EPIC: Efficient Packing for Inference using Cheetah

Sarabjeet Singh
sarab@cs.utah.edu

Shreyas Singh
shreyas.singh@utah.edu

Rajeev Balasubramonian
rajeev@cs.utah.edu

***Abstract*—Emerging cryptographic techniques, like Homomorphic Encryption, enable computation while preserving data privacy. However, their complexity restricts their adoption. In this work, we characterize a state-of-the-art work for private CNN inference, Cheetah [2], and identify its memory bottleneck. We then propose a packing technique to improve weight reuse, resulting in 10× energy saving while reducing latency by 54%.**

I. Introduction

Machine Learning's strength comes from working with large sets of data, which raises privacy concerns. Cryptographic techniques, like Homomorphic Encryption (HE), offer functionality of performing computations over encrypted data, but comes at a higher complexity. For instance, HE based CNN inference is 5 orders of magnitude slower than unencrypted inference on a CPU [1]. State-of-the-art work Cheetah [2] demonstrates techniques that improve upon the existing works. In this work, we first characterize Cheetah and identify its key bottlenecks - memory accesses and NTT operation. We propose a weight packing scheme to improve its reuse, lowering the total memory accesses, at the cost of increased NTT operations. We demonstrate that, for most layers in ResNet50, this trade-off is justified – we observe $10\times$ lower energy with almost half inference latency over Cheetah baseline.

II. Background

A. Homomorphic Encryption operations

Modern HE schemes, based on the hardness of Ring-LWE problem, enables addition (PolyAdd) and multiplication (PolyMult) over encrypted data. A single message (modulo-t) is encoded as a n-degree polynomial and then encrypted to a ciphertext - a pair of polynomials of same degree and modulo-Q coefficients. Operations are carried over ciphertext, and the result is obtained after its decryption. However, performing operations over ciphertext increase its intrinsic error/noise. To ensure correct decryption, ciphertexts have a certain noise budget, a depth factor which bounds the number of operations, typically multiplications. HE parameters (n, t, Q) determine the security strength and the noise budget of the scheme. All ciphertext computations in HE are performed modulo Q and PolyMult is modular multiplication.

B. Reducing Homomorphic Encryption's complexity

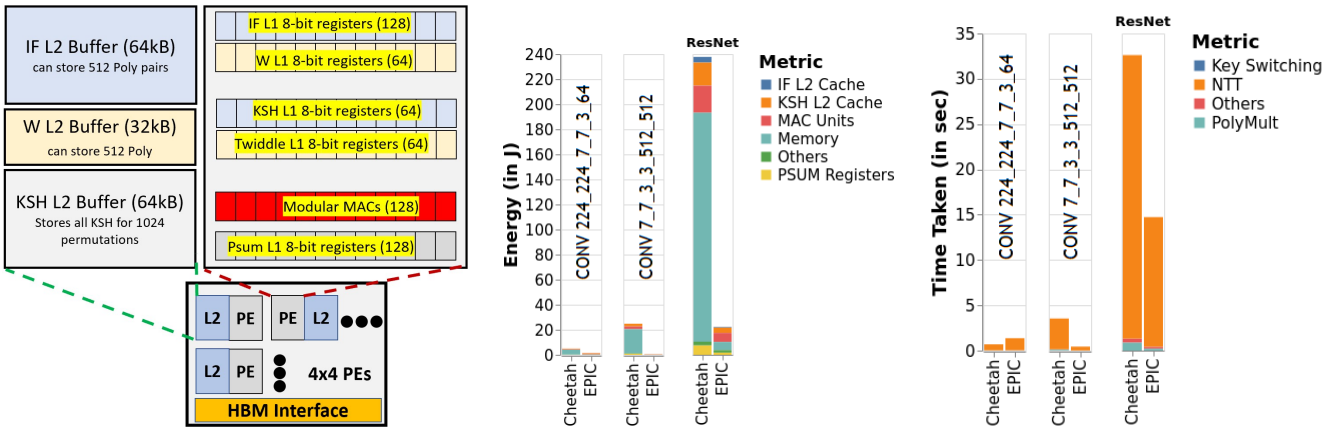
Existing works typically employ the following techniques to reduce HE's complexity: 1) Residue Number System (RNS) is used to decompose a single polynomial with wide coefficients into a set of polynomials with narrower coefficients, called residue polynomials, that can be evaluated in parallel. 2) Secondly, while PolyMult requires convolving their coefficients, an expensive $O(n^2)$ operation, Number Theoretic Transform (NTT) makes it faster. Degree-n polynomials are first transformed from coefficient space to evaluation space using the $O(n \log n)$ NTT algorithm, followed by an coefficient-wise multiplication ($O(n)$). Polynomials are stored in evaluation space and only revert back for decryption, avoiding domain conversion cost using expensive NTT. 3) Lastly, given the ciphertext complexity over plaintext, most schemes *pack multiple data within a single ciphertext*. Given n slots, n elements are encoded in a polynomial. However, to add two elements which reside at different slots, a *Rotation* operation is performed over the ciphertext to align the coefficients. This requires NTT to convert between the evaluation and coefficient formats, which is an **expensive operation**. Also, a Key Switching (KS) procedure is required upon ciphertext rotation that multiplies the cipher with a Key Switching Hint (KSH). KSHs are provided by the user for every possible permutation of ciphers, and cause large data movement.

