FLASH: An Efficient Hardware Accelerator Leveraging Approximate and Sparse FFT for Homomorphic Encryption

Tengyu Zhang[1†], Yufei Xue[5†], Ling Liang[1], Zhen Gu[6], Yuan Wang[13], Runsheng Wang[134], Ru Huang[134], and Meng Li[213*]
[1]*School of Integrated Circuits &* [2]*Institute for Artificial Intelligence, Peking University, Beijing, China*
[3]*Beijing Advanced Innovation Center for Integrated Circuits, Beijing, China*
[4]*Institute of Electronic Design Automation, Peking University, Wuxi, China*
[5]*The Hong Kong University of Science and Technology, Hong Kong SAR*
[6]*DAMO Academy, Alibaba Group, China*
[†]Equal contribution [*]Corresponding author: meng.li@pku.edu.cn

***Abstract*—Private convolutional neural network (CNN) inference based on hybrid homomorphic encryption (HE) and two-party computation (2PC) emerges as a promising technique for sensitive user data protection. However, homomorphic convolutions (HConvs) suffer from high computation costs due to the extensive number theoretic transforms (NTTs). While customized accelerators have been proposed, they usually overlook the intrinsic error resilience and native sparsity of DNNs and hybrid HE/2PC protocols. In this paper, we propose FLASH, leveraging these key characteristics for highly efficient HConv. Specifically, we observe the private DNN inference is robust to computation errors and propose approximate fast Fourier transforms (FFTs) to replace NTTs and avoid the expensive modular reduction operations. We also design a flexible sparse FFT dataflow leveraging the high sparsity of weight plaintexts. With extensive experiments, we demonstrate FLASH improves the power efficiency by $90.7\times$ for weight transforms and by $9.7\times$ for all transforms in HConvs compared to existing works. As for the HConvs in ResNet-18 and ResNet-50, FLASH achieves about 87.3% energy consumption reduction.**

***Index Terms*—Homomorphic encryption, homomorphic convolution, accelerator, approximation fast Fourier transform, coefficient-encoding sparsity**

I. Introduction

Privacy has emerged as one of the major concerns when deploying deep neural networks (DNNs) on the cloud [1]–[3]. Homomorphic encryption (HE) has recently been proposed and attracted a lot of attention [4]–[6]. By encrypting the data into ciphertext polynomials, HE allows computations over the encrypted data without leaking any knowledge of the data itself [7], [8]. To apply HE for private DNN inference, there are two main approaches, the fully HE (FHE)-based schemes [4]–[6] and the hybrid HE/two-party computation (2PC)-based schemes [1], [2], [9]. The main difference lies in the implementation of non-linear activation functions. Hybrid schemes use 2PC protocols [2], [3], [10], which help to avoid the activation function approximations [11] and costly Bootstrapping operations [12], [13] in the FHE schemes. Hence, we focus on the hybrid schemes in our paper.

This work was supported in part by National Natural Science Foundation of China (62495102, 92464104 62125401), Beijing Municipal Science and Technology Program (Z241100004224015), Beijing Outstanding Young Scientist Program (JWZQ20240101004), and 111 Project (B18001).

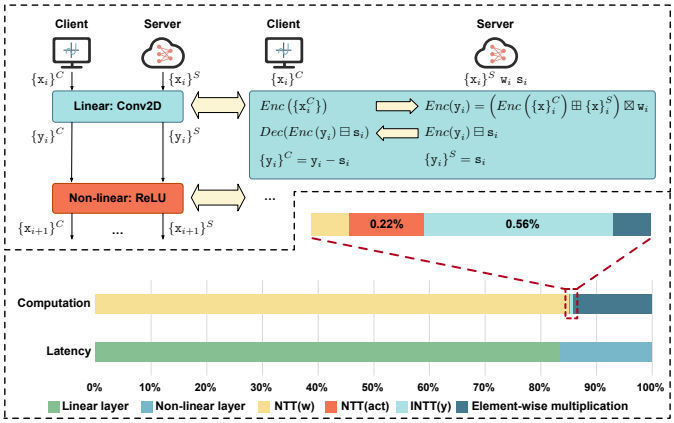

Fig. 1. Private CNN inference based on hybrid HE/2PC and its computation cost breakdown: computation latency accounts for the major latency bottleneck while NTTs of weight polynomials account for the major computation cost.

Private DNN inference remains significantly slower than regular inference in plaintext due to high computation overhead. We profile the latency of a residual block from ResNet-50 following a state-of-the-art (SOTA) hybrid protocol, i.e., Cheetah [2]. As shown in Figure 1, the homomorphic convolutions (HConvs) take more than 29.7 seconds on CPUs and account for the major bottleneck due to the repetitive computations of number theoretic transforms (NTTs) and its inverse (INTTs), especially for the weight polynomials. *While it is possible to pre-compute and store the weight polynomials in the NTT domain, it incurs significant memory overhead.* For example, it requires 23 GB to store the entire weights in the NTT domain for a 4-bit ResNet-50, causing more than $1000\times$ higher memory consumption [14].

Different HE accelerators have been proposed in recent years to speed up the costly NTTs [14]–[19]. [20]–[22] focus on optimizing dataflow and avoiding stalls between stages for high parallelism. [23], [24] implement large NTT by breaking into smaller ones. Though promising speedup has been achieved, these accelerators usually suffer from high area and power cost. For example, the area of the F1 accelerator is over 150 mm^2 and the power is around 100 W [23].

