PPGNN: Fast and Accurate Privacy-Preserving Graph Neural Network Inference via Parallel and Pipelined Arithmetic-and-Logic FHE Accelerator

Yuntao Wei
Beihang University
Beijing, China

Xueyan Wang*
Beihang University
Beijing, China

Song Bian*
Beihang University
Beijing, China

Yicheng Huang
Beihang University
Beijing, China

Weisheng Zhao
Beihang University
Beijing, China

Yier Jin*
University of Science and Technology of China
Hefei, China

ABSTRACT

Graph Neural Networks (GNNs) are increasingly used in fields like social media and bioinformatics, promoting the prosperity of cloud-based GNN inference services. Nevertheless, data privacy becomes a critical issue when handling sensitive information. Fully Homomorphic Encryption (FHE) enables computations on encrypted data, while privacy-preserving GNN inference generally necessitates ensuring graph structure data confidentiality and maintaining computation precision, both of which are computationally expensive in FHE. Existing schemes of GNNs inference with FHE are deterred by either computational overhead, accuracy degradation, or incomplete data protection. This paper presents PPGNN to address these challenges all at once. We first propose a novel privacy-preserving GNN inference algorithm utilizing a high-accuracy arithmetic-and-logic FHE approach, meanwhile only need much smaller parameters, substantially reducing computational complexity and facilitating parallel processing. Correspondingly, a dedicated hardware architecture has been designed to implement these innovations, with featured specialized units for arithmetic and logic FHE operations in a pipelined manner. Collectively, PPGNN achieves 2.7× and 1.5× speedup over state-of-the-art Arithmetic FHE and Logic FHE accelerators while ensuring high accuracy, simultaneously with about 18× energy reduction on average.

KEYWORDS

Graph neural networks, Privacy-preserving, Fully homomorphic encryption, Hardware-software co-design

*Corresponding Authors: Xueyan Wang, Song Bian and Yier Jin (wangxueyan@buaa.edu.cn, sbian@buaa.edu.cn, jinyier@ustc.edu.cn). This work was supported by National Key R&D Program of China (2023YFB3106200), Beihang Frontier Cross-Fund Project, the National Natural Science Foundation of China (NO. 62004011, NO. 62202028), Young Elite Scientists Sponsorship Program by CAST (2023QNRC001), Huawei Technologies Co., Ltd., and State Key Laboratory of Integrated Chips and Systems.

DAC '24, June 23–27, 2024, San Francisco, CA, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3656517

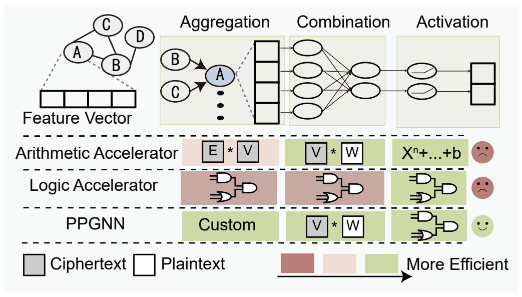


Figure 1: Comparison of GNNs operations with Arithmetic, Logic, and PPGNN accelerators.

1 INTRODUCTION

Graph Neural Networks (GNNs) are applied in diverse fields such as social network analysis, bioinformatics, and recommendation systems [1–3]. Platforms like PaddleHelix [4] enable clients to perform server-side GNN inference by leveraging high-precision models of server. However, data privacy concerns arise when dealing with sensitive information, such as personal sensitive information in social networks [1], confidential medical data in bioinformatics [2], or transactional data in finance [5], necessitating privacy protection.

Fully Homomorphic Encryption (FHE) enables secure computations on encrypted data, and it has been applied in privacy-preserving machine learning domain [6–10]. CryptoGCN [11] explores to apply FHE in GNNs. It focus on skeleton-based action recognition application, in which only node attributes are encrypted while graph structures are exposed to untrusted servers. Besides, it uses second-degree polynomial to approximate the non-linear ReLU function, resulting in a significant accuracy reduction and necessitating a re-training process. Therefore, CryptoGCN would encounter both security and accuracy issues when extended to more general scenarios, highlighting the need for an FHE-based GNN solution that ensures comprehensive privacy while maintaining high accuracy and moderate computational complexity. However, it is ultra challenging to achieve these goals. On the one hand, graph aggregation shown in Figure 1 is an essential process in GNNs to gather node's neighbor information, which would be extraordinary computational expensive then the graph structures (edges) are encrypted. On the other hand, accurate non-linear computations require either high-degree polynomial approximations in Arithmetic FHE schemes or Homomorphic Look-up Tables (HLUT) in Logic FHE schemes, both intensifying the computational load.

Hardware acceleration strategies have been extensively explored, aiming to address these computational overhead challenges in FHE.

