2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 978-1-6654-2235-2/20/$31.00 ©2021 IEEE | DOI: 10.1109/HPCA51647.2021.00013

Cheetah: Optimizing and Accelerating Homomorphic Encryption for Private Inference

Brandon Reagen*[1,2], Woo-Seok Choi*[3], Yeongil Ko[4], Vincent T. Lee[5]
Hsien-Hsin S. Lee[2], Gu-Yeon Wei[4], David Brooks[4]

*Equal contribution
New York University[1], Facebook AI Research[2], Seoul National University[3]
Harvard University[4], Facebook Reality Labs Research[5]

***Abstract*— As the application of deep learning continues to grow, so does the amount of data used to make predictions. While traditionally big-data deep learning was constrained by computing performance and off-chip memory bandwidth, a new constraint has emerged: privacy. One solution is homomorphic encryption (HE). Applying HE to the client-cloud model allows cloud services to perform inferences directly on clients' encrypted data. While HE can meet privacy constraints it introduces enormous computational challenges and remains impractically slow on current systems.**

This paper introduces Cheetah, a set of algorithmic and hardware optimizations for server-side HE DNN inference. Cheetah proposes HE-parameter tuning and operator scheduling optimizations, which together deliver up to 79× speedup over the state-of-the-art. However, HE inference still falls short of real-time inference speeds by nearly four orders of magnitude. Cheetah further proposes an accelerator architecture to understand the degree of speedup hardware can provide and whether it can bridge HE's real-time performance gap. We evaluate several DNNs and find that privacy-preserving HE inference for ResNet50 can approach real-time speeds with a 587mm^2 accelerator dissipating 30W in 5nm.

I. Introduction

Deep learning lies at the heart of many modern services and applications, and is one of the most widely used methods to process personalized data. These models have become so successful and computationally efficient that deep learning is now integral to everyday life. However, as such services become ever-intricately woven into our lives, there is growing demand for privacy-preserving machine learning—a daunting task that this paper seeks to address.

Several techniques exist that offer privacy for deep learning inference that trade off the degree of security delivered versus computational efficiency. Generally, these techniques deliver security via system implementation or mathematical guarantees. Implementation-based methods include (i) moving computation to edge devices, i.e., *local computation* [56], [73], and (ii) *trusted execution environments* (TEEs), e.g., SGX [10], [15], [62]. Both methods achieve security by monitoring and restricting data usage via a combination of software and hardware implementations. In contrast, methods offering provable mathematical guarantees provide a theoretically-quantifiable level of privacy. Such solutions include (i) *differential privacy* (DP) [4], [13], [18], [21], (ii) *secure multi-party compute* (MPC) [33], [37], [50], [51], and (iii) *homomorphic encryption* (HE) [7], [29], [32], [55]. Table I summarizes the techniques and limitations of each with respect to wide-scale deployment.

TABLE I: Generalization of privacy-preserving techniques.

Solution	Security	Limitation
Local	System	Edge performance; leaks model
TEE	System	Performance; side-channels
DP	Statistical	Applications; utility-privacy tradeoff
MPC	Cryptographic	Communication bandwidth
HE	Cryptographic	Compute

Each of the above solutions have differing limitations. Local execution offers individual users improved security, but there is risk of sensitive information leaking or being stolen through the model, plus model-privacy concerns for service providers [63]. TEEs have been shown to be vulnerable to side-channel attacks, e.g., [15]. DP offers statistical privacy levels quantified via privacy loss ε but imposes an abstruse trade-off between ε and data utility [20]. Moreover, while DP has seen success in training [12], [47], its application to inference is an open question. MPC also delivers cryptographically-strong privacy guarantees. However, MPC performance is limited by communication bottlenecks [37], [42], [51], which require consideration of network-protocol and technology levels, or redesigning neural architectures [28], [41].

This paper focuses on homomorphic encryption (HE) to enable privacy-preserving deep learning inference, or HE inference. The key strength of HE is that it offers cryptographically-strong privacy guarantees, but these guarantees come at the cost of massive computational overheads. These overheads are so high that existing state-of-the-art implementations of HE inference [29], [32], [55] are still *five to six orders of magnitude slower than unencrypted, or plaintext, inference speed running on a CPU*. To put this in perspective, the current state-of-the-art HE inference solution (Gazelle [33]) takes 800ms for a single MNIST inference. These computational overheads are so extreme that prior research has yet to consider modern datasets and models, e.g., ImageNet and ResNet50, as even MNIST is currently beyond the realm of feasibility. In this paper, we propose a set of optimizations to substantially

978-1-6654-2235-2/21/$31.00 ©2021 IEEE
DOI 10.1109/HPCA51647.2021.00013

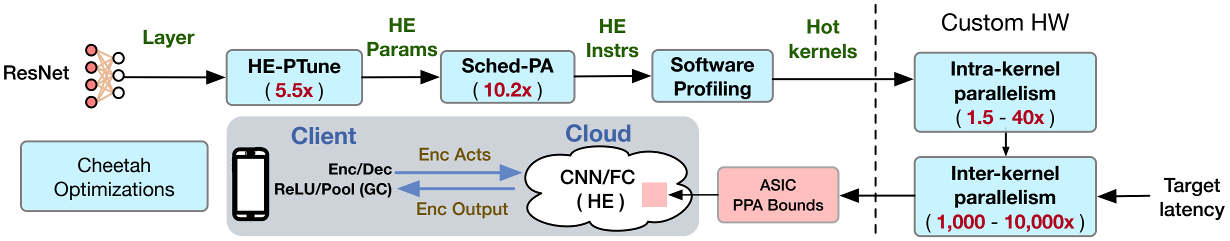


Fig. 1: The Cheetah optimizations and system design. Speedup achieved for ResNet50 is reported in red.

reduce this performance gap and show that HE can approach real-time performance levels when running on large custom hardware accelerators.

High-performance HE inference requires addressing three key challenges. First, at the algorithmic level, HE has configurable parameters that trade performance (i.e., HE operator latency) and "computational budget," canonically know as the *noise budget* in HE literature. This HE noise budget limits the amount of computation (i.e., number of HE operations) that can be applied to encrypted data while still allowing for correct decryption. When tuning HE parameters for performance, e.g., using smaller data types, the noise budget can be exceeded and cause the computation (i.e., decryption) to fail. The second challenge is how computations are *scheduled* and mapped to HE primitives. HE only supports a limited set of operators (e.g., add and multiply) that applications must be expressed as, and each operator increases noise differently. Therefore, noise-aware operator schedules can significantly improve performance by reducing accumulated noise, enabling higher-performing HE parameters to be used. The final challenge is the sheer number of computations HE inference entails. As we show, addressing this challenge requires hardware acceleration and leveraging the extreme degrees of parallelism in both DNNs and HE operators.

To address these challenges, this paper presents *Cheetah*: a set of optimizations (Figure 1) to speedup HE-based privacy-preserving machine learning inference by combining algorithmic optimizations and hardware acceleration. We assume Gazelle [33], the state-of-the-art, as our baseline. Our contributions are as follows:

First, we propose *HE-PTune* (Section IV), which is an analytical model that tunes HE parameters. HE-PTune automatically identifies the highest-performance HE parameter settings that satisfy noise budget constraints by tuning HE parameters based on the needs of each layer in a deep neural network model. HE-PTune's parameter tuning yields performance benefits of up to 11.7× for VGG16 and 5.5× for ResNet50 over the state-of-the-art.

Second, we propose a new schedule for dot product operations called *Sched-PA* to minimize the consumption of noise budget and improve performance. Sched-PA is a partial-aligned dot product schedule, which exploits the insight that the order of HE operations significantly impacts performance and noise budget. This allows Sched-PA to achieve a maximum additional speedup of 10.2× (5.2× harmonic mean) and, including HE-PTune, a combined speedup of up to 79.6× (13.5× harmonic mean) over the state-of-the-art.

Third, we propose a hardware accelerator architecture to accelerate HE inference leveraging the abundance of parallelism and opportunities for specialization. We begin by profiling HE kernels using a CPU software implementation [57]. The results indicate where cycles are spent and are used to perform an Amdahl's law-like study to provide insight into the orders of speedup each kernel requires to approach real-time latency. Next we implement each kernel in custom hardware and comprise them into a parameterized accelerator for processing HE inferences. Exploring the design space of possible design reveals a Pareto frontier, which we analyze to understand tradeoffs and conclude the feasibility of HE for private inference.