To enhance computation efficiency, we observe private DNN inference based on hybrid schemes exhibits good fault tolerance

978-3-9826741-0-0/DATE25/© 2025 EDAA

[25], [26], but cannot be effectively utilized in the modular arithmetic of NTTs. Furthermore, while the NTTs of weight polynomials incur the major computation bottleneck as shown in Figure 1, their inputs are highly sparse, e.g., more than 90%, providing an extra opportunity for efficiency improvements.

In this work, we propose FLASH, a highly efficient hardware accelerator for the hybrid HE/2PC scheme. Specifically, FLASH employs approximate fast Fourier transform (FFT) to replace NTT and avoid the expensive modular reduction, We use automatic design space exploration (DSE) to search for Pareto-optimal computation precision and to balance computation cost and accuracy. Note that FLASH introduces approximation during the computation process of BFV [7], which is different from other approximate homomorphic schemes [8]. A configurable sparse FFT dataflow is also proposed to skip unnecessary computations. Our contributions can be summarized as follows:

- We observe error resilience and sparsity of HConvs, offering the potential for improving efficiency.
- We prospose approximate FFT to eliminate the expensive modular reduction operations and develop a DSE framework for accuracy-efficiency trade-off.
- We develop a configurable sparsity-aware FFT dataflow to skip more than 86% unnecessary computations.
- FLASH improves power efficiency by $81.8\times \sim 90.7\times$ in weight transforms and reduces energy consumption by 87.3% for HConvs over SOTA works [14], [16], [23].

II. Background

A. Notations

This section introduces related notations and terminologies. Raw data (multi-dimension tensor in CNN) is called cleartext (**M**) and encoded to the polynomial denoted as plaintext ($\mathtt{m}$). $\mathtt{m}[k]$ indicates the k-th coefficient of $\mathtt{m}$. The plaintext can be encrypted into ciphertext ($\mathbf{m}$). We use the BFV HE scheme [7] and the main HE parameters comprise polynomial degree N, plaintext modulus t and ciphertext modulus q. t is determined by maximum sum-product (SP) bit-width, and q by the required noise budgets, security level. The symbols $\boxtimes$, $\boxplus$, and $\boxminus$ represent homomorphic multiplication, addition, and subtraction. The plaintext is defined over the polynomial rings $\mathbb{Z}_{N,t} = \mathbb{Z}_t[X]/(X^N+1)$ and ciphertext space over $\mathbb{Z}^2_{N,q} = \mathbb{Z}_{N,q} \times \mathbb{Z}_{N,q}$, where $\mathbb{Z}_{N,q} = \mathbb{Z}_q[X]/(X^N+1)$.

B. Hybrid HE-2PC Private CNN Inference

Flow of the Hybrid Scheme. The 2PC framework is mainly supported by arithmetic secret sharing (ArSS) and HE. For ArSS, an l-bit x is secretly shared by the client and server holding $\{x\}^C$ and $\{x\}^S$ respectively, with $x \equiv \{x\}^C + \{x\}^S \mod 2^l$. Figure 1 shows the basic flow of hybrid HE/2PC CNN inference [2], [27]. The client encrypts its share as $Enc(\{\mathtt{x}_i\}^C)$ and sends it to the server. The server performs homomorphic computation $(Enc(\{\mathtt{x}_i\}^C) \boxplus \{\mathtt{x}_i\}^S) \boxtimes w_i \boxminus s_i$ and send back to the client. After the one-round homomorphic evaluation, the server holds the share $\mathtt{s}_i$, and the client holds $\{\mathtt{y}_i\}^C = \mathtt{y}_i - \mathtt{s}_i$.

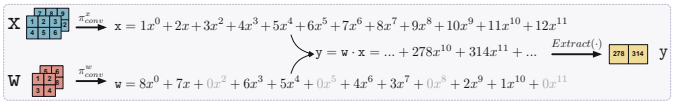

Fig. 2. Example of coefficient encoding.

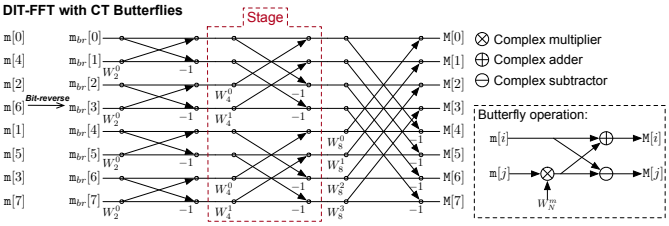

Fig. 3. Diagram of 8-point DIT FFT with CT butterflies.

Coefficient encoding. While CNNs compute over high-dimensional tensors, HE computes over 1-dimensional polynomials. Hence, encoding algorithms are needed to transform the tensors into polynomials while maintaining computation correctness and efficiency. Cheetah [2] recently presents a new coefficient encoding method for homomorphic linear evaluations to avoid expensive homomorphic rotations. It is achieved by directly placing elements in cleartext to the coefficients of plaintext according to certain encoding mappings. The two polynomials are then multiplied, followed by extracting specific coefficients from the product polynomial in Figure 2. When the tensor size is too large to be encoded into one plaintext, the tensors need to be tiled before the encoding.