As shown in Figure 1, state-of-the-art FHE accelerators primarily focus on either Arithmetic FHE schemes like CKKS [12] or Logic FHE schemes such as TFHE [13]. In GNNs, linear computations like aggregation and combination are compatible with Arithmetic FHE schemes, while precise non-linear computations would necessitate high-degree polynomial approximations, for example, 343-degree polynomial is used in [14]. This results in much larger FHE parameters, and even for the simplest three-layer GNNs, it potentially increases the ring dimension from 8192 to 65536. In turn, large parameters significantly increase the computational overhead of aggregation operations by 22× [15]. Alternatives like Crater-Lake [16] face a lower 5× computational overhead increase at the cost of $158.8mm^2$ additional area overhead of the special hardware unit for adopting optimized algorithm. Moreover, encrypted graph edge data demand an extra 512GB/s off-chip memory bandwidth for a full-pipelined aggregation, posing considerable challenges to overall system design and efficiency.

Utilizing Logic FHE schemes to implement linear operations such as combination and aggregation is inefficient, as it involves converting multi-bit multiplications into complex logic circuits and lacks of support for data packing as in Arithmetic FHE [9]. In fact, executing a single 32-bit multiplication in Logic FHE can take up to 333ms [17]. However, Logic FHE schemes are able to implement accurate non-linear operations by using Homomorphic Look-Up Table (HLUT) with small parameters (such as a ring dimension of 4096). This motivates us to develop an Arithmetic-and-Logic FHE scheme. Specifically, linear operations like aggregation and combination are executed via Arithmetic FHE schemes, while non-linear operations are handled through HLUT in Logic FHE. As such, Arithmetic-and-Logic FHE does not increase the overhead of linear operations compared to purely Arithmetic FHE schemes. And although HLUT is computation-intensive and becomes a new performance bottleneck, hardware acceleration potential for HLUT in non-linear functions of GNNs is promising. Due to the fact that the large constant auxiliary data involved in HLUT constitutes over 99% of total memory access and these auxiliary data can be reused by all HLUT units, increasing the number of parallel HLUT units could gain linear performance improvements. Moreover, GNNs' non-linear functions are highly parallel due to the vast number of nodes in real graphs, therefore, large-scale parallel acceleration could be enabled.

Based on these observations, we propose PPGNN, a accelerator for efficient and accurate privacy-preserving GNN inference. Our primary approach is a hardware-software co-design, starting with an Arithmetic-and-Logic FHE GNNs scheme that concentrates most computational demands on the Logic FHE side, and then speeds up the scheme with a parallel and pipelined hardware architecture. Specifically, the contributions are summarized as follows:

- **A hardware-optimized Arithmetic-and-Logic FHE GNNs scheme**, which is facilitated by a customized Homomorphic Aggregation method and allows end-to-end execution under a 64-bit modulus. It streamlines linear operations and concentrates computational overhead mainly on the HLUT operations of the Logic FHE side.
- **A parallel and pipelined hardware architecture.** We develop Logic FHE engine with massive HLUT units for computation-intensive non-linear operations, and lightweight Arithmetic FHE engine for memory-intensive linear operations. We further propose data flow optimizations to ensure pipeline execution and higher throughput.
- **Comprehensive experiments.** PPGNN achieves 2.7× over the state-of-the-art Arithmetic FHE accelerator [16], and achieves 1.5× over Logic FHE accelerators [18] while ensuring high accuracy, simultaneously with about 18× energy reduction on average.

2 PRELIMINARY

2.1 Graph Neural Network

GNN computations consist of three primary phases: the combination phase, aggregation phase, and activation phase [19], defined as follows:

$$\bar{h}_v^k = h_v^{k-1} W \tag{1}$$

$$h_v^k = \sigma(\text{AGGREGATE}\left(\bar{h}_u^{k-1} | u \in \mathcal{N}(v)\right)) \tag{2}$$

where h_v^k denotes the updated feature of node v, and $\sigma(\cdot)$ signifies an activation function. Typically, aggregation functions include operations such as summation, maximization, or averaging. In this paper, summation is chosen for its simplicity. Thus, the comprehensive GNN computation is represented as:

$$X^k = \sigma(AX^{k-1}W) \tag{3}$$

where A is adjacency matrix, and X is feature matrix, comprising the feature vectors of all nodes.

2.2 Fully Hormorphic Encryption

2.2.1 Ciphertext. **Arithmetic Ciphertext** encrypts a message polynomial $\tilde{m}$ using a key polynomial $\tilde{s}$, resulting in a tuple $(\tilde{b}, \tilde{a})$, where $(\tilde{b}, \tilde{a}) = (\tilde{m} + \tilde{e} - \tilde{s} \cdot \tilde{a}, \tilde{a})$. This tuple consists of the message polynomial altered by an error polynomial $\tilde{e}$ and a randomly chosen polynomial $\tilde{a}$.

