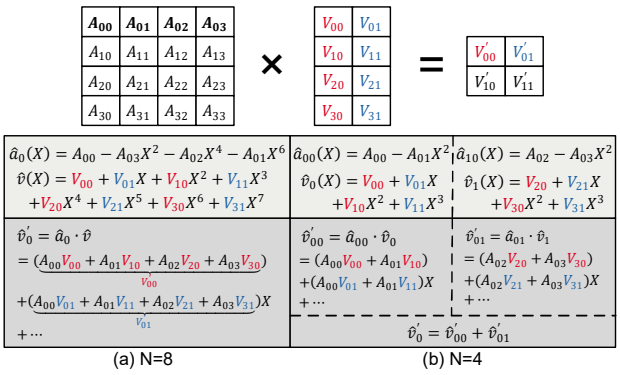


Fig. 4. Toy example of homomorphic aggregation.

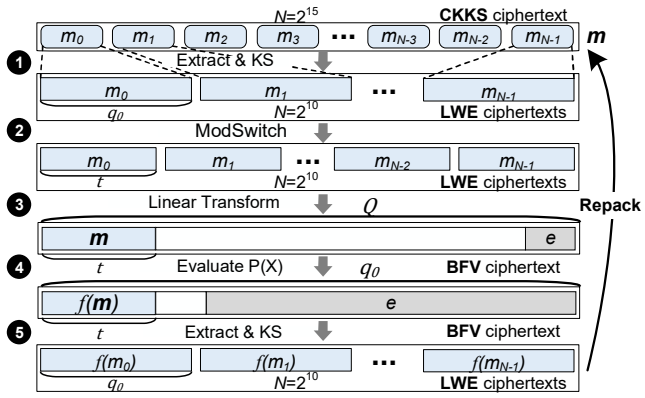


Fig. 5. Homomorphic activation process.

irregular polynomial coefficients, necessitating costly LWE-to-RLWE conversions [4] with substantial rotations. In contrast, our approach streamlines subsequent operations by eliminating these overheads.

After the combination operation, since the subsequent aggregation (step ②) involves MAC computations between ciphertexts, implementing it based on slots would introduce significant rotations. Therefore, we use S2C [12] to transfer the data to the coefficient space, leveraging the convolution property of polynomial multiplication for efficient implementation. Notably, the computational complexity of S2C ($O(\sqrt{N})$) is significantly lower than the LWE-to-RLWE conversion ($O(N)$) used in PPGNN.

2) Homomorphic Aggregation: In this operation, PPGNN employs the HMUX operator; however, this approach only supports simple additive aggregation and cannot handle weighted graphs. In contrast, we implement the operation based on standard matrix multiplication, which allows us to support regularization and weighted graphs (e.g., NELL [2]), offering greater practicality.

The design of the Uranus framework prioritizes both hardware friendliness and efficiency. In coefficient space-based aggregation, it is essential to optimize the distribution of result polynomial coefficients. Concentrating coefficients in the leading terms of the polynomial facilitates efficient hardware extraction

Formally, suppose the graph data input to a GNN layer has H nodes and F input features, then adjacency matrix $A \in \mathbb{R}^{H\times H}$ and the node features $V \in \mathbb{R}^{H\times F}$. For simplicity, consider the case when $H \times F \leq N$. After completing combination, node features are encoded in row-major order ($\hat{v}$ in (2)) in the coefficients. However, during aggregation, dot product is performed on each column of the node features, corresponding to an interval of F in the row-major order. Besides, to leverage the convolution properties of polynomials, the data involved in the operation must be arranged in reverse order. Thus, each row of the adjacency matrix should be encoded in reverse order and at intervals of F in the coefficients, as $\hat{a}_t$ shown in (2).

$$\begin{aligned} \hat{a}_t[N - i \times F \pmod N] &= A[t][i] \\ \hat{v}[i \times F + j] &= V[i][j] \end{aligned} \tag{2}$$

Fig.4(a) shows an example where $H = 4$ (nodes), $F = 2$ (features), and $N = 8$ (polynomial degree). According to the (2), the adjacency matrix is encoded as $\hat{a}_0$, $\hat{a}_1$, $\hat{a}_2$ and $\hat{a}_3$, and the node features as $\hat{v}$. During HAgg, $\hat{a}_0\cdot\hat{v}$, $\hat{a}_1\cdot\hat{v}$, $\hat{a}_2\cdot\hat{v}$ and $\hat{a}_3\cdot\hat{v}$ are computed respectively. Taking $\hat{a}_0 \cdot \hat{v}$ as exaple, it computes the first row of A (marked in bold) and the two columns of V (marked in red and blue), with the result distributed in the first two coefficients of the product polynomial. This encoding method distributes the effective results in the first F coefficients of the H polynomials of $\hat{a}_t \cdot \hat{v}$, minimizing extraction overhead.