FFT/NTT-based HConv. FFT and NTT are typically used to accelerate the polynomial multiplication (PolyMul). As shown in Figure 3, $\mathtt{m}$ is bit-reversed to $\mathtt{m}_{br}$ first. Take $\mathtt{m}[6]$ as an example. The binary format of its index is $(110)_2$, and changed to $(011)_2$ as $\mathtt{m}_{br}[3]$. Bit-reverse is followed by multi-stage butterfly operations. FFT and NTT share the same dataflow with Cooley-Turkey (CT) butterfly [28] and reduce the complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$. The only difference is the data type and the arithmetic unit used. FFT processes with complex floating-point (FP) coefficients, necessitating complex adders and multipliers, while NTT operates on polynomials over rings, requiring modular adders and multipliers. After encoding and encryption, computations are performed over polynomial rings $\mathbb{Z}_{N,t}$ and $\mathbb{Z}_{N,q}$. Therefore, NTT is used to calculate the exact multiplication results of weight and activation polynomials. In contrast, errors propagate in the FFT forward process because of the data type, unless the data width is large enough. Mainstream HConv accelerators are based on NTT and negacyclic convolution theorem [29], as shown in Figure 4(a). However, NTT is sensitive to errors, and the twiddle factors (W_N^{nk} in Figure 3) vary with different moduli. FFT does not have these issues. We present the efficient HConv algorithm via FFT proposed in [29] in Figure 4(b).

C. Related Work

There have been many prior efforts focusing on accelerating NTT using FPGAs [18], [20], [21], [30] or ASIC architectures [14], [16], [23], [24]. The inefficiency of NTT computation arises from data dependencies between stages, numerous computations, and costly modular operations. Related research has explored various methods to address these issues summarized in

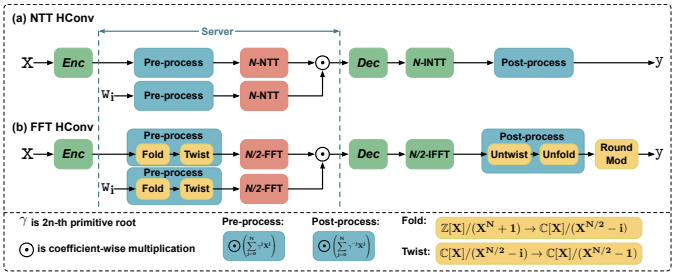

Fig. 4. HConv based on (a) NTT and (b) FFT.

TABLE I
COMPARISON WITH RELATED WORKS.

	PE	Application	Sparsity Support	NN Co-Opt	Optimization focus
[20]–[22]	NTT	FHE	No	No	Dataflow
[16], [23], [24], [30]	NTT	FHE	No	No	Decompsition
[18]	NTT	FHE	No	No	NTT-fusion
[31], [32]	FFT	Not for Private Inference	No	Yes	FP Multiplier
This work	FFT	Hybrid HE/2PC	Yes	Yes	Approximation and sparsity aware optimization

Table I. However, they fail to adequately consider the characteristics of encoded polynomials and fault tolerance during DNN inference. FLASH focuses on different optimization dimensions and proposes a DNN-aware HConv accelerator.

III. MOTIVATION

In this section, we discuss our key observations on the characteristics of hybrid HE/2PC protocol and DNN inference, which motivate our accelerator FLASH. The hybrid scheme can reduce noise levels [1], resulting in smaller HE parameters, such as ciphertext and plaintext modulus, when combined with widely used quantized DNNs [33]. In this paper, we target the low bit-width quantized CNN inference shown in Figure 5(a) and focus on hybrid HE/2PC protocol.

A. Error Resilience of Private DNN Inference

Private DNN inference is robust against random computation errors, which offers opportunities for approximate computation to improve efficiency. We observe the error robustness comes from kernel, layer, and network levels.

At the kernel level, the correct results can be acquired during the HE decryption as long as the total noise (including the original noise in encryption and the computation noise) does not exceed $\frac{q}{2t}$. At the layer level, the robustness comes from the re-quantization process. It converts the high-bit-width SP back to the low bit-width in Figure 5(a), discarding the errors in the original least significant bits (LSBs). At the network level, the final classification results may remain the same even when small errors exist at the layer output [34]. The robustness may come from the network architecture, e.g., pooling operations [26], or training methods [35], [36].

It should be noted that error robustness is not compatible with NTT entirely. In NTT, operands are always large integers, which means even tiny errors can be amplified while the magnitude of correct values are wrapped, reducing their ratio to an intolerable level. It is a potential way to replace NTT with FFT and leverage the error tolerance to reduce the bit-width in FFT. To achieve full equivalence with the 39-bit NTT, FP FFT with 8-bit exponent, 1-bit sign, and 39-bit mantissa is required. The hardware cost comparison is shown in Table II. The power

TABLE II
HARDWARE COST COMPARISON OF MODULAR MULTIPLIER AND COMPLEX FLOATING-POINT ONE.

Multiplier		Bit-width (bit)	Technology	Area (μm^2)	Power (mW)
Modular Mul	F1[1] [23]	32	14nm/12nm	1817	4.10
	CHAM[2] [21]	35, 39	28nm	3517	3.79
Complex FP Mul	FLASH	8+1+39	28nm	11744	8.26
Approx. FXP Mul		$39\times(k=5)$[3]		3211	1.11

[1] F1 selects modulus $q=-1 \mod N$ to remove a multiplier stage from traditional Barrett [37] and Montgomery [38] method.
[2] CHAM supports 3 moduli, each with only three non-zero bits to simplify multiplication as shifts and additions. We synthesize CHAM's RTL with the same technology as FLASH.
[3] $k=5$ means there are 5 bits "1" in the binary format of one multiplier, which will be elaborated in Section IV.