Logic Ciphertext encrypts an integer message m using a secret vector s, producing a ciphertext pair (b, a). The component b is calculated as $b = m + e - \langle a, s \rangle$, where a is a vector of uniformly chosen integers, and e is selected from an error distribution. To decrypt an LWE ciphertext, one computes $b + \langle a, s \rangle$, which should closely approximate the original message m.

2.2.2 Operations. **Homomorphic Multiplexer (HMUX)** evaluates a 2-to-1 multiplexer on two Arithmetic Ciphertext homomorphically. There is one RGSW Ciphertext as control inputs of multiplexer. It can be implement as follow:

$$c = a - (a - b) \odot S \tag{4}$$

where a and b are Arithmetic Ciphertext and S is RGSW Ciphertext. This process can be represented as $c = \text{HMUX}(a, b, S)$. The decrypted value of S can be 0 or 1, corresponding to c being a or b, respectively. The $\odot$ is a external product [13] and occupies most of the HMUX computation time.

Homomorphic Look-Up Table (HLUT) evaluation can be viewed as evaluating a LUT function on ciphertext [20]. For a given function $f(x)$, the LUT in FHE is represented by a polynomial $p(x) = \sum_{i=0}^{n} f(i)x^i$. Given an input i, to evaluate this LUT, we compute $x^{-i} * p(x)$ homomorphicly. As a result, $f(i)$ is obtained from the constant term of the resulting polynomial. The polynomial $p(x)$ is called test vector.

To perform HLUT on Logic Ciphertext (b, a), this process is implemented as a sequence of HMUX operations. Specifically,

$$\begin{aligned} &\mathrm{acc}_0 = p(x) * x^{-b}, \\ &\mathrm{acc}_{i+1} = \mathrm{HMUX}(\mathrm{acc}_i, \mathrm{acc}_i * x^{-a_i}, BK_i), i = 0, 1, \ldots, n-1 \end{aligned} \tag{5}$$

where bootstrapping key BK is RGSW encryption of secret key s, and n is the length of s. As the decryption of (b, a) is $m \approx b + \langle a, s \rangle$, we can get LUT result of $f(m)$ from the constant coefficient of decrypted acc_n. Consequently, the predominant computational overhead of the HLUT hinges on the external product.

3 MOTIVATION

Figure 1 illustrates our proposed categorization of potential FHE schemes into Arithmetic FHE, Logic FHE, or Arithmetic-and-Logic FHE types. Given the impracticality of purely Logic FHE schemes for GNNs, our focus is on exploring the performance enhancement prospects of Arithmetic FHE and Arithmetic-and-Logic FHE schemes when integrated with hardware acceleration.

For Arithmetic FHE scheme, accurate non-linear computations require high-degree polynomial approximations, such as a 343-degree polynomial for sign function approximation [14], resulting in at least 12 multiplication depths per ReLU operation. Consequently, a simple three-layer GNN model could necessitate over 42 multiplication depths, inflating FHE parameters, computational costs, and ciphertext size. The Key-Switching (KS) operation, crucial for aggregation, becomes a significant bottleneck in accelerators like F1 [15], facing a 22× computation time increase due to quadratic overhead scaling. While accelerators like CraterLake [16] and SHARP [21] address this issue by employing linearly scaling KS algorithms [22], they require substantial hardware for Change-of-RNS-base operations, consuming 158.8mm² of space in CraterLake's layout. Additionally, larger FHE parameters cause increased ciphertext sizes, further exacerbating memory demands, especially in the aggregation process. Thus, existing Arithmetic FHE accelerators offer limited improvements for GNN inference based on Arithmetic FHE schemes.

The Arithmetic-and-Logic FHE scheme, utilizing HLUT for non-linear computations, maintains modest FHE parameters like a ring dimension of 4096. While this minimizes overhead for aggregation, the intrinsic computational intensity of HLUT remains a significant bottleneck. To address this, we explore HLUT acceleration on Logic FHE accelerators. XHEC [23] and Strix [18] propose that parallel processing of multiple HLUT operations are particularly effective for high-parallelism scenarios. The feasibility of parallel HLUT execution primarily stems from the data demands of HLUT computation, involving the bootstrapping key, a test vector, and input ciphertexts (Logic Ciphertext). In our parameter, while Logic Ciphertext and the test vector are under 64KB, bootstrapping key's large size (256MB) requires off-chip memory access, making HLUT memory-intensive. However, the reuse of bootstrapping key and test vectors across parallel HLUT processes allows for linear performance enhancements without exacerbating memory bandwidth constraints. The strategy of parallel HLUT computations is highly effective for GNN non-linear operations, due to their inherent parallelism. With GNNs often involving thousands to millions of nodes, high parallelism is maintained even as feature length diminishes across layers. Thus, our aim with the Arithmetic-and-Logic FHE scheme is to centralize the computational overhead in HLUT, while designing an accelerator architecture specifically tailored for this scheme.