We find Cheetah can produce designs that approach real-time inference speeds through combining algorithmic optimizations and custom hardware. In the case of ResNet50, and adhering to practical area limitations [65], we estimate an accelerator of 587mm^2 and 30W in a 5nm technology node can process HE inference with a latency of 198ms (Section VII). Ignoring practical fabrication restrictions, a 1173mm^2 chip would process inferences in 101ms, matching CPU performance and meeting real-time requirements, which we assume to be 100ms as used in certain applications [66]. We conclude that while HE for neural inference is exceptionally computationally intensive the substantial parallelism and amenability of HE and DNNs to hardware acceleration puts real-time private inference within reach.

II. Overview and Assumptions

A. System Setup

A typical deep learning system setup is shown in the gray box of Figure 1. A client generates data and sends it to the cloud. The cloud performs inference and the result is returned to the client. The most direct way to apply HE is for the client to encrypt the data, the cloud processes the entire inference using HE, and the encrypted result is returned to the client. Unfortunately, this approach has two drawbacks: (1) HE cannot readily process nonlinear functions (without incurring prohibitively large penalties) and (2) many computations in DNNs requires a relatively large HE noise budget, which necessitates larger encryption parameters, resulting in poor performance. This effect is exacerbated by deeper networks.

A classic solution is to combine multiple cryptographic solutions, as done before in [37], [39], [42], [46], [54], and partition inferences across them. The typical approach is to

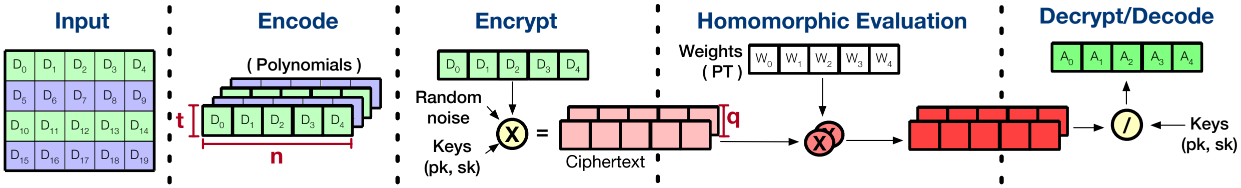

Input
Encode
Encrypt
Homomorphic Evaluation
Decrypt/Decode
D_0 D_1 D_2 D_3 D_4
D_5 D_6 D_7 D_8 D_9
D_{10} D_{11} D_{12} D_{13} D_{14}
D_{15} D_{16} D_{17} D_{18} D_{19}
(Polynomials)
t
n
Random noise
Keys (pk, sk)
Ciphertext
q
Weights (PT)
W_0 W_1 W_2 W_3 W_4
A_0 A_1 A_2 A_3 A_4
Keys (pk, sk)

Fig. 2: Overview of how data is processed using BFV homomorphic encryption.

execute linear operators (FC/CNNs) on the cloud using either homomorphic encryption (Gazelle [33]) or secret-sharing (MiniONN [37]), and non-linear functions on the client with Yao's garbled circuit (GC). This partition works quite well as GCs incur small computational overhead [5], [71] while the cloud can leverage powerful servers to handle the large processing load of HE/secret sharing. In this paper, we take Gazelle as our baseline as it is the fastest known implementation of private inference and uses HE, which is the focus of Cheetah.

In Gazelle, the client encrypts data to be processed and sends it to the cloud. The cloud applies a single linear layer (e.g., convolution) to the input using HE. ReLU and pooling functions are computed on the client using a GC. The GC is configured by the cloud and sent to the client along with the encrypted linear layer outputs. The client then decrypts the outputs and processes them using the GC. Note that allowing the client to observe the original outputs after decryption can leak the cloud's private model weights (knowing the inputs and outputs of a linear function would make it trivial to steal the cloud's model). To prevent this, the cloud obscures the actual activation values (both input and output) by adding random numbers to each, i.e., the client receives encrypted activation input also obfuscated with random numbers. After decryption, the client runs the GC, which includes a subtraction circuit to remove the added random numbers securely (recovering original values), an non-linear functions (ReLU or pooling), and finally an addition to obscure the plaintext output value and protect model weights. Once GC evaluation completes, the masked output is re-encrypted by the client and sent to the cloud. On the cloud, the random numbers added to the activation are removed via HE subtraction and the following linear layer is computed (using HE). The HE-MPC cycle repeats for each layer of the deep network.

Note that in homomorphic encryption, decryption resets the HE noise budget. Therefore, systems like Gazelle address both issues associated with nonlinear computation and limitations of HE noise budget. However, the computational overheads of HE—the focus of this paper—remain prohibitive. Cheetah addresses the HE compute bottleneck, which is an architecture/hardware problem, but the proposed optimizations for HE are more generally applicable to other solutions beyond Gazelle. Solving the communication/network bottleneck is beyond the scope of this paper. We expect contributions on the algorithmic (e.g., different MPC-based solutions [23], [43], [46]) and technology (e.g., 5/6G) front to help. Therefore, Cheetah assumes the same communication overheads as Gazelle. Whenever discussing HE performance results, it is always with respect to the server-side HE inference computation.

B. Threat Model

The threat model assumed by Cheetah is the same as in Gazelle [33], and similar to other two-party compute (2PC) solutions including DeepSecure [51], MiniONN [37], and SecureML [42]. The model assumes the client/user and cloud are honest but curious, i.e., each agent follows the protocol precisely but may try to infer information. Under this assumption, Cheetah preserves the privacy of both the clients' data and cloud's model weights. For more details, see [33].

Note that the protocol *does* leak some information about the model. Because ReLU and pooling layers are performed by the client, the client can learn the number and shape of each layer. The model weights values, however, are not leaked. It is possible to obscure this information (e.g., pad tensor dimensions and add null layers), but they are not considered here and left as future work. Cheetah focuses on improving users privacy while protecting the cloud's models (considered IP today [72]) from model-stealing attacks [63].

III. Background

This section provides a brief introduction to HE and the BFV scheme [22]. For a complete description see [8], [22].

A. Homomorphic Encryption: The Basics

HE is a privacy-preserving encryption technique that enables computation over encrypted data, which was first shown to be possible by Gentry [24]. Since its discovery, many algorithmic improvements have been made to improve performance [6], [8], [9], [22], [25]–[27]. Modern HE schemes such as BFV allow adds and multiplies between encrypted data and derive security from the hardness of the Ring Learning With Error (RLWE) problem [38]. In BFV, noise is added during plaintext encryption and accumulates over successive ciphertext computations. If the aggregate noise exceeds a noise budget threshold, decryption fails. This noise budget is a function of the HE parameters and defines how many computations can occur before decryption fails. HE schemes of this type are called *Leveled* HE (LHE). In contrast, fully homomorphic encryption (FHE) schemes enable an arbitrary number of computations. FHE schemes can be built from LHE schemes via *bootstrapping* [22], [24]. Bootstrapping reduces the noise in the ciphertext but is expensive to implement, so most applications focus on LHE. We do not consider bootstraping and therefore use LHE; throughout this paper we refer to this LHE BFV scheme as HE for simplicity.

TABLE II: BFV parameters.

Parameter	Description
n	Polynomial degree (vector length)
t	Plaintext (pt) modulus
q	Ciphertext (ct) modulus
W_{dcmp}	Weight (pt) decomposition base
A_{dcmp}	Activation (ct) decomposition base
σ^2	Variance of noise added for encryption (fixed)

B. BFV: Relatively Efficient HE

BFV [22] is a relatively efficient LHE scheme; Figure 2 shows an overview of the process. In BFV, data is *encoded* as a plaintext polynomial that is then *encrypted* as a pair of ciphertext polynomials. Ciphertexts are then input as operands to addition and multiplication operations during *evaluation*. The resulting ciphertexts from evaluation are *decrypted* to plaintext and finally *decoded* to individual scalars. Polynomials are implemented as integer vectors, where the vector length (polynomial degree) and bit-width (coefficient size) are set by HE parameters. BFV parameters (listed in Table II) must be carefully tuned as they affect computational efficiency and security.