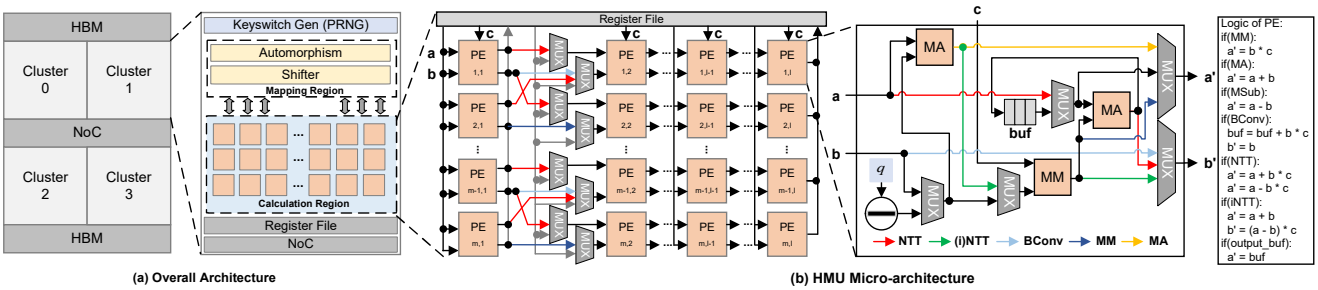
For cases when $H \times F > N$, the node features are distributed across multiple ciphertexts. Therefore, we need to partition the adjacency matrix accordingly to satisfy the aforementioned operational rules. We define a partition (H_c, F_c) for (H, F), such that $F_c = \lfloor \frac{N}{H} \rfloor$ and $H_c = \lfloor \frac{N}{F_c} \rfloor$ with $H_c \times F_c \leq N$. Since $H_c \times F_c < N$, the above operation can be repeated on each partition, and the results summed across the partitions. Fig.4(b) illustrates the computation process using the same H (4) and F (2), but with $N = 4$. Substituting into the above formula, each partition has $F_c = 2$ and $H_c = 2$, resulting in a total of $\lceil \frac{F}{F_c} \rceil \lceil \frac{H}{H_c} \rceil = 2$ partitions. Each partition completes the matrix multiplication of the submatrices. This partitioning approach ensures that results from different partitions can be merged using simple HAdd operation while retaining the property that effective results are distributed in the leading coefficients. This significantly reduces the overhead of merging and extraction.

3) Homomorphic Activation: Unlike PPGNN, which implements activation using PBS in a SISD manner, Uranus adopts the approach proposed in [20], introducing a unified homomorphic activation operator that supports high precision and SIMD computation. The core of this operation is to evaluate a polynomial (step ❹ in Fig. 5) as:

$$P(x) = f(0) - \sum_{i=0}^{t-1} x^i \sum_{k=0}^{t-1} f(k) k^{t-1-i} \tag{3}$$

Notably, this polynomial serves as an interpolation formula for $f(x)$ over the plaintext space Z_t of BFV, meaning that for any $x \in Z_t$, $P(x) = f(x)$. This computation format conforms to "table look-up," a.k.a. LUT, which allows for high-precision mapping for any activation function. Taking ReLU as example, $f(x)$ is substituted with $ReLU(x)$ into the formula and precomputes the corresponding interpolation polynomial, enabling inference by evaluating this polynomial.

Fig. 5 illustrates the specific process of the activation.



(a) Overall Architecture (b) HMU Micro-architecture

Fig. 6. Uranus accelerator architecture. (a) is the overall architecture, comprising four clusters interconnected by a NoC, each containing mapping regions and computation regions covering all operations in the Uranus framework. (b) is the micro-architecture of the HMU, the core computing unit in the accelerator, with high resource utilization as well as reconfigurable datapaths.

After aggregation, the results are distributed over certain coefficients of the polynomial. Therefore, Uranus first extracts these results as multiple LWE ciphertexts by Extraction (step ❶). Subsequently, ModSwitch (step ❷) and Linear Transform (step ❸) are performed to to pack LWE ciphertexts into the plaintext space of BFV ciphertext. By evaluating the polynomial $P(x) = f(x)$, $Enc(m_i)$ will be mapped to $Enc(f(m_i))$. Then, S2C and extraction (step ❹) are performed to extract LWE ciphtexts from BFV ciphertext. Among these steps, sample extraction is an operation that is unique to the multi-FHE scheme and requires a special acceleration unit, for which we have designed a special hardware unit, Shifter, which will be introduced in Section V-B2.