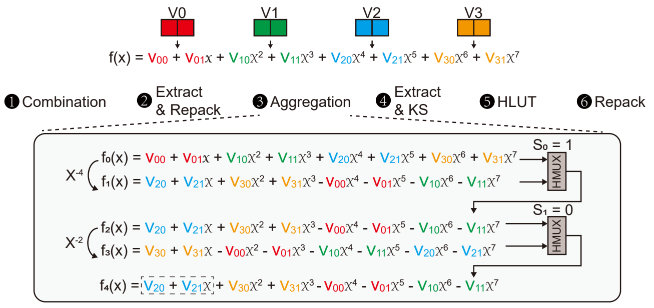
4 PRIVACY-PRESERVING GRAPH NEURAL NETWORK INFERENCE ALGORITHM

4.1 Threat model

In line with previous privacy-preserving machine learning works [6–9], our threat model involves a client uploading confidential graph data to a cloud service to receive predictions from an online machine learning model. The cloud server is considered semi-honest, meaning it follows prescribed protocols but may attempt to learn information from the data. To protect privacy of client data, the client encrypts their data using FHE and only they can decrypt the results with their private key. Our work specifically concentrates on the encryption of both graph node features and the graph structure data to protect sensitive information effectively. The parameters of the GNN model are owned by the cloud service and are kept in plaintext throughout the computation.

4.2 Algorithm Overview

Our GNN algorithm is illustrated in Figure 2, we initiate the process by encoding both node features and the weight vector as coefficients of polynomials. We utilize the method in [9] to obtain the inner product results by multiplication of input polynomial and weight polynomial. This is achieved by arranging the coefficients of weight polynomial in a reversed sequence, enabling us to extract the desired results from particular coefficients of the resultant polynomial multiplication.



V0
V1
V2
V3
$f(x) = V_{00} + V_{01}x + V_{10}x^2 + V_{11}x^3 + V_{20}x^4 + V_{21}x^5 + V_{30}x^6 + V_{31}x^7$
❶ Combination
❷ Extract & Repack
❸ Aggregation
❹ Extract & KS
❺ HLUT
❻ Repack
X^{-4}
$f_0(x) = V_{00} + V_{01}x + V_{10}x^2 + V_{11}x^3 + V_{20}x^4 + V_{21}x^5 + V_{30}x^6 + V_{31}x^7$
$f_1(x) = V_{20} + V_{21}x + V_{30}x^2 + V_{31}x^3 - V_{00}x^4 - V_{01}x^5 - V_{10}x^6 - V_{11}x^7$
$S_0 = 1$
HMUX
X^{-2}
$f_2(x) = V_{20} + V_{21}x + V_{30}x^2 + V_{31}x^3 - V_{00}x^4 - V_{01}x^5 - V_{10}x^6 - V_{11}x^7$
$f_3(x) = V_{30} + V_{31}x - V_{00}x^2 - V_{01}x^3 - V_{10}x^4 - V_{11}x^5 - V_{20}x^6 - V_{21}x^7$
$S_1 = 0$
HMUX
$f_4(x) = V_{20} + V_{21}x + V_{30}x^2 + V_{31}x^3 - V_{00}x^4 - V_{01}x^5 - V_{10}x^6 - V_{11}x^7$

Figure 2: An illustrative example of PPGNN protocol.

The inner product results are extracted from the Arithmetic Ciphertext and repacked into the original encoding format for use as input in aggregation. To accomplish this, we employ the extraction and repacking method as proposed in [24]. Additionally, we introduce the Homomorphic Aggregation method based on HMUX, allowing computing on compressed edge ciphertext while maintaining small overall parameters. Further details on this are discussed in Section 4.3. The output of Homomorphic Aggregation is then converted to Logic Ciphertext, serving as the input for HLUT. This conversion process, primarily involving rearrangement of the Arithmetic Ciphertext coefficients, is computationally efficient. Following this, HLUT is performed on these Logic FHE ciphertexts to perform activation function such as ReLU homomorphically. Finally, the results of HLUT are converted into Arithmetic

Ciphertext format by repacking method, to serve as inputs for the next layer.

Since real numbers are represented as scaled fixed-point numbers in our system, truncation is necessary to remove extra scale factors resulting from the multiplications in combination. We address this by integrating truncation directly into the HLUT functions and combining it with the ReLU operation, thereby avoiding any extra computational burden.

4.3 Homomorphic Aggregation

Efficient and accurate conversion between Logic Ciphertext and Arithmetic Ciphertext, as detailed in [24], demands uniform parameters which is challenge for HLUT's 64-bit modulus. Addressing this, we proposed customized Homomorphic Aggregation, optimized for the parameter limits.

Typically, aggregation can be interpreted as matrix multiplication or vector addition [25]. The matrix multiplication approach requires a higher multiplication level, demanding larger parameters incompatible with the constraints of modulus. Conversely, vector addition—which refers to the addition of feature vectors between source node and its neighboring nodes—is challenging to implement in FHE. The difficulty arises from the necessity to keep the indices of the neighboring nodes secret.