Core BFV Parameters (n, t, q): Plaintext polynomials are elements of the ring: $R_t = \mathbb{Z}_t[x]/(x^n+1)$, where the degree of the polynomial is less than n (a power of 2). Polynomial coefficients are integers in $\mathbb{Z}_t$ (integers in the range $(-\frac{t}{2}, \frac{t}{2}]$). t is called the *plaintext modulus* as all HE operations are taken modulo t in the plaintext space. Setting t requires profiling the application to ensure enough bits are used for correctness and no more, as over provisioning causes unnecessary slowdown.

Similarly, the two polynomials of a ciphertext are in $R_q = \mathbb{Z}_q[x]/(x^n+1)$, where q is the *ciphertext modulus*. The ratio between q and t determines the noise budget, which sets the number of HE operators that can be computed per ciphertext before decryption fails. The ratio between n and q for a given variance (σ^2) of Gaussian noise added for encryption sets the security strength of the HE scheme (see [22] for details).

Encoding (Packing) Data to Polynomial: When proper HE parameters are used (i.e., t is prime and $t \equiv 1 \mod 2n$), a property of the ring R_t enables a form of algorithmic parallelism. Here, each plaintext polynomial in R_t and, hence, the ciphertext, can be *packed* with n data. This means that each HE addition or multiplication can actually perform an n-way parallel element-wise computation. With packing, each scalar data is tied to a *slot*, and slots can be thought of as individual elements in the integer array. Packing significantly improves HE performance; n is typically on the order of thousands and the benefits of packing are proportional [60].

Polynomial Representations: Polynomials are represented in two spaces—coefficient and evaluation. The coefficient representation is how polynomials are typically represented, e.g., $\sum_{i=0}^{n-1} \alpha_i x^i$. The evaluation space is analogous to the frequency domain of time-domain signals. Similar to FFT, efficient conversion between the two is done via the Number Theoretic Transform (NTT) [9], [60]. Cheetah keeps polynomials in the evaluation space and converts to coefficient space only as needed for operations like decomposition (see below). Using the evaluation space as a default representation reduces the number of NTTs needed for homomorphic CNN/FC. Note that applying NTT to ciphertexts does not affect noise.

TABLE III: Impact on Noise of basic BFV operations.

	Noise Bound after Each Operation
Noise (v_0) in fresh ct$_0$	$2nB^2$ ($B = 6\sigma$)
`HE_Add`(ct$_0$, ct$_1$)	$v_0 + v_1$ (additive)
`HE_Mult`(pt, ct$_0$)	$nl_{pt}W_{dcmp}v_0/2$ (multiplicative)
`HE_Rotate`(ct$_0$)	$v_0 + l_{ct}A_{dcmp}Bn/2$ (additive)

1) Operations of BFV: BFV consists of three operators: `HE_Add`, `HE_Mult`, and `HE_Rotate`. Recall that the `HE_Add` and `HE_Mult` operate on vectors of packed data, so they are effectively SIMD-add and SIMD-multiply operations. Note that the underlying implementations of `HE_Add` and `HE_Mult` consist of many modular arithmetic calculations, different from a single-cycle integer add or multiply computation. Table III shows the amount of noise introduced by each operator, which depends on BFV parameter values. B is the bound of the noise added during encryption while v_i represents the initial noise in ciphertext ct$_i$. The remaining parameters (l_{pt}, l_{ct}, W_{dcmp}, and A_{dcmp}) are for decomposition, defined in Section III-B2.

`HE_Add`: Two ciphertexts can be added homomorphically by summing each ciphertext coefficient followed by a modulo operation. I.e., a resulting coefficient outside the range $\mathbb{Z}_q$ is reduced to be in $\mathbb{Z}_q$. Reduction is implemented as a comparison and subtraction to keep the performance overhead low. Each `HE_Add` operation increases noise additively.

`HE_Mult`: BFV supports both ct-ct and pt-ct multiplication. Cheetah uses pt-ct multiplication to multiply plaintext weights by encrypted activations. Pt-ct multiplication is achieved by multiplying evaluation space ciphertext polynomials by the evaluation space plaintext polynomial containing weights on a per-element basis. Performance is limited by the modular reduction required for each polynomial coefficient of output. Cheetah uses Barret reduction (see Section IV-A). `HE_Mult` operations increases noise multiplicatively.

`HE_Rotate`: BFV supports slot rotation within a packed polynomial to enable computation between data in different slots. Since `HE_Add` and `HE_Mult` are element-wise operations, computations like dot products require `HE_Rotate` to align partial products and implement the reduction (see Section V-A). `HE_Rotate` is computationally expensive with many steps, and increases noise additively. We refer the reader to [9], [67] for details.

2) Polynomial Decomposition: Decomposition is used to segment polynomials into multiple components with smaller-valued coefficients. The key idea is that HE operations over smaller coefficient polynomials reduces noise growth. To enable this, Cheetah has two parameters for polynomial decomposition: W_{dcmp} and A_{dcmp} (Table II), which defines the base that polynomials are decomposed to. Decreasing decomposition base increases the number of decomposed polynomials

TABLE IV: HE-PTune performance models.

CNN	HE_Mult	HE_Rotate
$n \geq w^2$	$l_{pt} c_i c_o f_w^2 / c_n$	$c_i c_o f_w^2 / c_n$
$n < w^2$	$l_{pt}(2c_n - 1) c_i c_o f_w^2$	$(2c_n - 1) c_i c_o (f_w^2 - 1)$
FC	**HE_Mult**	**HE_Rotate**
$n \geq n_i, n \geq n_o$	$l_{pt} n_i n_o / n$	$n_i n_o / n - 1 + \log(n/n_o)$
$n \geq n_i, n < n_o$	$l_{pt} n_i n_o / n$	$(n_i - 1) n_o / n$
$n < n_i, n \geq n_o$	$l_{pt} n_i n_o / n$	$(n_o + \log(n/n_o)) n_i / n$
$n < n_i, n < n_o$	$l_{pt} n_i n_o / n$	$(n-1) n_i n_o / n^2$

which decreases operator noise growth but increases the total amount of compute. Once decomposed operators complete, resulting segments are summed to get the final result.

HE_Rotate requires ciphertext decomposition, otherwise a single operation can exceed the noise budget. The decomposition base A_{dcmp} is used to factor ciphertext polynomials into multiple smaller-magnitude polynomials when HE_Rotate is applied. We denote $l_{ct} \approx \log_{A_{dcmp}}(q)$ as the number of polynomials with base A_{dcmp} resulting from the decomposition. Since HE_Rotate noise increase is additive, with decomposition noise increase by an additive factor proportional to A_{dcmp} and the increase in number of polynomial operations l_{ct}.

HE_Mult also benefits from decomposition to reduce noise. For neural networks, we use HE_Mult with decomposition to compute the partial products since weights are presented in plaintext. Using a decomposition base W_{dcmp}, the plaintext polynomial can be decomposed into $l_{pt} \approx \log_{W_{dcmp}}(t)$ polynomials. The resulting HE_Mult with decomposition requires l_{pt} polynomial multiplications to implement but reduces noise growth by a factor of around $t/(l_{pt} W_{dcmp})$.

IV. HE-PTune: Models & Parameter Tuning

HE parameter selection is a major source of complexity (i.e., setting n, t, q, W_{dcmp}, A_{dcmp}), where proper selection strikes a balance between noise budget and performance. A greater noise budget enables more computations per ciphertext but slower HE operators. Existing solutions rely on over-provisioning noise budgets, resulting in suboptimal performance. This section proposes HE-PTune: analytical performance and noise models for HE DNN operators to maximize performance via fine-grained parameter tuning. Tuning parameters with HE-PTune delivers up to a 11.7× speed up over the state-of-the-art.

A. Performance Modeling

HE-PTune's performance model analytically derives the total number of underlying integer multiplications per layer. (Recall that the HE operator HE_Mult consists of many integer multiplications.) Most HE operators resolve to multiplication operations and ones that do not have run-times either strongly correlated or dominated by those that do. Performance models for CNN and FC layers are built by first capturing all HE and NTT operations. Then all operations are reduced to the total number of underlying integer multiplication operations.

TABLE V: Noise models for CNN and FC layer.