After activation, LWE ciphertexts are repacked according to the order required by combination using the packing method proposed in PEGASUS [21]. After repacking, the data are in the slot space of the ciphertext and returns to the initial step of framework for the next round of inference computation.

C. Parameter Optimization and Selection

The main objective of the Uranus framework is to optimize the encryption parameters for hardware friendliness. The Uranus framework is implemented by integrating linear layer computations based on CKKS coefficient encoding and batch nonlinear function evaluation based on BFV (simultaneously achieving bootstrapping). Unlike traditional CKKS implementations, which separate nonlinear operations and bootstrapping and introduce significant multiplicative depth (thus requiring large encryption parameters such as N=65536 and logQ $>$ 1500 bits), Uranus framework supports full GNN inference with much smaller parameters.

Furthermore, in contrast to the PPGNN accelerator, which primarily reduces encryption parameters through an unbalanced hardware-software co-design (where the non-linear layer operates in SISD mode), Uranus adopts a SIMD approach for both linear and non-linear layer computations, significantly reducing computational complexity. As a result, despite using slightly larger parameters than PPGNN, Uranus achieves substantially better overall performance, as demonstrated in the performance evaluation section (Section VI). The optimized parameter settings in Uranus are as follows: The LWE degree $n = 1024$, and modulus $q = 65537$; In BFV ciphertext, the degree $N = 32768$, plaintext modulus $t = 65537$ (the same as the LWE modulus), and the ciphertext modulus is 765 bits; In CKKS ciphertext, the degree $N = 32768$ and the ciphertext modulus is 675 bits, which provides a rescale level $L = 15$. These parameter settings guarantee a security level of 119 bits (same as PPGNN).

V. Uranus Accelerator

A. Overall Architecture

Fig. 6(a) illustrates the Uranus accelerator's overall architecture, consisting of four clusters interconnected via a network-on-chip (NoC). Each cluster is divided into a *mapping region* and a *calculation region*. The mapping region handles data indexing operations in the Uranus framework, such as Automorphism for rotation (*Auto core*) and Shifting for extraction (*Shifter core*). The calculation region accelerates FHE modular computations, including number theoretic transformation (NTT), modular multiplication (MM), modular addition (MA), and base conversion (BConv). Unlike conventional CKKS-based GNN implementations where NTT and BConv dominate computation, Uranus introduces additional overhead from MA and MM due to polynomial evaluations in activation (step ❹ in Fig. 5), which require numerous ciphertext multiplications and additions. Since NTT and BConv are both composed of MA and MM, to improve the hardware utilisation, we design the reconfigurable multifunctional unit HMU to support all homomorphic modular operations by resource reusing. The specific micro-architecture will be introduced in Section V-B.

For memory, the Uranus accelerator integrates HBM and scratchpad. HBM manages data exchange with the host, while scratchpad caches intermediate results and supports computations in functional regions. All the cores have their own input and output buffers, and the scratchpad is connected to the buffer via NoC. Each cluster has a data parallelism of 512 and the total parallelism of the accelerators is 2048. Compared to recently proposed architectures like ARK or CraterLake, Uranus avoids large scratchpad memory, as its optimized algorithm framework requires smaller encryption parameters, reducing scratchpad capacity and bandwidth demands by 4×.

B. Microarchitecture

1) HMU: The HMU efficiently leverages shared modulo addition and modulo multiplication units to perform the four core FHE operations: NTT, MA, MM and BConv. As illustrated in Fig. 6(b), the HMU comprises an array of PEs. Each PE array is reconfigurable, with its data paths dynamically adjusted to support various operations such as NTT, MA,

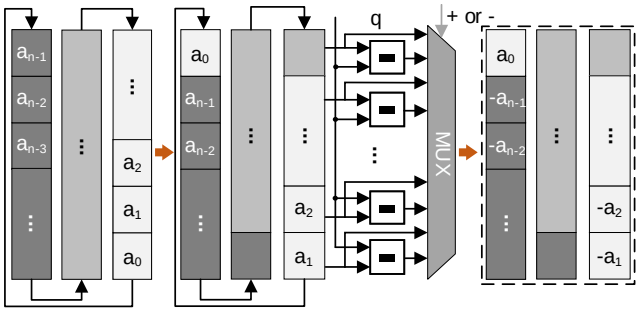


Fig. 7. The Shifter micro-architecture.