We solve this difficulty by utilizing HMUX to achieve aggregation. The node feature is still represented by the coefficients of the polynomial $p(x)$. When node i aggregates its neighbor j, it essentially conducts a binary search within this polynomial. We find this operation can be implemented as a sequence of HMUX operations in FHE. As shown in Figure 2, the HMUX input data consists of $p(x)$ and its rotated version $p(x) \times x^{-step}$, guided by the control data S_i. The term $p(x) \times x^{-step}$ shifts $p(x)$ by a specified "step". An aggregation operation is represented to a sequence of multiplexing between the output of previous HMUX and its rotated version, with the step halving in each iteration. After the final step, the node feature of neighbor j is positioned at the lowest few coefficients of the polynomial $p(x)$. These coefficients are then extracted and added to node i. Thus, node i complete the aggregation of single neighbor. This process is iteration until all the source nodes have aggregate all of their neighbor. The sequence of S_i values effectively represents the binary encoding of the neighbor node's index, encrypted on the client's side. Since the server is unable to differentiate the outcome of each HMUX operation based on its two inputs, it remains oblivious to the index of the neighbor node.

5 HARDWARE ACCELERATION ARCHITECTURE

5.1 Architecture Overview

Our PPGNN architecture is shown in Figure 3. The Linear Engine is in charge of the execution of aggregation, combination, and other ciphertext operations such as repacking and key-switching. It utilizes a polynomial unit for performing polynomial multiplications and additions, and shift operations such as automorphisms.

Moreover, the public key (PK) used in key-switching and repacking operations is reused between multiple inputs, and our small FHE parameters leads to correspondingly smaller key sizes. Therefore, a scratchpad memory is integrated within the Linear Engine, dedicated to storing the key-switching key and the repacking key.

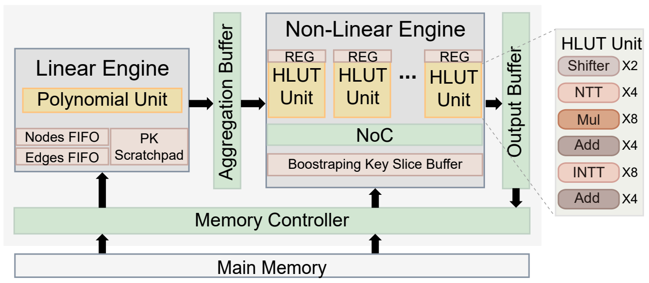


Figure 3: Architecture Overview.

The Non-linear Engine is mainly composed of HLUT units that are designed to execute the external product in a fully-pipelined manner. The unit includes several key components: the Number Theoretic Transform (NTT) and its inverse (INTT), a shifter for polynomial negacyclic rotation and ciphertext extraction, as well as modular multipliers and adders. The word length of these components is 64bit. The NTT, INTT, and shifter within our design function in a five-stage fully-pipelined manner, akin to the models presented in previous studies [15, 16]. This setup converts a 4096-point NTT/INTT into operations on a three-dimensional matrix, enabling the processing of 16 elements per cycle. Consequently, to complete a external product in a full-pipeline mode, the HLUT unit requires 256 cycles. The load and usage of bootstrapping key in HLUT is divided into many small slice and all HLUT units share the same bootstrapping key slice. The slice is buffered in bootstrapping key slice buffer and distributed to each HLUT unit via a one-to-all network on chip (NOC).

5.2 Data Flow Optimization for Pipelined Execution

The key insight of PPGNN is to create separate Linear and Non-linear Engines for handling respective operations, with a focus on ensuring their pipelined functionality. Since Logic Ciphertext is generated on-the-fly during the aggregation phase, balancing the throughput between the Linear Engine and Non-linear Engine is essential for their pipelined execution.

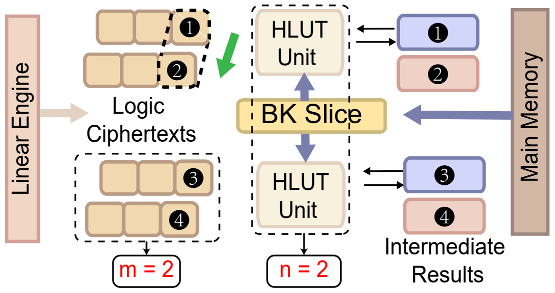


Figure 4: Data flow design of Non-linear Engine.