CNN	Output Noise
$n \geq w^2$	$f_w^2 c_i \eta_M v_0 + \eta_A c_i (f_w^2 - 1 + (c_n - 1)/c_n)$
$n < w^2$	$(2f_w - 1) f_w c_i \eta_M v_0 + \eta_A c_i (2f_w + 1)(f_w - 1)$
FC	**Output Noise**
$n \geq n_i$	$n_i \eta_M v_0 + \eta_A (n_i - 1)$
$n < n_i$	$n_i \eta_M v_0 + \eta_A n_i (n-1)/n$

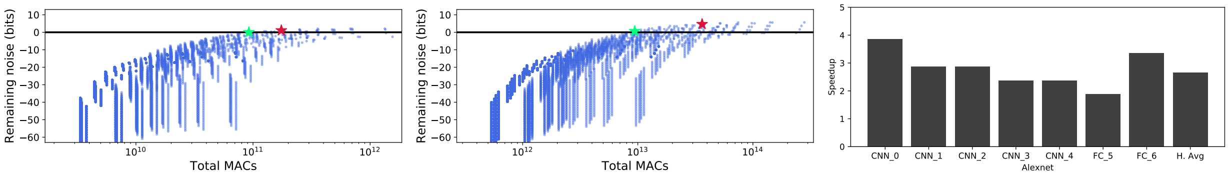
1) Modeling CNNs: CNN layers are parameterized as (w, f_w, c_i, c_o), where w^2 and f_w^2 represent the size of input image and weight filter, and c_i and c_o denote the number of input and output channels. Encryption parameters follow the notation defined in Table II. Effective modeling of HE_Mult and HE_Rotate counts requires consideration of two cases: 1) the ciphertext slot count is greater than an input image (i.e., $n \geq w^2$), and 2) the ciphertext slot count is less than an input image (i.e., $n < w^2$). We use c_n to model the relationship between ciphertext slots and input image size. c_n is defined as the number of input image channels per ciphertext (i.e., n/w^2) in the first case and the number of ciphertexts per input image channel (w^2/n) in the second. Table IV shows how each case counts the number of HE operations per CNN layer.

HE_Rotate operations require both polynomial multiplication and NTTs. Precisely, assuming a ciphertext decomposition base A_{dcmp}, $2l_{ct}$ multiplications and $l_{ct} + 1$ NTT ($l_{ct} \approx \log_{A_{dcmp}} q$) are required per HE_Rotate. Each n-point NTT entails $n \log n / 2$ butterflies. Cheetah uses Harvey's butterfly (3 integer-multiplications per butterfly). HE_Mult does not require NTTs as in Cheetah the default polynomial representation is the evaluation space. Each HE_Mult requires two element-wise modular multiplications between the two polynomials, resulting in $2n$ modular multiplications per HE_Mult. Cheetah uses Barrett reduction [34], which uses five integer-multiplications per reduction.

2) Modeling FCs: A similar process is repeated to model FC layers. The required number of integer multiplications per HE_Mult and HE_Rotate operations is the same in both CNN and FC, the only difference is the number of HE_Mult and HE_Rotate counts. Here, an FC layer is parameterized as (n_i, n_o), where n_i and n_o represent the number of input and output activations. The required number of HE_Mult and HE_Rotate for all possible cases are summarized in Table IV.

B. Noise Modeling

Once CNN/FC layers are implemented as HE operations (see Section V-A), noise growth can be modeled using the equations in Table III. We developed a model for layer noise as a function of both HE ($n, t, q, W_{dcmp}, A_{dcmp}$) and DNN ($f_w, w, c_i, c_o$ for CNN and n_i, n_o for FC) parameters. Note that directly applying the equations in Table III estimates noise in the *worst* case. The worst case is very rare (see below) and causes unnecessarily slow HE parameters to be selected.



(a) Layer5 (FC)

(b) Layer0 (CNN)

(c) Alexnet Speedup

Fig. 3: Comparison of HE-PTune and Gazelle using AlexNet. Blue dots are HE configurations modeled by HE-PTune. The red star is Gazelle's configuration and the green star is the optimal found by HE-PTune. Layer5 and Layer0 show the best and worst configuration for Gazelle with respect to utilized noise budget. HE-PTune's speedup for all layers on the right.

To overcome this, we develop practical noise estimations for HE operators and provide a theoretical analysis of the decryption failure rate. We also note that all prior work on high-performance HE [11], [29], [33] sets HE parameters using heuristics, providing high-likelihoods for success but not guaranteeing it.

Cheetah builds a theoretically-motivated, empirically-derived noise model that minimizes computational overheads for a targeted probability of success. We observe that added encryption noise is sampled from an independent bounded discrete Gaussian (IBDG) distribution with variance σ^2, and if X_i's are IBDG with variance σ_i^2, then $\sum_i \alpha_i X_i$ is also IBDG with variance $\sum_i \alpha_i^2 \sigma_i^2$. As the noise grows multiplicatively in `HE_Mult` and additively in `HE_Add` and `HE_Rotate`, we can compute the variance of the output noise after each layer under the independence assumption, which was validated in [19]. Then, since the output noise (Y) is IBDG with standard deviation (σ_Y), the probability of decryption failing is bound by $\Pr(|Y| \geq q/(2t)) \leq 2\exp(-q^2/(4t^2\sigma_Y^2))$. We use these equations to derive an output noise threshold for a probability of correct decryption. Therefore, instead of using worst-case bounds and guaranteeing correct decryption, our noise model uses the scaled expressions given in Table III. Cheetah uses a scaling factor c such that the decryption failure rate is provably less than 10^{-10}, which is negligible as it is much lower than the DNN's misclassification rate.

The noise models are given in Table V. Here, v_0 is the initial noise for the input ciphertext, η_M is the noise due to `HE_Mult`, and η_A is the growth factor from `HE_Rotate`. By dividing $\frac{q}{2t}$ by the output noise (and taking the log), the remaining noise budget in bits is given. When the budget is negative, decryption fails; when positive, it fails with probability $\leq 10^{-10}$.

C. HE Parameter Space Exploration

Using a single set of HE parameters for all DNN layers results in poor performance, as HE parameters are provisioned for the worse case layer noise. Using HE-PTune's models for noise and performance, parameters can be readily tuned on a per-layer basis. HE-PTune takes layer hyperparameters as input and outputs optimal HE parameters found via a design space exploration. Because the model is analytical, a vast parameter space can be explored in a matter of minutes.

Examples of HE parameter space exploration are given in Figure 3 for AlexNet on ImageNet. Each blue dot is unique set of HE parameters modeled with HE-PTune to estimate computation and remaining noise budget. Red stars indicate parameters used by Gazelle and green stars show the optimal point found using HE-PTune. Gazelle uses the same sets of HE parameters for all layers. Of all layers in the model, Layer 5 has the smallest remaining noise budget, and it follows that the speedup between Gazelle and Cheetah is the lowest for this layer (see bars in Figure 3). Using HE-PTune, empirical results show using a single set of parameters is inefficient and unnecessary. The highest Cheetah speedup is in Layer 0, where Gazelle has an excess noise budget of 4.6 bits whereas HE-PTune finds a configuration leaving only 1 bit of noise budget. Improvements come from tailoring parameters to the requirements of each layer.

HE-PTune also eases finding functional HE parameter settings in the first place. Recall that any point where the noise budget is exceeded fails to decrypt. Of all the points evaluated in the design space search, over 99% have a negative remaining noise budget and will not work. Finding HE parameters is difficult, further motivating HE-PTune.

We validated HE-PTune using different CNN and FC layers used in popular DNNs, including: LeNet-300-100 and LeNet5 for MNIST [36], and AlexNet [35], VGG16 [59], and ResNet50 [31] for ImageNet [53]. Each layer is tested using a variety of HE parameters with no consideration of noise budget to explore the parameter space. Execution times are collected by implementing each CNN/FC layer in the SEAL HE library [57] and measuring its performance on a Xeon server. The remaining noise budget is collected after each run using SEAL's internal measuring capability and API. Overall, we find that due to the underlying randomness of the noise, the noise model shows slightly larger error than the performance model. However, this is acceptable as the worst-case errors are within 1 bit in the low-remaining noise budget region of the space.