MM, and BConv. For example, when performing NTT, the data path between PEs forms a butterfly operation (red line), where each PE executes a two-input butterfly computation. The same configuration, but for BConv in accordance with the proposal put forth by ARK, a continuous transfer of data is conducted from one PE to the next along the rows of PEs. This involves the sharing of one input data set between the PEs in different rows, with no data interaction between the rows themselves. No data interaction occurs between the rows of the PE array (blue line). The data paths of MA and MM are similarly reconfigured and selected in a similar manner (gray line). In addition to the control of the inter-PE data paths, the internal PEs in the HMU must be designed with configurable computational logic to support different computational modes. As illustrated in Fig. 6(b), each PE comprises one modulo multiplication unit and two modulo addition units. Each PE is equipped with three input ports and two output ports, enabling support for up to four distinct operations. For instance, the NTT configuration permits all five of the data ports to be utilized, whereas in the BConv configuration, two input ports and one output port are operational. In each cluster, the dimensions of the PE array are 256 rows (m) and 5 columns (l). Its parallelism is 512, and it is capable of completing five iterations at a time in NTT computation. By means of a two-level reconfigurable design between and within PEs, the computational resources within HMU can be fully utilized in the four different computational modes, thereby markedly increasing the efficiency of hardware utilization and reducing the impact of computational congestion in Uranus.

2) Shifter and Auto Cores: Uranus's Shifter unit converts RLWE ciphertexts into LWE ciphertexts, i.e., extraction (step ❶ in Fig. 5), with minimal computation but significant data relocation. As shown in (4), to extract $ct_i(lwe)$, we need to cyclically shift all coefficients of the ciphertext term a of RLWE by $i+1$ positions and negate the $(i+1)$-th to N-th element. While the barrel shifter implements this process with $O(log_2 N)$ complexity, it becomes inefficient for frequent continuous shifts. In Uranus, we try to fill the slots of a single ciphertext to the maximum with the results of the homomorphic combination and aggregation (step ① and ② in Fig. 3). This requires close to N extraction operations for that ciphertext. Thus, the Shifter unit (Fig. 7) employs a straightforward register shifter. Each cycle shifts data by one

$$\vec{a}_i[j] = \begin{cases} \mathbf{a}_{i-j}, & j \le i, \\ -\mathbf{a}_{i-j}, & j > i. \end{cases} \tag{4}$$

position, selecting elements as positive or negative based on identification signals. In addition, since the input data must be in the corresponding modulus, only the subtractor is required for a negative value. In this way, the average cycle of an extraction operation is close to 1. The Automorphism unit primarily handles index mapping with minimal computational demands. We adopt the optimized design in ARK [17] to implement the Auto core. Our framework, Uranus, has a fixed polynomial degree N of 2^{15}, and the parallelism a single Auto core is 256. We deploy 8 cores to unify the total parallelism of 2048.

VI. Evaluation

A. Experimental Setup

Platform. The major logic units of Uranus are implemented in RTL using the ASAP7 7.5-track 7nm predictive process design kit (PDK) [9], and the SRAM components are evaluated by a cache modeling tool FinCACTI [28]. The area and power of two HBM modules are estimated based on prior work [14], [22]. We measure the benchmark runtime with a cycle-level simulator, and use the activity-level energy consumption from the synthesized components for energy evaluation.

Additionally, it is important to note that, to ensure a fair comparison, all other accelerators are normalized to a 7nm technology node (including both area and power) based on CMOS device performance scaling equations [29], as they were originally implemented using different technology nodes.

Baseline. We employ the CKKS implementation CryptoGCN [23] on the top of the SEAL library [27] and simulate its performance on the SOTA CKKS FHE accelerator F1 [25], F1+ [26], CraterLake [26], and SHARP [16] as our CKKS baseline. Combination and aggregation adopt homomorphic matrix-vector multiplications with diagonal encoding [15], while ReLU uses a high-degree polynomial approximation [19]. In addition, PPGNN [31] is also selected for the software-hardware co-design comparison with Uranus. The high-performance prototype of PPGNN (referred to as the "Total 2" version in the original paper) is reported to be implemented using a 14nm process and operates at 475 MHz. For fair comparison, in addition to scaling its chip area and power to 7nm, we also scale its operating frequency, matching that of our Uranus accelerator.