Recognizing that Homomorphic Aggregation and HLUT are the most demanding tasks in linear and non-linear operations respectively, we start by examining the characteristics of these two types of operations. In both HLUT and Homomorphic Aggregation, the computational cost is largely determined by HMUX operations. For Homomorphic Aggregation, each source node v requires $l \times \log_2 N$ HMUX operations, where N is the total number of nodes and l is the neighbor count of v. In HLUT, a node v undergoes $d \times n_{\text{lwe}}$ HMUX operations, with d representing the feature vector dimension and n_{lwe} being the FHE parameter, set to 1024 in our configuration. Given that l is typically small for most nodes due to graph sparsity, Homomorphic Aggregation incurs significantly less computational overhead compared to HLUT in practical graph scenarios. This aligns with our expectations, as a primary aim of the PPGNN software design is to shift the bulk of the computational load to HLUT.

On the other hand, Homomorphic Aggregation is significantly more memory-intensive compared to HLUT, mainly because the graph edge ciphertext data, its heaviest data component, cannot be reused during the aggregation of one GNN layer.

We address this challenge by optimizing for data flow within the HLUT. As illustrated in Figure 4, the HLUT engine processes multiple Logic Ciphertext in HLUT iteratively. As HLUT is a iteration processes, we ensure that the HLUT engine performs the same iteration stage for different Logic Ciphertext, prioritizing the completion of a single HLUT. Although this approach increases the demand for on-chip storage of intermediate results, the memory bandwidth needed decreases with the number of Logic Ciphertext. Importantly, the limited off-chip bandwidth is left more for Linear Engine by doing this. Specifically, with n HLUT units on-chip, each handling m Logic Ciphertext instances in alternation, the conditions for pipelining the Linear Engine and Non-Linear Engine are mathematically represented as follows:

$$\frac{Ct_{\text{Edge}}}{B_{\text{Total}} - \frac{B_{\text{BK}}}{m}} = \frac{1}{n \cdot TP} \tag{6}$$

where Ct_{Edge} is the size of edge ciphertext, B_{Total} is the total available memory bandwidth, B_{BK} is the bandwidth demand for bootstrapping key loading when HLUT unit works in full-pipeline, and TP is the throughput of each HLUT unit. These equations suggest that by appropriately setting the values of m and n, we can synchronize the Linear and Non-linear engines to work in a pipelined manner. Furthermore, simultaneously increasing m and n can also enhance the overall throughput of the accelerator.

5.3 Optimization Based on Graph Sparsity

The ability of the Homomorphic Aggregation method to operate on compressed graph formats such as COO and CSC, coupled with its utilization of graph sparsity, offers the potential to alleviate the memory intensity issue. A concern with this approach is the potential exposure of sensitive information, such as the degrees of nodes. To counteract this, we have decided to utilize the intrinsic power-law distribution characteristic of real-world graphs. This strategy allows us to balance computational efficiency with the need to maintain the confidentiality of specific structural details.

According to the power-law distribution, we categorize nodes into three distinct groups based on their degree: high, medium, and low. This classification is determined using two thresholds: 20% and 2.5% of the maximum degree. Nodes within each group are padded with dummy edges to match the degree corresponding to their respective threshold. Furthermore, the proportion of nodes in the three groups is fixed at a ratio of 10%:20%:70% by adding dummy nodes, thereby preventing the leakage of additional information through the ratio of node numbers in each group. Ultimately, the only information potentially disclosed to the server is the graph's maximum degree, a common trade-off in such scenarios [26].

Table 1: Area and Power of PPGNN.

Component	Area [mm^2]	TDP [W]
HLUT Unit	102.68	35.20
BK Slice Buffer (512KB)	3.30	3.52
Polynominal Unit	11.00	4.44
PK Scratchpad (1MB)	6.60	7.04
Linear Engine	**17.60**	**11.48**
Memory Interface (HBM2E) (14nm)	14.90	1.23
Total 1 (m=4,n=3, Interface*1) (14nm)	**56.38**	**16.30**
Total 2 (m=8,n=16, Interface*2) (14nm)	**349.94**	**109.94**

6 EVALUATIONS

6.1 Experimental Setup

Implementation. We developed and executed a cycle-accurate simulator to measure the execution time in cycle counts. Each component's area, power, and frequency measurements were obtained through RTL implementation and synthesis. We employed the Synopsys Design Compiler with the TSMC 40 nm standard library to carry out the synthesis. PPGNN system works in a 475 MHz clock frequency. The HBM bandwidth is 512GB/s per interface. To ensure a fair comparison with other works, we have scaled the area data to a 14nm process according to [27]. The area and power consumption details of PPGNN are presented in Table 1.

GNN Model and Datasets. As shown in Table 2, we selected widely-used datasets: Cora, Citeseer, and Pubmed. We employed a three-layer GNN architecture with hidden layer dimensions set to 32, 16, and 16.

Table 2: Dataset information.