V. Partial-aligned Scheduling

This section introduces a new dot product schedule, named Sched-PA, to improve HE performance on FC and CNN layers. Recall that each HE primitive has different run-time and additive noise trade-offs (Section III) and the overheads of different primitive schedules are not associative so order of operations

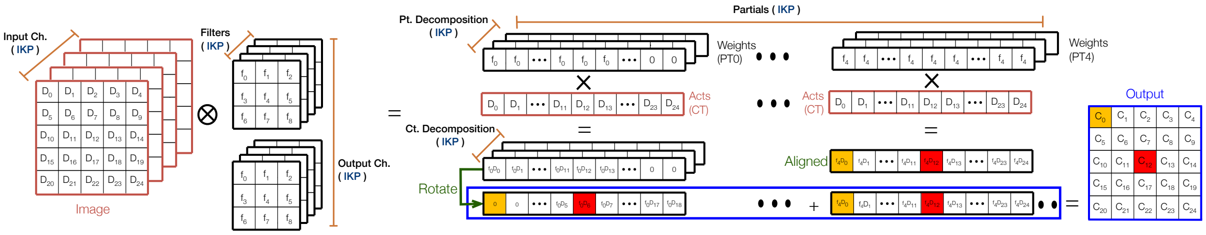


Fig. 4: How Cheetah implements CNNs using Sched-PA. Sources of inter-kernel parallelism (IKP) are labeled.

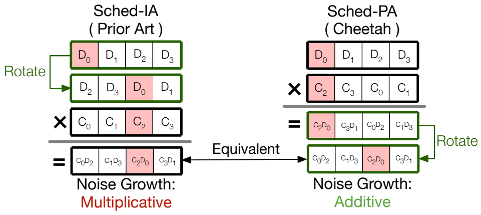


Fig. 5: Sched-IA (input-aligned) versus Sched-PA (partial-aligned) dot product schedules. Cheetah uses Sched-PA to improve performance of CNN and FC layers.

matters. Operation orderings with less noise are beneficial as it enables higher-performance via more computationally efficient HE parameters.

A. Sched-PA: Partial-Aligned Dot Products

The key challenge for implementing HE dot products is optimizing how data is packed into polynomial slots and the relative order of operations. Computing a dot product in HE requires all three primitives: HE_Mult, HE_Add, and HE_Rotate. Partial products are computed using an HE_Mult operation between a ciphertext (encrypted activation) and a plaintext (model weights). Each partial is accumulated with a series of HE_Add operations to reduce the final output. The challenge is that HE operations only support computation between *aligned* polynomial slots. This means that when polynomial A and B are multiplied (resulting in C), $\mathrm{C[i]} = \mathrm{A[i]} \times \mathrm{B[i]}$, $\forall i \in [0,n)$. To properly reduce each of the partials of a dot product, the slots in C must be aligned to use the correct values.

Prior work aligns the inputs before performing multiplication, referred to here as an *input-aligned* schedule (Sched-IA) [30], [33]. In Sched-IA, the input ciphertext is first aligned, or rotated, to the correct output slot, and plaintext weights are packed appropriately. The post-rotation ciphertext and plaintext are then multiplied, resulting in a dot product partial (ciphertext). Resulting partial ciphertexts can be readily accumulated to compute the final value.

Cheetah proposes a new dot product implementation called *Sched-PA* (see Figure 4, 5). Our key insight is that HE_Mult increases noise by a multiplicative factor η_M ($\leq nl_{pt}W_{dcmp}/2$) whereas HE_Rotate is additive η_A. In Sched-PA, the initial input ciphertext is always kept in its original order. Weights are again packed into a plaintext polynomial and aligned with ciphertext slots to compute the correct partial product via HE_Mult. Finally, resulting partial product ciphertexts are aligned such that the partial slot matches the correct output slot. Figure 5 also shows Sched-PA compared to the other approach.

The benefit stems from noise accumulation in chained HE operations. Recall that v_0 and η_A represent the initial input ciphertext noise and additive noise from HE_Rotate, respectively. Thus, a dot product using the partial aligned schedule experiences a noise growth of $\eta_M v_0 + \eta_A$. In contrast, the Sched-IA dot product first rotates *then* multiplies, resulting in noise growth of $\eta_M(v_0 + \eta_A)$, significantly larger than Sched-PA. Saving noise enables HE-PTune to identify higher performance HE parameter settings, ultimately resulting in performance benefit.

B. Implementing Low-Noise Convolution

Figure 4 shows an example of how CNNs are implemented in HE using Sched-PA. FC layers follow precisely the same steps as CNNs, as the core primitives are also dot products. First, the input activation ciphertext (Acts) is encoded by placing adjacent pixels from the client's image sequentially in polynomial slots. This ordering eases partial ciphertext alignment. Next, CNN filter weights (Filter) are encoded into plaintext polynomials. Each activation-weight polynomial is multiplied with HE_Mult to compute the partials. The resulting partial polynomials are then rotated to align partial slots to the proper output-neuron slot. Finally, with all partials computed and aligned, the ciphertexts are reduced with HE_Add. Note how polynomial slots allow multiple output neurons to be computed in single ciphertext. This algorithmic parallelism provides substantial performance and memory savings for HE as without it, each thousand degree polynomial would only compute a *single* output neuron.

The zeros found in weight plaintext slots (e.g., PT0) ensure the correct computation. For example, the red slot in Figure 4 shows how accumulation works. After f_0 is multiplied to D_6 in the first HE_Mult, the result is rotated right 6 times to be accumulated in the red slot (C_{12}). When this rotation is performed, D_{19} aligns to slot 0, however $f_0 D_{19}$ should not be accumulated in the output of slot 0 (i.e., C_0). Selectively adding zeros in the plaintext slots avoids this boundary case.

C. Evaluation Results

The effectiveness of Sched-PA is evaluated using five standard CNN models. HE-PTune is employed to maximize benefits and tune HE parameters on a per-layer basis. Multiple

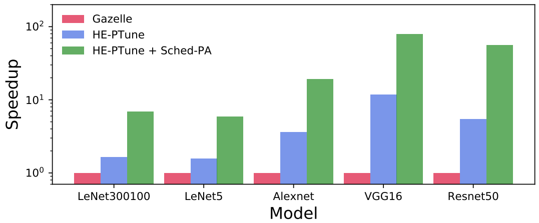


Fig. 6: Per-benchmark speedup achieved by Cheetah using HE-PTune and Sched-PA. Speedup is relative to Gazelle [33].

experiments are run to show the benefits of HE-PTune and Sched-PA independently and relative to Gazelle.

The results for each model are shown in Figure 6. Overall, the Cheetah optimizations substantially outperform Gazelle. Using the harmonic mean, 2.98× speedup comes from HE-PTune alone (5.25× ignoring MNIST). Sched-PA provides an additional speedup of 5.20× (6.11× ignoring MNIST) for a total mean performance improvement of 13.5× and maximum of 79.5× over Gazelle (30.3× mean without MNIST).

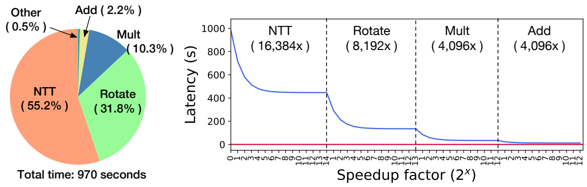
Significant performance overheads are incurred by Gazelle as Sched-IA requires substantial ciphertext and plaintext decomposition. Each time a polynomial is decomposed to reduce noise, the number of polynomials that must be computed grows proportionately. In ResNet50, Cheetah's optimizations result in a ciphertext decomposition base of 8 to 16 more bits. A higher ciphertext decomposition bases result in fewer decomposed polynomials for HE_Rotate, and substantial performance improvements. With Sched-PA, Cheetah avoids all plaintext decomposition.

VI. Profiling HE Inference

HE-PTune and Sched-PA significantly improve the performance over the state-of-the-art [33], e.g., 55.6× for ResNet50. However, with these optimizations alone HE inference is still 3-4 orders of magnitude slower than plaintext inference, i.e. unencrypted inference on a CPU. To better understand performance bottlenecks we profile a software implementation of HE inference and compute the speedup needed from hardware acceleration.