GNN Model and Datasets. As shown in Table II, we selected widely-used datasets: Cora, Citeseer, and Pubmed. We employ a three-layer GNN architecture with hidden layer dimensions set to 32, 16, and 16.

B. Performance

1) Accuracy: We evaluate the inference accuracy of Uranus on different datasets. As shown in Table II, the encrypted accuracy is only 0.2%-0.7% lower than the plaintext, which is much smaller than the CKKS-based GNN inference implementation CryptoGCN [23]. BFV-based lookup table in Uranus's activation layer ensures high accuracy, avoiding the errors introduced by CKKS's polynomial fitting approach. In addition, although PPGNN does not report accuracy results

TABLE II
DATASET INFORMATION AND ACCURACY.

Datasets	Vertex	Edge	#Feature	Plain (%)	Cipher (%)
Cora	2,708	10,556	1,433	81.9	**81.7**
Citeseer	3,327	9,104	3,703	70.2	**69.5**
Pubmed	19,717	88,648	500	79.0	**78.7**

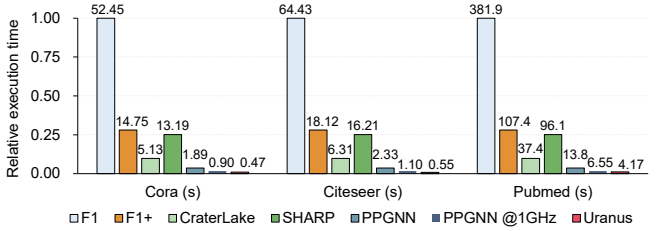


Fig. 8. Performance comparison with prior accelerators. We use relative execution times to better demonstrate the performance of GNN models at different scales on the accelerators, normalized to F1.

and we are unable to make accuracy comparisons, the number of bits of high accuracy it achieves based on the TFHE is only 11 bits while Uranus has 16 bits of high accuracy, indicating that Uranus likely achieves comparable or superior inference accuracy to PPGNN.

2) Speedup: Fig. 8 shows the performance comparison between Uranus and other CKKS accelerators (F1, F1+, CraterLake and SHARP) as well as PPGNN, where the CKKS accelerator runs a CKKS-based implementation of GNN. Uranus gets over 92× speedup over F1 and over 9.0× speedup over CraterLake, which is the best CKKS accelerator that works on GNN. Compared to GNN-oriented PPGNN, Uranus demonstrates 3.3×-4.2× performance improvement. To ensure a fair comparison under identical operating conditions, we also evaluate PPGNN scaled to Uranus' 1GHz frequency. Even under this normalized scenario, Uranus retains a significant 1.6×-2.0× advantage, attributed to its efficient matrix multiplication and homomorphic activation based on BFV-based SIMD, which greatly reduces the computational complexity compared to PPGNN. It is also noteworthy that at this performance, the area power consumption of Uranus is also much smaller than these accelerator schemes, as detailed in Section VI-D.

Fig. 9 illustrates the execution time breakdown of Uranus. It is evident that Aggregation, Extraction, and Activation dominate the runtime across the three GNN models and datasets, primarily due to their high computational complexity. Aggregation is implemented via matrix multiplication, which involves a large number of expensive homomorphic rotations, making it the most computationally intensive step. Notably, Aggregation accounts for up to 29% of the execution time on the Pubmed dataset, mainly because of its large number of vertex (up to 19,717 as shown in Table II), which results in larger matrices and increased computation. Extraction converts all data from RLWE ciphertexts into LWE ciphertexts and performs individual key switching operations. Since its complexity scales with the number of valid data elements in the ciphertext, it also incurs a significant overhead. Activation involves evaluating high-degree polynomial functions on encrypted data, requiring numerous ciphertext multiplications and additions. This makes it another major computational

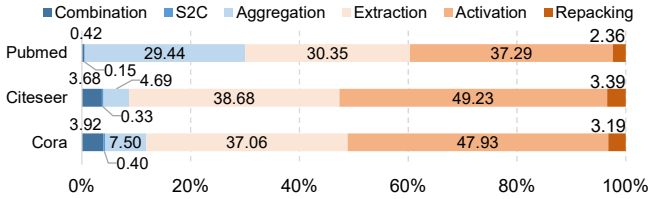


Fig. 9. Execution time breakdown. The breakdown consists of six key operations, which include the three core steps of the Uranus framework: Combination, Aggregation, and Activation, as well as the transition and transformation steps between them, i,e., S2C, Extraction, and Repacking.