Datasets	|Vertex|	|Edge|	#Feature
Cora (CR)	2,708	10,556	1,433
Citeseer (CS)	3,327	9,104	3,703
Pubmed (PB)	19,717	88,648	500

Concrete FHE parameters. Our system employs a security level of 119 bits. For the Logic Ciphertext and Arithmetic Ciphertext ciphertexts, we have set $n = 1024$ and $N = 4096$, respectively. The Arithmetic Ciphertext Q parameter is 105, composed of two moduli: $q_0 \approx 2^{45}$ and a special modulus $q' \approx 2^{60}$ [22], with σ set to 3.2.

Baseline. We employ a Arithmetic FHE scheme on the top of the SEAL library [28] and simulate its performance on the advanced Arithmetic FHE accelerator F1 [15], F1+ [16], CraterLake [16], and SHARP [21] as our Arithmetic FHE baseline. The packing format is in line with the one used in CryptoGCN [11], both the combination and aggregation are executed as homomorphic matrix-vector multiplications using the diagonal encoding method [29]. ReLU is approximated using the high-degree polynomial proposed in [14].

In the case of our Logic FHE baseline, we incorporate Logic FHE accelerators MATCHA [30], XHEC [23], and Strix [18] to execute our software scheme. Those operations that accelerators cannot process, such as automorphism, are computed on the CPU.

We have defined two resource configurations, labeled as Total 1 and Total 2 in Table 1. These correspond to the Arithmetic FHE and Logic FHE baselines, respectively and accommodate the considerable differences of area footprints in these two baselines.

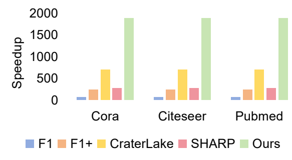
6.2 Performance and Energy

Speedup. As shown in Figure 5(a), when compared to the Arithmetic FHE baseline, PPGNN exhibits an average speedup of 11.3×.

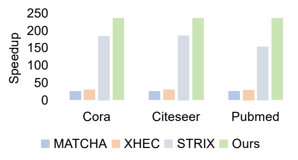
This acceleration is the cumulative result of an software-hardware co-design strategy that not only reduces the overall computational complexity but also incorporates dedicated hardware design for enhanced performance. The computation bottleneck of Arithmetic FHE baseline lies in aggregation phase, especially key-switching operation.

In Figure 5(b), PPGNN demonstrates a 6.2× speedup over the Logic FHE baseline. Given that most operations within our scheme are compatible with existing Logic FHE accelerators, our substantial performance gains are largely attributed to the deep hardware optimizations tailored specifically to our software scheme.

Specifically, MATCHA employs the bootstrapping key unrolling technique to lower the latency of bootstrapping operations at the cost of increased computation load, which is suboptimal for highly parallel ReLU computations. XHEC and Strix, on the other hand, capitalize on parallelism to enable the reuse of the BK across multiple operations. However, their performance is hampered during the aggregation phase, as this cannot be seamlessly pipelined with non-linear operations, creating a bottleneck that affects overall efficiency.



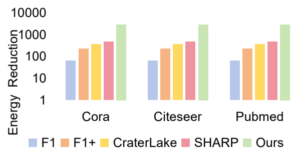
(a) Compared with Arithmetic FHE baseline.



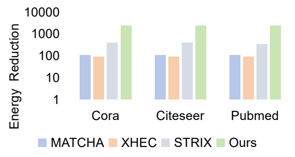
(b) Compared with Logic FHE baseline.

Figure 5: Speedup comparison normalized to CPU implementation.

Energy reduction. As depicted in Figure 6(a), PPGNN achieves a 17.7× reduction in energy consumption compared to the Arithmetic FHE baseline. In Figure 6(b), it is shown that PPGNN attains a 18.0× energy reduction relative to the Logic FHE baseline. The primary contributor to this energy efficiency is the reduction in total computation time.



(a) Compared with Arithmetic FHE baseline.



(b) Compared with Logic FHE baseline.

Figure 6: Energy comparison normalized to reduction over CPU implementation.

7 CONCLUSION

In this paper, we present PPGNN, a homomorphically encrypted GNN inference accelerator developed through a hardware and software co-design approach. PPGNN protects the privacy of both node attributes and graph structure data in GNNs, while ensuring high-precision computation. We first design a FHE-based GNN inference algorithm optimized for smaller parameters, simplifying computational complexity and concentrating processing efforts on non-linear operations conducive to parallel acceleration. This strategy enabled us to further develop a hardware architecture with parallel, fully pipelined non-linear units for high-throughput computation. The proposed design effectively lowers memory bandwidth requirements for these units, which allows to allocate more bandwidth to the memory-intensive linear units. Finally, the linear and non-linear units are executed in pipelined manner. Overall, PPGNN achieves 2.7× and 1.5× speedup over state-of-the-art Arithmetic FHE and Logic FHE accelerators while ensuring high accuracy, simultaneously with about 18× energy reduction on average.