We implement ResNet50 in HE using the SEAL library [57]. Using Cheetah to tune parameters and maximize performance, one HE inference takes 970 seconds on an Intel Xeon E5-2667 server. The same unencrypted inference (on the same server) takes 100 milliseconds using Keras [14]. Since SEAL only supports CPUs, we perform profiling on the CPU platform. Below we benchmark NTT running on a GPU.

Profiling results are summarized in the pie chart of Figure 7. Notice that only a few kernels dominate performance (HE_Mult, HE_Add, HE_Rotate, and NTT). HE_Rotate in Figure 7 does not include NTT as this is shown separately. Of the four, NTT is the primary bottleneck taking 55.2% (535 seconds) of the run time. The SEAL profile also contains a long-tail of small functions, labeled "Other" in Figure 7. We note that most of the "Other" function time is in construction/destruction.



(a) Time breakdown

(b) Speedup Needed

Fig. 7: Profiling results for ResNet50 and speedup needed by each kernel to match real-time inference latency.

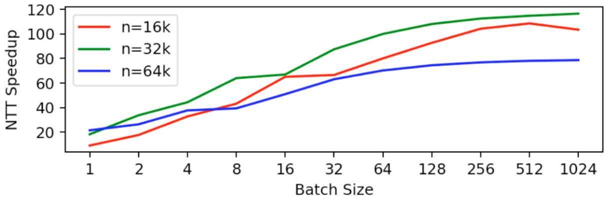


Fig. 8: NTT GPU speedup over CPU.

Using the profiling results, we compute the speedup needed from each HE kernel to achieve real-time inference latency. Figure 7 shows the results of an Amdahl's law-like limits study of how various speedup factors impact overall run time. The x-axis shows the speedup factor applied to each kernel function (note the log scale); the final speedup factor for each kernel is the speedup needed (e.g., 16,384 for NTT). The y-axis shows absolute latency. From left to right, the plot shows how the total inference latency decreases as each theoretical speedup factor is applied to each function. Kernel speedup is applied successively where the run time from the most aggressive speedup factor is taken as the base for the next function. The horizontal red line indicates the 100ms real-time inference latency target.

Speeding up HE with GPUs: One way to improve kernel performance is with GPUs. To understand the the limitations of HE on GPUs, we benchmark NTT, the primary HE bottleneck, using the cuHE library [16] on an NVIDIA 1080-Ti GPU. GPU speedup is reported for different NTT batch sizes (1 to 1024) and vector lengths n = 16K, 32K, and 64K (Figure 8). At larger batch sizes (512/1024), the speedup saturates at 120×. The nvprof profiler shows that for a batch size of 512, the GPU is utilized with 70% warp occupancy and 85% warp execution efficiency.

Other first order limitations to performance likely derive from (a) non-native, long integer data types requiring emulation, (b) modular arithmetic, which adds branch instructions and over 10 compute instructions per multiplication. Despite the two orders of magnitude speedup, GPUs fall well short of the improvements required to reach real-time speeds.

VII. HE Inference Accelerator Architecture

This section proposes a general accelerator architecture for HE inference to bridge the remaining performance gap.

A. Accelerator Architecture

The proposed accelerator architecture is shown in Figure 9. At a high level, it is composed of ciphertext (CT) *processing*

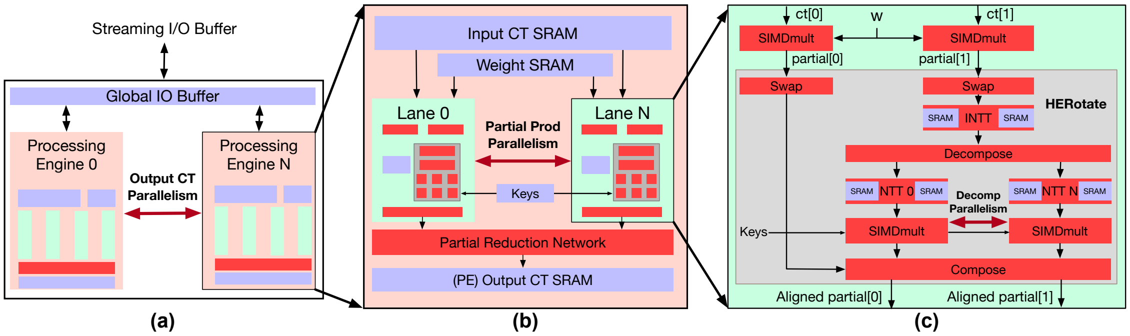


Fig. 9: Cheetah accelerator architecture. (a) The accelerator is composed of parallel PEs operating in output stationary fashion. Off-chip data is communicated via a PCIe-like streaming interface, and data is buffered on-chip using global PE SRAM. (b) Each PE contains Partial Processing Lanes which compute the HE dot product. (c) Lanes comprise individual HE operators.

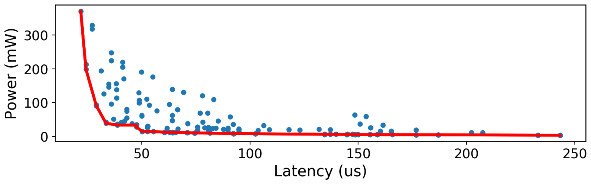


Fig. 10: DSE for NTT; Pareto frontier in red.

engines (PEs) that receive data from a PCIe-like streaming interface and buffer intermediary results in SRAMs (Figure 9a). Hierarchically, PEs are composed of *partial processing lanes* or *Lanes*, and a *partial reduction network*, which implement the HE dot product (Figure 9b). Lanes are further decomposed into individual HE operators (Figure 9c).

1) PEs (Output Neuron Engines): Our architecture is designed to maximize performance and parallelism by being output-stationary. Each PE processes a single ciphertext of output neurons and all compute-memory resources necessary for the output are local to the PE; the number of PEs is parameterized. When there are more output neurons per layer (Parallel Output CTs) than physical PEs, we time multiplex the computation across multiple PE executions. The PEs are connected to input and output buffers used to route data to and from the host. These buffers constitute small SRAMs as they only handle communication, all state and intermediates are local to PEs.

The internals of the PE contain partial processing lanes and reduction networks. Each PE contains an *Input CT buffer* to store a copy of activation CTs locally, SRAM for rotation keys, a relatively small SRAM for weights, a set of partial processing lanes, a partial reduction network, and output CT SRAM. Each Lane is capable of processing a unique dot product partial; the number of lanes is parameterizable. Lanes within a PE operate in lockstep to enable reuse of twiddle factor SRAMs required for NTTs. The partial reduction network is configured based on the number of partials computed in parallel (i.e., number of Lanes). Input CT SRAMs are provisioned with enough capacity to double buffer inputs with sufficient bandwidth to feed all Lanes.

2) Lanes (Partial Engines): Lanes are the backbone of the accelerator and implement the HE operators. In Figure 9c, HE kernel blocks are denoted in red. Intermediary SRAMs, shown in blue, are used to store results between HE kernels. We use SRAMs instead of off-chip DRAM for intermediary results because of the high internal bandwidth required within NTT modules to support high degrees parallelism. In the worst case, each NTT kernel requires 13 GiB/s of bandwidth; each lane contains multiple NTTs and each PE contains many lanes. Aside from the NTT kernels, which have a strided memory access pattern, all operations within a Lane can be made streaming (i.e, no SRAMs needed after kernels). This allows the architecture to save SRAM resources. The NTT activation decomposition factor A_{dcmp} introduces a parametrizable degree of inter-NTT parallelism within a Lane. For high-performance, enough lanes are allocated to execute all decomposed values in parallel.

The lane architecture shows the datapath and kernel dependencies to compute a single partial dot product. Both input polynomials (CT[0] and CT[1]) are first multiplied by plaintext weights using the `HE_Mult` operator, outputting partial polynomials. The datapaths diverge as the BFV scheme splits the compute asymmetrically between partial[0] and partial[1]. `HE_Rotate` is applied to perform polynomial slot alignment. For partial[1], inverse NTT (INTT), decomposition, NTT, and composition units are applied. The datapath for partial[1] splits after the INTT computation in order to implement ciphertext decomposition. Recall that decomposition reduces noise growth; however, the trade-off manifests here as additional compute requirements. Fortunately, the additional computations can be parallelized (note the multiple NTT and SIMDmult units). Each decomposed polynomial is then run through a parallel NTT block to return to the evaluation domain where the rotation key is applied using parallel SIMDmult accelerators. These decomposed, rotated polynomials are then composed and combined with swapped partial[0] to produce the aligned partial that is fed to the partial reduction network in the PE, which consists of SIMDadd units.