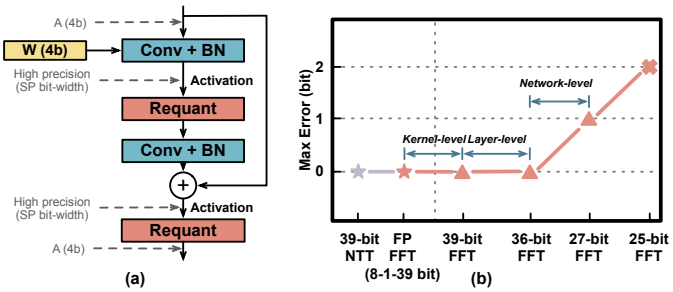

Fig. 5. (a) Low bit-width quantized ResNet block; (b) Computation bit-width reduction based on the kernel, layer, and network-level robustness.

of complex FP multiplications is approximately twice that of modular multiplication. As the number of multiplications in an N/2-point FFT is less than half of that in an N-point NTT in Figure 4, the overall cost of both operations is comparable. Considering robustness at three levels, we can replace it with fixed-point (FXP) data and reduce the bit-width to 27 without changing the final classification results as in Figure 5(b). The hardware cost can be further reduced with the optimization method proposed in Section IV. Table II illustrates that the approximate FXP multiplication performs more efficiently than optimized modular one used in [21]. Moreover, twiddle factors of NTT vary with different moduli, leading to storage or on-the-fly generation overhead. Above all, it is more effective to replace NTT with approximate FFT to accelerate PolyMul.

B. Weight Polynomials with Sparse Coefficients

Valid data in weight polynomials is usually sparse with the coefficient encoding, which can be leveraged to further reduce the NTT/FFT computation. Take the activation $\mathbf{X}$ of size $C \times H \times W$ as an example with $M \times C \times k \times k$ weight $\mathbf{W}$. Following the coefficient encoding algorithm proposed in Cheetah [2], for every $H \times W$ weight coefficients, at most $k \times k$ coefficients have valid values. Considering $H \times W$ is usually much larger than $k \times k$, e.g., $H = W = 58, k = 3$ for ResNet-50, the weight polynomials demonstrate high coefficient sparsity, as shown in Figure 7. To leverage it, there are two possible approaches:

- The first approach is to directly compute the polynomial multiplications in the coefficient domain without FFTs/NTTs, which makes it easy to skip the multiplications with zero coefficients.
- The second approach is to continue to use the FFT-based polynomial multiplications and leverage the sparsity to reduce the cost in the FFT of weight polynomials.

The second approach always requires much fewer multiplications across a wide range of sparsity, as the transforms of

activation polynomials are shared along output channels.

Based on the observations, our accelerator FLASH proposes an approximation method and customized dataflow to leverage these characteristics and improve hardware efficiency.

IV. FLASH Accelerator Design

A. Design Overview

FLASH leverages the sparsity of weight polynomials and the error resilience to accelerate coefficient-encoded HConvs. The hardware architecture of FLASH is shown in Figure 6. There are 60 approximate FFT processing elements (PEs), each including 4 butterfly units (BUs) and implementing FFT for one polynomial, which is the same scale as our baseline [21]. Other operations, e.g., FFT of activation and point-wise multiplications, are accomplished in FP field. Similarly, 4 BUs correspond to one input polynomial. We set the number of FP PEs to 4 as the transforms of activation account for only a small portion of the overall workload.

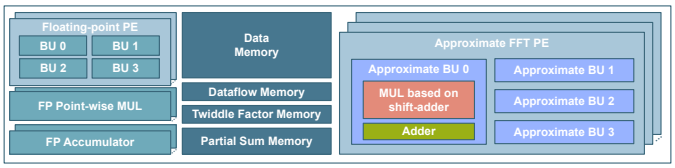

Fig. 6. Overview of FLASH's hardware architecture.

B. Sparse Butterfly Acceleration

In this section, we elaborate on the sparsity patterns in weight polynomials and show the customized dataflow to reduce the number of calculations. After being encoded by Cheetah, there are k contiguous valid values within intervals of H (or W) for the same channel, while the spacing between different channels spans $H \times W$ as shown in Figure 7. Therefore, there are two possible distributions of valid numbers after bit-reverse, contiguous and scattered, corresponding to two optimization methods, "skipping" and "merging" respectively.

Skipping Method. In the first case, when H and W are powers of two, such as 16, data originally located at multiples of $H \times W$ become contiguous after bit-reverse. We use "skipping" to drop calculation steps. As shown in Figure 8(a), when the inputs to an N-point butterfly network contain valid values only from 0 to $\frac{N}{2^x}$, the computation can be simplified by performing an $\frac{N}{2^x}$-point butterfly network and then duplicating the results. As a result, the other calculation steps are skipped and the total number of operations is significantly reduced.

Example 4.1: Consider that 4 valid data is contiguous from $\mathtt{m}_{br}[0]$ to $\mathtt{m}_{br}[3]$ in Figure 8(a) with $N = 16$. It takes $1/2N \log_2 N = 32$ multiplications in classical dataflow. With "skipping", only the first 4-point butterfly network (solid arrow) is required to get $\mathtt{m}'[0]$ to $\mathtt{m}'[3]$. And then duplicate four times. 87.5% operations of the original (dashed arrow) are reduced.

Merging Method. In the alternate scenario, valid coefficients become highly scattered after bit-reverse. We use "merging" to combine calculation steps across multiple stages. For example, only one valid coefficient exists in every 32 positions for the layer 28 in ResNet-50. The intervals between valid data points are usually uniform. In Figure 8(b), with only one valid element

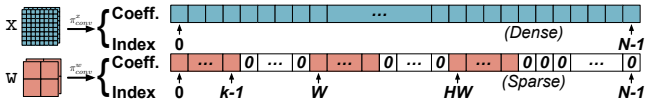

Fig. 7. Visualization of the coefficient-sparse polynomial.