REFERENCES

[1] X. Wei *et al.*, “Dual subgraph-based graph neural network for friendship prediction in location-based social networks,” *ACM Trans. Knowl. Discov. Data*, vol. 17, no. 3, pp. 1–28, 2023.
[2] H.-C. Yi *et al.*, “Graph representation learning in bioinformatics: trends, methods and applications,” *Brief. Bioinform.*, vol. 23, no. 1, p. bbab340, 2022.
[3] S. Wu *et al.*, “Graph neural networks in recommender systems: a survey,” *ACM Comput. Surv.*, vol. 55, no. 5, pp. 1–37, 2022.
[4] I. Baidu, “Paddlehelix,” 2023.
[5] J. Wang *et al.*, “A review on graph neural network methods in financial applications,” *arXiv*, 2021.
[6] R. Gilad-Bachrach *et al.*, “Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy,” in *ICML 2016*. PMLR, 2016, pp. 201–210.
[7] Q. Lou *et al.*, “Autoprivacy: Automated layer-wise parameter selection for secure neural network inference,” *NeurIPS*, vol. 33, pp. 8638–8647, 2020.
[8] S. Bian *et al.*, “Ensei: Efficient secure inference via frequency-domain homomorphic convolution for privacy-preserving visual recognition,” in *CVPR 2020*, 2020, pp. 9403–9412.
[9] Z. Huang *et al.*, “Cheetah: Lean and fast secure two-party deep neural network inference.” in *USENIX Security 2022*, 2022, pp. 2505–2522.
[10] Y. Wei *et al.*, “THE-V: Verifiable privacy-preserving neural network via trusted homomorphic execution,” in *ICCAD 2023*. IEEE, 2023, pp. 1–9.
[11] R. Ran *et al.*, “CryptoGCN: Fast and scalable homomorphically encrypted graph convolutional network inference,” *NeurIPS*, vol. 35, pp. 37 676–37 689, 2022.
[12] J. H. Cheon *et al.*, “Homomorphic encryption for arithmetic of approximate numbers,” in *ASIACRYPT 2017*. Springer, 2017, pp. 409–437.
[13] I. Chillotti *et al.*, “TFHE: fast fully homomorphic encryption over the torus,” *J. Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.
[14] S. Lee *et al.*, “HETAL: Efficient privacy-preserving transfer learning with homomorphic encryption,” in *ICML 2023*. PMLR, 2023, pp. 19 010–19 035.
[15] N. Samardzic *et al.*, “F1: A fast and programmable accelerator for fully homomorphic encryption,” in *MICRO 2021*, 2021, pp. 238–252.
[16] N. Samardzic *et al.*, “Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data,” in *ISCA 2022*, 2022, pp. 173–187.
[17] Zama, “TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data,” 2022, https://github.com/zama-ai/tfhe-rs.
[18] A. Putra *et al.*, “Strix: An end-to-end streaming architecture with two-level ciphertext batching for fully homomorphic encryption with programmable bootstrapping,” *MICRO 2023*, 2023.
[19] Y. Wei *et al.*, “IMGA: Efficient in-memory graph convolution network aggregation with data flow optimizations,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.
[20] I. Chillotti *et al.*, “Programmable bootstrapping enables efficient homomorphic inference of deep neural networks,” in *CSML 2021*. Springer, 2021, pp. 1–19.
[21] J. Kim *et al.*, “SHARP: A short-word hierarchical accelerator for robust and practical fully homomorphic encryption,” in *ISCA 2023*, 2023, pp. 1–15.
[22] C. Gentry *et al.*, “Homomorphic evaluation of the aes circuit,” in *CRYPTO 2012*. Springer, 2012, pp. 850–867.
[23] K. Nam *et al.*, “Accelerating n-bit operations over tfhe on commodity cpu-fpga,” in *ICCAD 2022*, 2022, pp. 1–9.
[24] H. Chen *et al.*, “Efficient homomorphic conversion between (ring) lwe ciphertexts,” in *ACNS 2021*. Springer, 2021, pp. 460–479.
[25] T. Yang *et al.*, “PIMGCN: a reram-based pim design for graph convolutional network acceleration,” in *DAC 2021*. IEEE, 2021, pp. 583–588.
[26] S. Wang *et al.*, “SecGNN: Privacy-preserving graph neural network training and inference as a cloud service,” *IEEE Trans. Serv. Comput.*, 2023.
[27] O. Villa *et al.*, “Scaling the power wall: a path to exascale,” in *SC 2014*. IEEE, 2014, pp. 830–841.
[28] “Microsoft SEAL (release 4.1),” https://github.com/Microsoft/SEAL, Jan. 2023, microsoft Research, Redmond, WA.
[29] C. Juvekar *et al.*, “GAZELLE: A low latency framework for secure neural network inference,” in *USENIX Security 2018)*, 2018, pp. 1651–1669.
[30] L. Jiang *et al.*, “MATCHA: A fast and energy-efficient accelerator for fully homomorphic encryption over the torus,” in *DAC 2022*, 2022, pp. 235–240.