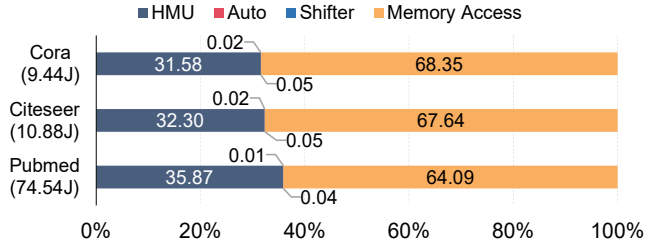


Fig. 10. Energy consumption and breakdown.

bottleneck in the Uranus framework. In contrast, the operations Combination, S2C, and Repacking contribute relatively little to the overall execution time. This is because the coefficient-encoded Combination only requires a single ciphertext multiplication, and both S2C and Repacking are essentially matrix-vector multiplications, which are efficiently handled using the optimized BSGS algorithm and thus incur low computational overhead.

C. Energy

1) Energy Consumption and breakdown: In addition to performance, we evaluate the full-system energy consumption of the Uranus accelerator when running the four selected GNN models. As shown in Fig. 10, memory access accounts for approximately 65% of the energy consumption. Among the compute units, HUM is the dominant contributor, whereas Automorphism and Shifter contribute only minimally. This is because HMU, serving as the core computational engine in Uranus, is highly reconfigurable and supports a wide range of operations, including NTT, BConv, MM, and MA. These operations constitute the majority of homomorphic computations, making HUM responsible for most of the workload in GNN execution and thus the primary source of energy consumption. In contrast, Automorphism and Shifter are mainly implemented through data storage structures and control logic for memory access. As they involve limited arithmetic computation, their energy consumption remains significantly lower.

2) Energy Efficiency: We use energy-delay-area product (EDAP) as the metric of energy efficiency. As shown in Table III, Uranus demonstrates exceptional energy efficiency, achieving over 119× improvement compared to SOTA PPGNN and surpassing other CKKS accelerators by more than 4 orders of magnitude. This is due to Uranus' low-complexity, hardware-friendly inference framework and highly reconfigurable hardware architecture designed specifically for GNN. The Uranus framework imposes minimal requirements on encryption parameters, which significantly reduces the memory volume demand in accelerator design and consequently lowers

TABLE III

EFFICIENCY ANALYSIS. WE USE ENERGY DELAY AREA PRODUCT (EDAP) IN $J \cdot s \cdot mm^2$ AS THE METRIC. LOWER IS BETTER.

	Cora	**Citeseer**	**Pubmed**
F1 [25]	27,233,879	41,107,190	1,443,756,190
F1+ [26]	5,162,087	7,791,725	273,658,970
CraterLake [26]	1,446,949	2,184,046	76,707,487
SHARP [16]	2,350,332	3,547,624	124,598,733
PPGNN [31]	36,137	54,546	1,915,766
Uranus	**230.0**	**311.7**	**16,138**

TABLE IV

AREA AND POWER BREAKDOWN (@1 GHZ, 7NM).

Component	**Area** $[mm^2]$	**Peak Power** $[W]$
Automorphism	1.9	1.5
PRNG	1.2	1.9
HMU	7.5	6.5
Shifter	0.16	0.59
NoC	2.9	3.9
Register Files (6MB)	3.4	1.96
SRAM (45MB)	20.1	4.8
HBM (HBM2E $\times$1)	14.8	15.9
Sum	**51.96**	**37.05**
F1	71.2 (7nm)	$\sim$117
F1+	170.6 (7nm)	/
CraterLake	222.7 (7nm)	$\sim$207
SHARP	178.8	/
PPGNN	119.0 (7nm)	$\sim$87

both chip area and power consumption (detailed in Section VI-D). Furthermore, the highly reconfigurable HUM design enables efficient support for diverse computations, leading to improved hardware utilization. These advantages in area and resource efficiency collectively contribute to the excellent energy efficiency of Uranus.

D. Area and Power

We implement the Uranus accelerator in RTL and synthesize it in a commercial 7nm technology node using state-of-the-art tools, including the SRAM compiler. As shown in Table IV, With a size of 51.96mm^2 and power consumption of 37.05W, Uranus is significantly more compact and power-efficient than SOTA FHE accelerators, including GNN-optimized designs like PPGNN. HMU is the compute unit with the highest hardware overhead because it is designed for all the modulo operation with high-overhead in Athena. Meanwhile, Uranus accelerator has smaller on-chip storage and HBM requirements compared to the CKKS accelerators, benefiting the smaller size of ciphertext. In addition, Although PPGNN consumes less memory, its activation layer based on the TFHE via SISD approach requires a large number of HLUT units and thus introduces a significant hardware overhead, resulting in more than 2$\times$ the area/power consumption. This also demonstrates the strength of Uranus' software-hardware co-design.