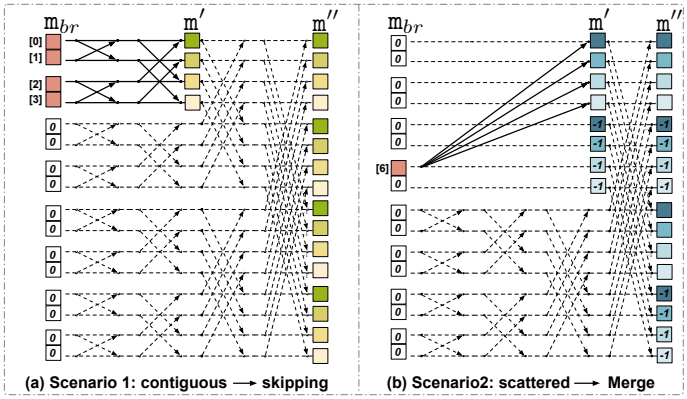

Fig. 8. Optimization of contiguous and scattered distribution. Solid arrows are required calculations and dashed arrows are reduced ones.

in an N-point butterfly network, computations across stages can be consolidated into a single multiplication by summing twiddle factor exponents. Each valid element connects directly to N output nodes via independent paths, alternating multiplications with twiddle factors and additions/subtractions with 0. This reduction allows $1/2N \times \log_2 N$ butterfly operations to be streamlined to just N multiplications.

Example 4.2: Consider that only 1 valid data at $\mathtt{m}_{br}[6]$ in Figure 8(b) with $N = 16$. With "merging", the computations within the first 8-point butterfly network can be simplified to $\mathtt{m}'[i]_{i=0}^{3} = \mathtt{m}_{br}[6] \otimes W_N^j$. W_N^j is the cumulative product of twiddle factors from the first three levels related to $\mathtt{m}_{br}[6]$. Besides, $\mathtt{m}'[4 \sim 7]$ are additive inverses of $\mathtt{m}'[0 \sim 3]$. As a result, calculations in the first three stages are merged together. There are only 4 times multiplication required to get $\mathtt{m}'[0 \sim 3]$, and then calculate additive inverses for $\mathtt{m}'[4 \sim 7]$, followed by duplicating for $\mathtt{m}''[8 \sim 15]$.

Furthermore, "merging" can be effectively combined with "skipping". As demonstrated in Figure 8(b), upon completion of $\mathtt{m}'[0 \sim 7]$ as the first 8-point butterfly network by merging three stages, the remaining operations can be skipped. With "skipping" and "merging", the amount of calculations can be greatly reduced in different layers of CNN inference. Twiddle factor exponents serve as addresses to fetch values from the ROM. The overhead for pre-computation and storage is minimal as twiddle factors remain the same set in FFT. Moreover, a single dataflow can be utilized across transforms in the same convolutional layer, typically involving thousands to tens of thousands of operations within butterfly networks. It makes the overhead of control logic trivial.

C. Approximate FFT Acceleration

As explained in Section III-A, the bit-width of fixed-point FFT could be reduced with error resilience from three levels. In this section, we propose to decrease the bit-width further so that hardware cost for one operation can be reduced. Besides, we construct a parameterized space for different bit-width in

multiple stages of FFT and explore it to find the optimal results in power-accuracy balance.

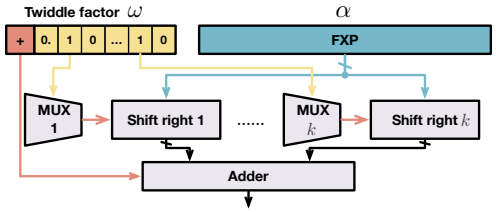

Fig. 9. Multiplier with quantized twiddle factor implemented by shift-add.

1) Quantized Twiddle Factors: Since one of the multipliers in FFT multiplication is the pre-known and fixed twiddle factor, we propose to quantize it further. The real and imaginary parts of the twiddle factors are fractions between -1 and 1, and can be represented by a 1-bit sign, a 1-bit integer, and a K-bit fraction. Multiplication between a fixed-point number α and a pre-known twiddle factor ω can be implemented using the right shifts and additions. For example, $\omega = \frac{21}{32}$, $\alpha \times \omega = \alpha >> 1 + \alpha >> 3 + \alpha >> 5$. Therefore, two factors impact the cost of shift-add multipliers: the number of "1" in the binary representation of the twiddle factor, denoted as k, and the positional distribution of the i-th "1" across all twiddle factors. The former determines the number of multiplexers (MUX), while the latter affects the size of each MUX, as shown in Figure 9. We use k to represent the quantization level of the twiddle factors and empirically reduce the MUX size to 8-to-1. Experimental results show that k is around 18 while ensuring that the classification accuracy degradation remains within 1%. With further approximation-aware training [25], [26], [35], k can be reduced to around 5 with 39-bit α, while the inference accuracy of W4A4 ResNet-50 [39] remains nearly unchanged. The power is comparable to 11-bit multiplier.

2) Approximate FFT Design Space Exploration: Based on the optimization opportunities from bit-width reduction discussed above, we propose a parameterized design space since the fault tolerance ability varies from different stages in FFT. Furthermore, we explore the space to find the accuracy and hardware cost balance.

We first formulate this problem as follows:

$$\begin{aligned} \min \quad & \mathrm{Cost}(N, \{dw_i\}_{i=2}^{\log_2(\frac{N}{2})-1}, \{k_i\}_{i=2}^{\log_2(\frac{N}{2})-1}) \\ \text{s.t.} \quad & \mathrm{Err}(N, \{dw_i\}_{i=2}^{\log_2(\frac{N}{2})-1}, \{k_i\}_{i=2}^{\log_2(\frac{N}{2})-1}) \leq T_{err} \end{aligned}$$

where dw refers to the bit-width of data flowing in FFT , T_{err} is the threshold of the computation error that can be tolerated, and i means the i-th stage in FFT.