VIII. Experimental Results

This section presents the design space exploration results of the parameterized accelerator architecture. We split sources of hardware speedup into two parts: intra- and inter-kernel parallelism. Intra-kernel parallelism refers to speedup achieved leveraging parallelism within a kernel (e.g., NTT) in hardware whereas inter-kernel parallelism is realized by allocated multiple instances of the same hardware kernel (e.g., three NTT accelerator units) to process independent inputs simultaneously. We show that by combining Cheetah's algorithmic optimizations with large-scale custom hardware, HE inference can approach real-time performance.

A. Methodology

Design space exploration consists of sweeping various accelerator microarchitectural parameters. Each kernel (`HE_Mult`, `HE_Add`, and `HE_Rotate`–which is split into Swap, INTT, Decompose, NTT, SIMDMult, and Compose) is built using Catapult HLS 10.3d [1] and synthesized with a commercially-available 40nm standard cell library targeting 400 MHz. For each kernel we evaluate hundreds of design points to explore different design tradeoffs and identify optimal implementations. Each kernel accelerator's microarchitecture is parameterized by memory bandwidth (or I/O in the case of streaming kernels), datapath parallelism (i.e., hardware loop unrolling), and pipelining (i.e., initiation-interval). We estimate power, performance, and area using Catapult's output RTL and power analysis flows. We use a commercial SRAM compiler to compile each SRAM dimension used across different design points due to different memory tiling factors.

Based on these kernel design sweeps, we select Pareto optimal points and use them to further identify optimal HE accelerator designs using a simulator for the architecture presented in Figure 9, see Section VII for details. The simulator takes HE parameter settings and user defined accelerator microarchitectural parameters (HE kernel implementation, number of PEs, and number of lanes) as input. We swept PEs per accelerator from 2-1024 and lanes per PE from 4-8192. Area is estimated based on architectural parameters alone while power and latency are derived through simulating layers running on the modeled hardware. To estimate performance and power for an input DNN, each layer is represented as the number of input/output ciphertexts and partials per output ciphertext. The simulator then maps and multiplexes the number of output neuron ciphertext to available PEs and partials to Lanes to derive hardware activity factors and energy consumption. Combining multiplexing and activity factors with HLS latency and power results estimates accelerator performance. The overall performance of a full inference is modeled on a per-layer granularity; this is because after each layer's linear computations, activations are sent to the client for ReLU and Pooling.

To capture the benefits of technology scaling, we report power and area estimates for 5nm using foundry-reported scaling factors. Specifically, we use 0.2× power and 0.22× area to scale from 40nm to 16nm, based on [44], [45], [64], [68]. From 16nm to 5nm, the power and area scaling factors are 0.28× and 0.17×, using [58] and recent data from [70]. Together, the power and area scaling factors (40nm to 5nm) are 0.056× and 0.038×, respectively.

B. Evaluation Results

1) Intra-Kernel Parallelism: We begin by first sweeping each kernel's microarchitectural parameters in HLS to explore the design space and understand the power-latency trade-offs of intra-kernel parallelism. An example design space Pareto frontier for NTT is shown in Figure 10. Recall that these frontiers are used as the cost model for the larger architecture, whose sweeps consider the power-latency tradeoffs of each kernel to estimate HE inference accelerator characteristics.

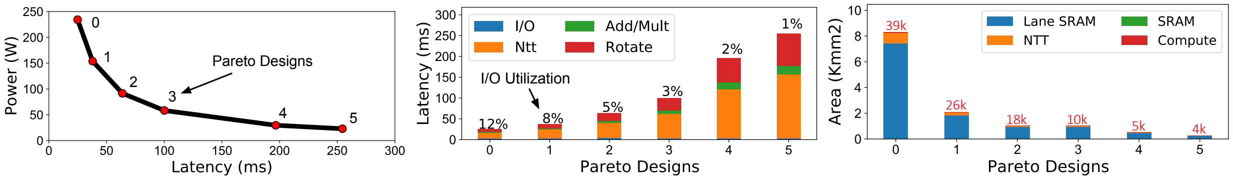
Speedup is computed relative to the SEAL library implementation of kernels running on a 3GHz Intel Xeon (Skylake) server. We observe modest speedups of individual kernels, with a maximum of 40× and averaging roughly 10× across designs. The `HE_Add` and `HE_Mult` kernels provide substantial parallelism as the underlying computation consists of element-wise modular additions and multiplications, which are easily parallelized. The `HE_Rotate` (Swap, Decompose, Compose) and NTT kernels contain a mix of sequential dependencies and code regions easily parallelized, such as the element-wise multiplications and butterfly computations. The key takeaway is that intra-kernel parallelism can improve HE performance by roughly one order of magnitude.

2) Inter-Kernel Parallelism: Fortunately, DNNs and HE contain abundant parallelism across kernel calls. With the exception of kernel dependencies within a Lane and the reduction of partial products in PEs, partials and output neurons can be executed in parallel by allocating more hardware resources.

For example, consider CNN Layer6 in ResNet50 ($f_w = 3$, $w = 64$, $c_i = c_o = 64$). If each ciphertext contains a single input channel ($n = 4096$), then all partial products can be computed with 36,864 `HE_Mult` and `HE_Rotate` parallel kernel invocations. The partial products for these layers cannot be parallelized since `HE_Mult` must be performed before `HE_Rotate` under Sched-PA. In `HE_Rotate`, domain conversion from evaluation to coefficient using INTT must be done before decomposition, but the NTT to convert back the domain of decomposed polynomials can be parallelized. As a result, we find that *the degree of parallelism that can be exploited at the Lane and PE level is on the order of thousands for ResNet50*. The key takeaway is that application inter-kernel parallelism exposes two to three orders of magnitude performance improvement.

3) Lane and PE DSE: When combined, inter-kernel and intra-kernel parallelism can be exploited to bridge the remaining 3-4 order of magnitude speedup needed to approach real-time inference performance. We conduct a design space exploration of accelerator microarchitecture parameters to evaluate whether these designs leveraging these degrees of parallelism are practical with respect to fabrication constraints [65]. The sweeps consider each kernel's power-latency Pareto frontier of designs, number of Lanes per PE, and PE's per accelerator.

Figure 11 shows the results from the design space exploration of ResNet50. The power-latency Pareto points identified in the left-most subplot shows the ideal architectures when designing an accelerator tuned for ResNet50. The Pareto frontier provides insight into the hardware cost-per-ms tradeoff



(a) ResNet50 DSE

(b) Time Breakdown

(c) Area Breakdown

Fig. 11: (a) Power-latency Pareto for ResNet50 DSE. (b) Run time breakdown for each Pareto design point. (c) Energy and area breakdown for each Pareto design point. Red numbers on top of the bars indicate accelerator speedup over SEAL's software implementation.

of inference latency. For ResNet50, we find the Pareto optimal design point dissapates 30W of power and takes 587mm^2 for ResNet50, which are within practical (albeit high) fabrication limits and not uncommon for datacenter coprocessors [61].

The low power density is due to aggressive SRAM tiling to meet the high internal bandwidth targets for NTT units. Upon further analysis, we find that the 128×60 bit SRAM sizes have a bit density that is ≈ 2.5× worse than larger 1024×60 SRAMs, which results in low power density. We also note that the 400 MHz clock target is low for a 5nm technology, furthering reducing power density. When scaling from 40nm to 5nm we did not scale frequency to be conservative, any frequency scaling would improve estimated results.

To understand the limitations to efficiency and performance of each Pareto design point, Figure 11 shows the Pareto optimal design result for ResNet50 (AlexNet, VGG16, and MNIST exhibit similar trends). Figure 11a shows six design points on the Pareto frontier. Figure 11b and Figure 11c show the breakdown of run time and area respectively for these six design points. For extreme low-latency designs (Pareto points 0 and 1), results show that most of the design area goes into small SRAMs that are required to support the high internal bandwidth required by NTT units (discussed next). As a result, this leads to impractically large area overheads.