E. Ablation Study

Uranus is a co-designed GNN acceleration system in which the software framework and hardware accelerator are optimized collaboratively, rather than being developed independently. As a result, the Uranus framework requires a dedicated hardware accelerator to fully realize its performance potential. To validate this, we deploy the Uranus software framework on other accelerators for performance evaluation. It is important to note that the PPGNN hardware architecture supports only very limited hardware parameters and offers extremely constrained storage resources, making it incompatible with the Uranus framework. Therefore, we select CKKS-based accelerators as comparison targets. They lack Shifter units for the extraction operation of Uranus, we assume that identical Shifter units are incorporated into them to ensure a fair comparison.

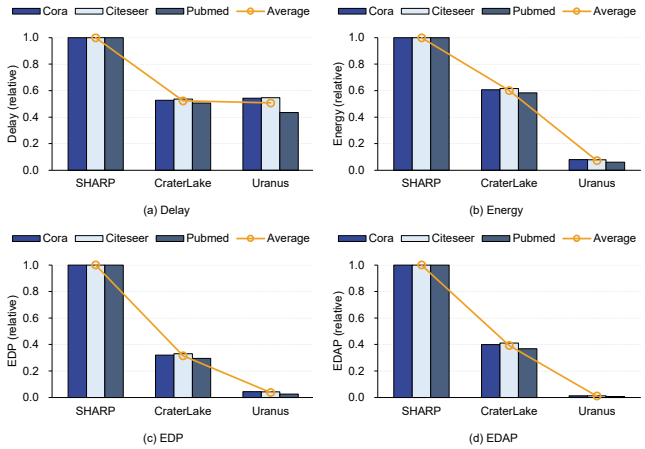


Fig. 11. We deploy the Uranus framework on the existing FHE accelerators and the Uranus accelerator. Since Uranus differs from CKKS-based implementations, existing accelerators are not suitable for Uranus. For the Uranus-specific modules HMU we use MA/MM and RNSConv to substitute in SHARP and CraterLake.

Fig. 11 presents the performance of Uranus framework when running on SHARP and CraterLake, including metrics such as latency, energy consumption, EDP, and EDAP. The results are obtained using a cycle-accurate simulator extended from SimFHE [1]. As shown, Uranus significantly outperforms SHARP in terms of latency and achieves performance comparable to CraterLake, while occupying only 30% of SHARP's chip area and 23.3% of CraterLake's. In terms of energy, EDP, and EDAP, Uranus also consistently outperforms both state-of-the-art CKKS accelerators. The main reason is that these accelerators only focus on NTT and RNSConv, which are known bottlenecks in CKKS applications, and cannot effectively accommodate the computational characteristics of Uranus, such as LUT-based homomorphic activation, resulting in suboptimal hardware utilization and limited performance scalability.

VII. CONCLUSION

In this paper, we propose Uranus, a software-hardware co-designed acceleration architecture of secure GNN inference, including a hardware-friendly inference framework and the design of hardware accelerators to match the framework. Based on CKKS and BFV, Uranus implements an efficient nonlinear layer and a high-precision nonlinear layer capable of batch table lookup, and addresses the computational bottleneck of the algorithmic framework by improving hardware utilization efficiency through highly reconfigurable hardware computation units, thus achieving far better performance than the SOTA accelerator in GNN secure inference.

REFERENCES

[1] Rashmi Agrawal, Leo De Castro, Chiraag Juvekar, Anantha Chandrakasan, Vinod Vaikuntanathan, and Ajay Joshi. Mad: Memory-aware design techniques for accelerating fully homomorphic encryption. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 685–697, 2023.

[2] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam Hruschka, and Tom Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 24, pages 1306–1313, 2010.

[3] Hervé Chabanne, Roch Lescuyer, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. Recognition over encrypted faces. In *Mobile, Secure, and Programmable Networking: 4th International Conference, MSPN 2018, Paris, France, June 18-20, 2018, Revised Selected Papers 4*, pages 174–191. Springer, 2019.

[4] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient homomorphic conversion between (ring) lwe ciphertexts. In *International Conference on Applied Cryptography and Network Security*, pages 460–479. Springer, 2021.

[5] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. 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, 2018 Proceedings, Part I 37*, pages 360–384. Springer, 2018.

[6] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[7] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *international conference on the theory and application of cryptology and information security*, pages 3–33. Springer, 2016.