The exploration workflow is illustrated in Figure 10. For error estimation, we utilize analytical simulations. Power consumption evaluations via register transfer level (RTL) simulations for each configuration are time-consuming. Given the FFT is constructed with butterfly operations, we employ the lookup table (LUT) based method for fast hardware cost estimation. We propose to pre-synthesize a set of butterfly units with different bit-width and store the hardware cost in the LUT. Then the cost of a given configuration can be estimated by summing small units together. Given the fast cost and error estimation, we leverage Bayesian optimization algorithms to solve the optimization problem iteratively and find the best configuration we need.

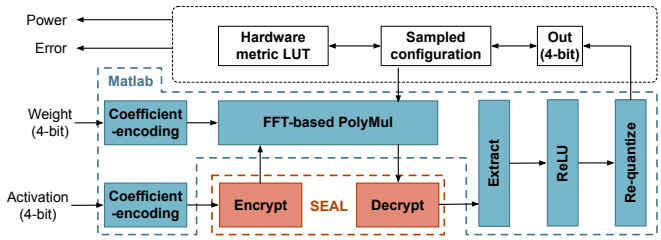

Fig. 10. Accuracy and hardware cost evaluation workflow for the multi-objective DSE.

V. Experimental Result

A. Experiment Setup

Accuracy Evaluation Setup. During the DSE, we sample the input features from convolution layers in pre-trained quantized ResNet-50 [39] on the ImageNet dataset and compute the error variance of convolution outputs for computation error evaluation. Approximate FFT and HE-related modules are implemented by MATLAB and SEAL library [40], respectively.
Hardware Evaluation Setup. We implement the RTL code of FLASH, including the approximate BUs with quantized twiddle factors, the floating-point BUs, and other logic units for HConv acceleration. All designs are synthesized with Synopsys Design Compiler (DC) using commercial 28nm PDK at 1GHz. We report the hardware metrics based on the synthesis results and perform power analysis using PrimeTime PX (PTPX).

B. Main Results

Hardware Efficiency and Error Comparison. FLASH implements the architecture shown in Figure 6, approximate BU, FP BU, FP MUL and FP accumulator for weight transforms, activation transforms, point-wise multiplications and accumulations, respectively. Unless otherwise specified, the average quantization level of the twiddle factors is set to $k = 5$.

Figure 12 shows the breakdown for each component. Considering the original computation bottleneck is weight transforms, FLASH reduces its hardware cost significantly. As a result, point-wise multiplication becomes a new bottleneck, which is the focus of our research in the future. We take the SOTA hardware accelerators on FPGA [20], [21] and ASIC designs [14], [16], [23] as our baseline. Table III demonstrates the area and power efficiency improvement of FLASH at different levels, ranging from transforms of weight polynomials to all transforms in convolutional inference of ResNet-50. *For transforms of weight, FLASH achieves* $81.8\times$ *to* $90.7\times$ *improvement.* It is attributed to the reduction in the cost per computation and the overall decrease in the number of computations. *As for all transforms in the inference of ResNet-50 on ImageNet, FLASH produces* $8.7\times$ *and* $9.7\times$ *improvement in power efficiency.* Taking the different technology nodes into consideration, *FLASH achieves about* $11.2\times$ *to* $18.8\times$ *improvement in area efficiency.* CHAM [21] instantiates the same number of BUs with FLASH. FLASH shows $21.84\times$ and $64.02\times$ speedup with negligible accuracy loss in the linear layers of different NN inference compared to CHAM in Table IV.

TABLE III
HARDWARE COST CONSUMPTION COMPARISON OF HCONV IN RESNET-50 ON IMAGENET.

Accelerator		N	Technology	Frequency (Hz)	Norm. Throughput[1] (MOPS)	Area (mm^2)	Power (W)	Area efficiency (MOPS/mm^2)	Power efficiency (MOPS/W)
HEAX [20]		2^{12}	FPGA	300M	1.95	-	-	-	-
CHAM [21]		2^{12}		300M	2.93	-	-	-	-
F1 [23]		2^{14}	14nm/12nm	1G	583.33	36.32	76.80	16.06	7.60
BTS [16]		2^{17}	7nm	1.2G	200.00	19.45	24.92	10.28	8.03
ARK [14]		2^{16}	7nm	1G	333.33	34.90	39.60	9.55	8.42
FLASH	Weight transforms	2^{12}	28nm	1G	186.34	0.74	0.27	250.23 (**15.6∼26.2**×)	688.82 (**81.8∼90.7**×)
	All transforms in HConv				187.90	4.22	2.56	44.54 (**2.8∼4.7**×)	73.48 (**8.7∼9.7**×)

[1] "Norm. Throughput" indicates the count of transforms (NTT or FFT) performed per second that is normalized to N=4096 for NTT or N=2048 for FFT.