III. Proposal

A. Performing CNN inference with Cheetah

Cheetah [2] is the state-of-the-art implementation of HE-based CNN inference. It assumes a threat model where the user and client are curious, but honest. Input features from user are encrypted and sent to the cloud, which performs convolution with its model weights (unencrypted). Since HE cannot perform non-linear layers, like ReLU, pooling, etc, trivially, the cloud sends the output neuron of each convolution layer to the user, who decrypts and performs the non-linear layers and re-encrypts it for the cloud. Input and weight polynomials are brought on-chip and multiplied to generate partial sum. With a new weight, a new partial sum is generated. In order to accumulate, the partial sums have to be aligned. Therefore, after every PolyMult, partial sums have to be rotated, an expensive operation consisting of NTTs and KS. We use a similar inference model as Cheetah.

Cheetah packs input feature map ($IF = (X, Y, C)$) values from a single channel in a single ciphertext polynomial. If more slots are available ($n > XY$), they go across channels as well (C_t). A **single** weight ($W = (R, S, C, K)$) value from a single kernel and channel is taken and encoded as a weight plaintext polynomial. To exploit parallelism, the weight value is replicated across all slots in that polynomial, as long as the channel matches the channel in corresponding IF slots; if not, pad with zeroes. Since a single value is packed in a polynomial, they need to fetch a polynomial from a larger



(a) Architecture (b) Energy (in J) (c) Performance (in s)

Fig. 1: **(a)** Architecture. **(b,c)** Energy and Performance comparison of Cheetah v/s EPIC, running ResNet50. First two bars are 2 example layers from ResNet50, while last bar illustrates results while running whole ResNet50 model.

memory for all $RSCK$ iterations over IFs. Usually, the reuse distance is too large to cache on-chip, and **all weight fetches become memory fetches**. While the latency of these fetches are hidden behind large computation cycles of NTT, these fetches consume quite a lot of energy. Additionally, if weight polynomial is padded with zeros, Cheetah's PolyMult results in many **ineffective computations**. In this work, we propose a technique to address both these issues.

B. EPIC: Packing weights efficiently

Cheetah's ineffective computations and memory footprint arise due to an inefficient weight encoding. A single weight value from $RSKC$ weights is packed in 1 polynomial. Instead, we propose EPIC - packing multiple values in the weight polynomial. Given C_t channels packed in the IF cipher, we first pack RS values from those C_t channels; then fill the remaining slots with $K_t < K$ weight values. In short, we pack RSC_tK_t weight values in the weight polynomial. This means that we can reuse the same polynomial, by rotating RSK_t times, reducing weight memory accesses by $RSK_t\times$ compared to Cheetah. However, weights must be rotated along with psum, which incurs more NTT operations. We demonstrate that *in most cases*, this is a worthwhile trade-off.

C. Methodology

We compare Cheetah and EPIC by running encrypted inference on ResNet50 model. Similar to Cheetah, we present results only for server-side convolution operations. Since most CNNs perform a non-linear layer after every convolution layer, the HE parameters can be reduced to serve a depth of one cipher-cipher multiplication. We use a popular scheme, BFV, to extract parameters - $(n = 1024, \log Q = 19)$. Since many models, and hence CNN accelerators, employ 8-bit INT precision activations, we encode 8-bit plaintext per slot. We decompose each ciphertext into 4 RNS polynomials with 8-bit coefficients. We model a simple architecture (Figure 1a) that consists of 16 PEs in a torus network, each with a IF, W, and KSH L2 buffer, and connected with HBM2 memory. Each PE is responsible for 64 coefficients - 64 W registers, 128 IF registers ($2\times$ since ciphertexts are a pair of polynomials), 128 Modular Multiply-Add units, and 64 registers to store KSH and Twiddle factors (used by NTT). A fast optimized modular multiplier is taken from [3].

D. Results

We first characterize the component-wise energy consumption of Cheetah in Figure 1b. We note that memory accesses contribute to the majority of energy. This is because Cheetah requires a new weight after performing a single PolyMult. Since Cheetah packs a single value in a polynomial, it is unable to capture the large reuse distance using a L2 cache. Our dataflow reuses IFs by bringing RS weight values. Next, a different channel is brought on the PEs to accumulate to the same psum. Once we exhaust input channels, we iterate over output channels. The reuse distance of IFs here is determined by C/C_t, which is usually <512 in our experiments. We capture this reuse with a L2 IF cache. EPIC reuses the weights on-chip, reducing the memory accesses. Due to this, EPIC consumes $\sim 10\times$ lower energy than Cheetah, as seen from the last bar (represents running whole ResNet50).

From Figure 1c, we note that, overall, EPIC observes a speedup of 2.2 over Cheetah. This is because Cheetah pads weight polynomials with zeros for $C_t > 1$, resulting in ineffective computations in parts of psum polynomials. This increases the number of NTT calls, increasing the inference latency. However, when $C_t = 1$, Cheetah outperforms EPIC, as seen by layer *conv224_224_7_7_3_64*. This is because EPIC's packing requires weights to be rotated with psums, hence doubling the NTT calls over Cheetah. However, this scenario rarely happens since in most layers $XY < n$, and despite the performance drop, EPIC still saves energy by reducing memory accesses, as seen in Figure 1b.

IV. Conclusion

In this work, we characterize Cheetah and identify memory as a major bottleneck. We then propose a weight packing technique that results in speedup of 2.2 while saving $10\times$ energy, over Cheetah running ResNet50.

REFERENCES

[1] 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*, 2016.

[2] B. Reagen, W.-S. Choi, Y. Ko, V. Lee, H.-H. Lee, G.-Y. Wei, and D. Brooks, "Cheetah: Optimizing and Accelerating Homomorphic Encryption for Private Inference," in *27th International Conference on High Performance Computer Architecture (HPCA)*, 2021.

[3] 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 *54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021.