[8] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *arXiv preprint arXiv:1811.09953*, 2018.

[9] Lawrence T Clark, Vinay Vashishtha, David M Harris, Samuel Dietrich, and Zunyan Wang. Design flows and collateral for the asap7 7nm finfet predictive process design kit. In *2017 IEEE international conference on microelectronic systems education (MSE)*, pages 1–4. IEEE, 2017.

[10] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*, 28, 2015.

[11] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.

[12] Kyoohyung Han, Minki Hhan, and Jung Hee Cheon. Improved homomorphic discrete fourier transforms and fhe bootstrapping. *IEEE Access*, 7:57361–57370, 2019.

[13] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189*, 2017.

[14] Norman P Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, et al. Ten lessons from three generations shaped google's tpuv4i: Industrial product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–14. IEEE, 2021.

[15] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In *27th USENIX security symposium (USENIX security 18)*, pages 1651–1669, 2018.

[16] Jongmin Kim, Sangpyo Kim, Jaewan Choi, Jaiyoung Park, Donghwan Kim, and Jung Ho Ahn. Sharp: A short-word hierarchical accelerator for robust and practical fully homomorphic encryption. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–15, 2023.

[17] Jongmin Kim, Gwangho Lee, Sangpyo Kim, Gina Sohn, John Kim, Minsoo Rhu, and Jung Ho Ahn. Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse. *arXiv preprint arXiv:2205.00922*, 2022.

[18] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In *International Conference on Machine Learning*, pages 12403–12422. PMLR, 2022.

[19] Seewoo Lee, Garam Lee, Jung Woo Kim, Junbum Shin, and Mun-Kyu Lee. Hetal: efficient privacy-preserving transfer learning with homomorphic encryption. In *International Conference on Machine Learning*, pages 19010–19035. PMLR, 2023.

[20] Zeyu Liu and Yunhao Wang. Amortized functional bootstrapping in less than 7 ms, with $\tilde{o}(1)$ polynomial multiplications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 101–132. Springer, 2023.

[21] Wen-jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. Pegasus: bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1057–1073. IEEE, 2021.

[22] Mike O'Connor, Niladrish Chatterjee, Donghyuk Lee, John Wilson, Aditya Agrawal, Stephen W Keckler, and William J Dally. Fine-grained dram: Energy-efficient dram for extreme bandwidth systems. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 41–54, 2017.

[23] Ran Ran, Wei Wang, Quan Gang, Jieming Yin, Nuo Xu, and Wujie Wen. Cryptogcn: Fast and scalable homomorphically encrypted graph convolutional network inference. *Advances in Neural information processing systems*, 35:37676–37689, 2022.

[24] Ran Ran, Nuo Xu, Tao Liu, Wei Wang, Gang Quan, and Wujie Wen. Penguin: parallel-packed homomorphic encryption for fast graph convolutional network inference. *Advances in Neural Information Processing Systems*, 36, 2024.

[25] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald Dreslinski, Christopher Peikert, and Daniel Sanchez. F1: A fast and programmable accelerator for fully homomorphic encryption. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 238–252, 2021.

[26] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sanchez. Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data. In *ISCA*, pages 173–187, 2022.

[27] Microsoft SEAL (release 4.0). https://github.com/Microsoft/SEAL, March 2022. Microsoft Research, Redmond, WA.

[28] Alireza Shafaei, Yanzhi Wang, Xue Lin, and Massoud Pedram. Fincacti: Architectural analysis and modeling of caches with deeply-scaled finfet devices. In *2014 IEEE Computer Society Annual Symposium on VLSI*, pages 290–295. IEEE, 2014.

[29] Aaron Stillmaker and Bevan Baas. Scaling equations for the accurate prediction of cmos device performance from 180 nm to 7 nm. *Integration*, 58:74–81, 2017.

[30] Xuemei Wei, Yezheng Liu, Jianshan Sun, Yuanchun Jiang, Qifeng Tang, and Kun Yuan. Dual subgraph-based graph neural network for friendship prediction in location-based social networks. *ACM Transactions on Knowledge Discovery from Data*, 17(3):1–28, 2023.

[31] Yuntao Wei, Xueyan Wang, Song Bian, Yicheng Huang, Weisheng Zhao, and Yier Jin. Ppgnn: Fast and accurate privacy-preserving graph neural network inference via parallel and pipelined arithmetic-and-logic fhe accelerator. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, DAC '24, New York, NY, USA, 2024. Association for Computing Machinery.

[32] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983, 2018.