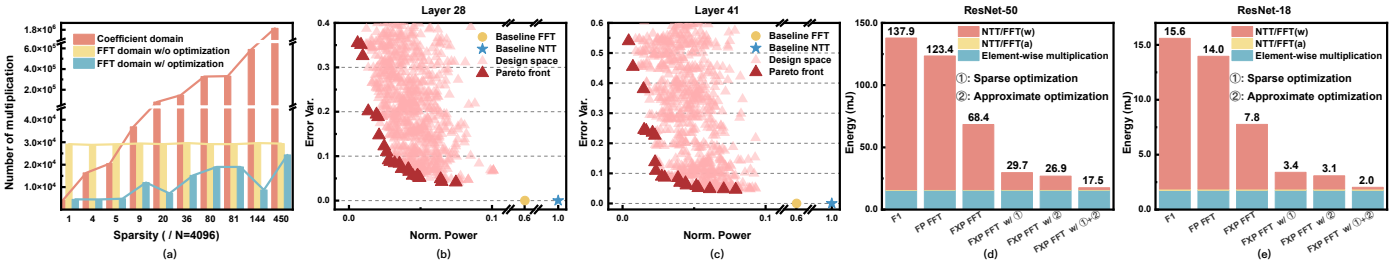

Fig. 11. (a) Reduction in number of multiplication. (b) DSE for layer 28 in ResNet-50. The x-axis is the normalized power estimation of weight FFT and the y-axis is the error variance of the HConv outputs. (c) DSE for layer 41. (d) Ablation study for contributions of sparse and approximate optimization to energy consumption during ResNet-50 inference. (e) Ablation study during ResNet-18.

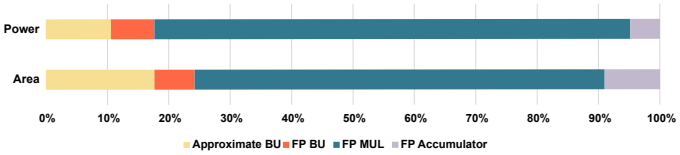

Fig. 12. Area and power consumption breakdown of FLASH.

TABLE IV
PERFORMANCE OF FLASH FOR LINEAR LAYERS OF NN INFERENCE.

	ResNet-18		ResNet-50	
	Latency (ms)	Accuracy (%)	Latency (ms)	Accuracy (%)
CHAM [21]	35.9	68.45	317.26	74.24
FLASH	1.64 (**21.84**×)	68.15	4.96 (**64.02**×)	74.19

We will thoroughly analyze the efficiency improvements brought by each optimization method.

Multiplication Count Reduction. In polynomial multiplication, the number of multiplications is the primary factor affecting energy consumption. Taking ResNet-50 as an example, Figure 11(a) illustrates how the butterfly network with optimized dataflow significantly reduces the multiplication count at various levels of sparsity, compared to the traditional dataflow of the butterfly network and direct computation in the coefficient domain. Note that the data is normalized to represent a single PolyMul per layer. Even with high sparsity, our method outperforms direct coefficient domain computation because the transforms of activation values are shared along output channels, rendering their cost per polynomial multiplication virtually negligible.

Multiplier Cost Reduction. With the DSE method proposed in Section IV, the design balance between computation accuracy and power consumption can be achieved for arbitrary convolution layers. We select two representative layers with different sizes and HE parameter settings from ResNet-50 for approximate FFT presentation. Figure 11(b) and (c) show 1000 solutions found for approximate FFT testing of each layer without approximate-aware training. The hardware cost will be reduced by 62.8% after training.

Ablation Study. We conduct an ablation study on the proposed practices, sparse and approximate optimizations, to verify their effectiveness respectively, as shown in Figure 11(d) and (e). "FFT(a)" is implemented on FP. "FXP FFT" sets the bit-width to 27 without accuracy loss in NN inference as mentioned in Figure 5(b). For the original energy consumption bottleneck in weight transforms, both optimization methods can reduce the cost to about 10% of the original. FLASH combines both sparse and approximate optimization techniques, reducing the power consumption of weight transforms to approximately 1%. The remaining activation transforms and point-wise multiplications in FLASH are performed using FP calculations. Since these operations account for a small portion of the overall workload, compared to F1 [23], the hardware cost of FLASH for performing complete privacy inference in linear layers of ResNet-18 and ResNet-50 is significantly reduced. *Overall, FLASH reduces about 87% energy consumption compared with F1.*

VI. CONCLUSION

In this paper, we propose an accelerator, FLASH, for HE to enable accurate, efficient, and configurable HConv based on approximate and sparse FFT. We observe the extreme sparsity of encoded weight polynomials and optimize the dataflow for the butterfly network to support it and reduce the number of multiplications. We analyze the fault tolerance in kernel, layer and network levels and replace NTT with approximate FFT to take advantage of it. Moreover, we explore the design space to determine the degree of approximation for each stage in FFT to ensure the computation accuracy as well as significantly improve the power efficiency. FLASH integrates these optimizations and implements the complete HConv. We demonstrate 90.7× power efficiency improvement for weight transforms and 87.3% energy consumption reduction in the private DNN inference.

REFERENCES

[1] 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)*, 2018, pp. 1651–1669.

[2] Z. Huang, W.-j. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure Two-Party deep neural network inference," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 809–826.

[3] Q. Pang, J. Zhu, H. Möllering, W. Zheng, and T. Schneider, "Bolt: Privacy-preserving, accurate and efficient inference for transformers," *Cryptology ePrint Archive*, 2023.

[4] J. W. Bos, K. Lauter, and M. Naehrig, "Private predictive analysis on encrypted medical data," *Journal of biomedical informatics*, vol. 50, pp. 234–243, 2014.

[5] 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 *International conference on machine learning*. PMLR, 2016, pp. 201–210.

[6] A. Sanyal, M. Kusner, A. Gascon, and V. Kanade, "Tapas: Tricks to accelerate (encrypted) prediction as a service," in *International conference on machine learning*. PMLR, 2018, pp. 4490–4499.

[7] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.

[8] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer, 2017, pp. 409–437.

[9] D. Kim, J. Park, J. Kim, S. Kim, and J. H. Ahn, "Hyphen: A hybrid packing method and its optimizations for homomorphic encryption-based neural networks," *IEEE Access*, 2023.

[10] W.-j. Lu, Z. Huang, Z. Gu, J. Li, J. Liu, K. Ren, C. Hong, T. Wei, and W. Chen, "Bumblebee: Secure two-party inference framework for large transformers," *Cryptology ePrint Archive*, 2023.

[11] E. Hesamifard, H. Takabi, and M. Ghasemi, "Cryptodl: Deep neural networks over encrypted data," *arXiv preprint arXiv:1711.05189*, 2017.

[12] C. Gentry, *A fully homomorphic encryption scheme*. Stanford university, 2009.

[13] N. Samardzic, A. Feldmann, A. Krastev, N. Manohar, N. Genise, S. Devadas, K. Eldefrawy, C. Peikert, and D. Sanchez, "Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 173–187.

[14] J. Kim, G. Lee, S. Kim, G. Sohn, M. Rhu, J. Kim, and J. H. Ahn, "Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1237–1254.

[15] S. Kim, K. Lee, W. Cho, Y. Nam, J. H. Cheon, and R. A. Rutenbar, "Hardware architecture of a number theoretic transform for a bootstrappable rns-based homomorphic encryption scheme," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2020, pp. 56–64.

[16] S. Kim, J. Kim, M. J. Kim, W. Jung, J. Kim, M. Rhu, and J. H. Ahn, "Bts: An accelerator for bootstrappable fully homomorphic encryption," in *Proceedings of the 49th annual international symposium on computer architecture*, 2022, pp. 711–725.

[17] R. Agrawal, L. de Castro, G. Yang, C. Juvekar, R. Yazicigil, A. Chandrakasan, V. Vaikuntanathan, and A. Joshi, "Fab: An fpga-based accelerator for bootstrappable fully homomorphic encryption," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 882–895.

[18] Y. Yang, H. Lu, and X. Li, "Poseidon-ndp: Practical fully homomorphic encryption accelerator based on near data processing architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[19] Y. Zhu, X. Wang, L. Ju, and S. Guo, "FxHENN: Fpga-based acceleration framework for homomorphic encrypted cnn inference," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 896–907.

[20] M. S. Riazi, K. Laine, B. Pelton, and W. Dai, "HEAX: An architecture for computing on encrypted data," in *Proceedings of the twenty-fifth international conference on architectural support for programming languages and operating systems*, 2020, pp. 1295–1309.

[21] X. Ren, Z. Chen, Z. Gu, Y. Lu, R. Zhong, W.-J. Lu, J. Zhang, Y. Zhang, H. Wu, X. Zheng *et al.*, "CHAM: A customized homomorphic encryption accelerator for fast matrix-vector product," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[22] J. Mu, Y. Ren, W. Wang, Y. Hu, S. Chen, C.-H. Chang, J. Fan, J. Ye, Y. Cao, H. Li *et al.*, "Scalable and conflict-free ntt hardware accelerator design: Methodology, proof and implementation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[23] N. Samardzic, A. Feldmann, A. Krastev, S. Devadas, R. Dreslinski, C. Peikert, and D. Sanchez, "F1: A fast and programmable accelerator for fully homomorphic encryption," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 238–252.

[24] J. Kim, S. Kim, J. Choi, J. Park, D. Kim, and J. H. 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*, 2023, pp. 1–15.

[25] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei, "Ares: A framework for quantifying the resilience of deep neural networks," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[26] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[27] W. Z. Srinivasan, P. Akshayaram, and P. R. Ada, "Delphi: A cryptographic inference service for neural networks," in *Proc. 29th USENIX Secur. Symp*, 2019, pp. 2505–2522.

[28] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[29] J. Klemsa, "Fast and error-free negacyclic integer convolution using extended fourier transform," in *International Symposium on Cyber Security Cryptography and Machine Learning*. Springer, 2021, pp. 282–300.

[30] C. Wang and M. Gao, "Sam: A scalable accelerator for number theoretic transform using multi-dimensional decomposition," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 1–9.

[31] C. Wen, Y. Wu, X. Yin, and C. Zhuo, "Approximate floating-point fft design with wide precision-range and high energy efficiency," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, 2023, pp. 134–139.

[32] X. Zhao, C. Yan, C. Wang, and W. Liu, "Design of approximate floating-point fft with mantissa bit-width adjustment algorithm," in *2022 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*. IEEE, 2022, pp. 265–269.

[33] T. Xu, M. Li, and R. Wang, "Hequant: Marrying homomorphic encryption and quantization for communication-efficient private inference," *arXiv preprint arXiv:2401.15970*, 2024.

[34] H. D. Dixit, S. Pendharkar, M. Beadon, C. Mason, T. Chakravarthy, B. Muthiah, and S. Sankar, "Silent data corruptions at scale," *arXiv preprint arXiv:2102.11245*, 2021.

[35] S. Kim, P. Howe, T. Moreau, A. Alaghi, L. Ceze, and V. Sathe, "Matic: Learning around errors for efficient low-voltage neural network accelerators," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1–6.

[36] Z. He, J. Lin, R. Ewetz, J.-S. Yuan, and D. Fan, "Noise injection adaption: End-to-end reram crossbar non-ideal effect adaption for neural network mapping," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[37] P. Barrett, "Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor," in *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1986, pp. 311–323.

[38] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

[39] Z. Yao, Z. Dong, Z. Zheng, A. Gholami, J. Yu, E. Tan, L. Wang, Q. Huang, Y. Wang, M. Mahoney *et al.*, "Hawq-v3: Dyadic neural network quantization," in *International Conference on Machine Learning*. PMLR, 2021, pp. 11 875–11 886.

[40] "Microsoft SEAL (release 3.2)," https://github.com/Microsoft/SEAL, Feb. 2019, microsoft Research, Redmond, WA.