Overall, the results in Figure 11b confirm NTT and reduction (`HE_Rotate`) dominate HE accelerator computation cost. Recall NTT is data intensive and has many small internal SRAMs, which at extreme design points result in high power and area usage. This is compounded by the sheer number of NTT units that operate in parallel, making NTT computations the largest overall area component. We note that even in the most parallel design point considered, the accelerator is compute bound (I/O utilization is only 12%) and NTT remains the primary bottleneck. Moreover, we find that the input and output SRAMs in the architecture do not incur as high of a power and area cost. This means that the input duplication into each PE to support output-stationary computation is relatively inexpensive.

4) Accelerator Generality: Designing a fixed-size HE accelerator for each DNN model is impractical. Instead, the accelerator can be programmed to support different-sized networks by multiplexing compute logic (PEs and Lanes) to handle different DNN tensor shapes. To quantify the loss associated with under utilized units stemming from imperfect dimension matching, we measure performance loss for different ImageNet models (AlexNet and VGG16) running on the HE accelerator optimized for ResNet50.

TABLE VI: Performance of running VGG16 and AlexNet on PT-ResNet50 accelerator. Prt is partials per output CT.

	Practical			Match CPU			
Model	Lat.	Area	PEs-Lanes	Lat	Area	PEs-Lanes	Slowdown
ResNet50	198	587	4-512	101	1173	8-512	0%
VGG16	386	587	16-128	104	2347	32-256	113%
AlexNet	111	587	16-128	61	1174	32-128	26%

Table VI shows each ImageNet model running on three distinct architectures. The Practical column shows the highest performing design achieved when respecting the reticle limit of EUV fabrication [65]. Match CPU ignores these limits and reports the chip area needed to match the performance of running a plaintext inference on the Xeon server, which coincides with our mark for real-time performance with ResNet50 and VGG16 at 100ms. The architecture of each design is reported as PEs-Lanes, the number of PEs and Lanes per PE used by the accelerator, respectively. Note that the NTT speedup of ResNet50's Match CPU design confirms the estimates from Figure 7. The 8 PEs and 512 Lanes per PE provide 4,096 parallel Lanes. Then within each lane, NTTs run 1.5× faster than the CPU and the three (out of the four) decomposition NTTs run in parallel. Thus the total NTT speedup is: $8 \times 512 \times (3 \times \frac{3}{4} + \frac{1}{4}) \times 1.5 = 15{,}360$.

Slowdown shows the latency penalty incurred by AlexNet and VGG16 when they are executed on ResNet50's Match CPU architecture compared to their optimal Match CPU architectures. We find both experience considerable slowdown. This is due to the choice of PE and Lane allocations and the differences in layer dimensions. AlexNet and VGG16 layers have a higher average number of output CTs per layer and partials per output CT than ResNet50. The high number of output CTs makes it straightforward to parallelize work across PEs. However, ResNet50 is very structured given its use of bottleneck layers, many of which have partials per output ciphertext that are divisible by or less than 512, yielding high Lane utilization within a PE. Conversely, VGG16's partials per output CT tend to fall just above multiples of 512, (e.g., 34, 687, 1086) resulting in poor utilization. Thus, AlexNet and VGG16 favor more PEs with fewer Lanes. Looking forward, research of less rigid architectures is needed to improve design efficiency.

IX. Related Work

A growing interest in privacy and machine learning has resulted in a body of related work on developing cryptographic solutions. Techniques can be categorized into two groups: HE only [11], [29], [32], [55], or multiparty computation (MPC)-based [33], [37], [50], [51]. While each has significantly advanced the field all suffer from either accuracy loss due approximation or high communication/computation overheads.

HE only techniques must address evaluating non-linear functions (e.g., ReLU, MaxPool) using only available addition and multiply operations. CryptoNets [29], CryptoDL [32], and LoLa [11] propose replacing ReLU with low-order polynomials that can readily be computed with HE primitives. However, even with square activations [11], this requires very large HE parameters (e.g., $q \approx 1000$ [29], 440 bits [11], while Cheetah uses 60) for an appropriate noise budget. Moreover, approximate activation functions require re-training [11] and can degrade accuracy [29]. Others propose accelerating HE kernels with accelerators. NTT has been ported to FPGAs [48], [52] and GPUs [2], [3], [17] to speedup polynomial multiplication. Raizi et al [49] propose HEAX to accelerate HE kernels with FPGAs but only reports two orders of magnitude speedup. While related, the results of HEAX are orthogonal to the contributions of this paper; HEAX uses CKKS (Cheetah uses BFV). Mostly, above works focus on ciphertext-ciphertext multiplication (Cheetah uses ciphertext-plaintext), and targets kernel acceleration (Cheetah focus on the application of DNN inference and general chip architecture).

MPC-based schemes provide an alternative to approximation by combining HE with other security solutions, typically garbled circuit (GC) [33], [37], [42], [46], [50], [51]. Among them, Gazelle is considered the state-of-the-art [33]. Gazelle uses HE for linear layers in the cloud and GC [69] for ReLU and MaxPool on the client. This can significantly improve the latency for small models but results in a severe computational bottleneck in deep models (e.g., ResNet50). Cheetah takes Gazelle as a baseline and focuses on reducing the significant computational overheads of HE.

Other work assumes different threat models with non-cryptographic solutions. E.g., [10], [62] use TEEs to isolate private data from untrusted software. Others have looked at limiting information leakage by adding noise (similar to DP) [40]; this provides increased average-case privacy with negligible loss in accuracy.

X. Conclusion

This paper presents Cheetah, a series of optimizations for improving private inference performance using homomorphic encryption. We develop algorithmic and schedule-based optimizations, HE-PTune and Sched-PA, to provide up to a 79× performance improvement over the state-of-the-art. Profiling the optimized implementation using SEAL, we find the inference performance is still four orders of magnitude slower than real-time requirements. We then build hardware models and conduct design space explorations of various accelerator microarchitectures to understand the degree of speedup hardware acceleration can deliver. Our estimates indicate that, with a 587mm^2 chip, a ResNet50 inference can be processed in 198ms, which is within a small constant factor of real-time constraints for some classes of applications. Looking forward, more research on flexible accelerator architectures for HE and better NTT accelerator designs would improve results.

XI. Acknowledgements

This work was partially supported by the Applications Driving Architectures (ADA) Research Center, a JUMP Center co-sponsored by SRC and DARPA.

References

[1] "Catapult high-level synthesis," https://www.mentor.com/hls-lp/catapult-high-level-synthesis/, accessed: 04-16-2020.

[2] S. Akleylek, O. Dağdelen, and Z. Y. Tok, "On the efficiency of polynomial multiplication for lattice-based cryptography on GPUs using CUDA," in *International Conference on Cryptography and Information Security in the Balkans*. Springer, 2015, Conference Proceedings, pp. 155–168.

[3] A. Al Badawi, B. Veeravalli, C. F. Mun, and K. M. M. Aung, "High-performance FV somewhat homomorphic encryption on GPUs: An implementation using CUDA," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 70–95, 2018.

[4] Apple Differential Privacy Team, "Learning with privacy at scale," *Apple Machine Learning Journal*, vol. 1, no. 8, pp. 1–25, 2017.

[5] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, "Efficient garbling from a fixed-key blockcipher," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 478–492.

[6] J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig, "Improved security for a ring-based fully homomorphic encryption scheme," in *IMA International Conference on Cryptography and Coding*. Springer, 2013, pp. 45–64.

[7] 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.

[8] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Advances in cryptology–crypto 2012*. Springer, 2012, pp. 868–886.

[9] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, p. 13, 2014.

[10] F. Brasser, T. Frassetto, K. Riedhammer, A.-R. Sadeghi, T. Schneider, and C. Weinert, "VoiceGuard: Secure and private speech processing." in *Interspeech*, 2018, pp. 1303–1307.

[11] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *International Conference on Machine Learning*, 2019, pp. 812–821.

[12] K. Chaudhuri and D. Hsu, "Sample complexity bounds for differentially private learning," in *Proceedings of the 24th Annual Conference on Learning Theory*, 2011, pp. 155–186.

[13] W.-S. Choi, M. Tomei, J. R. S. Vicarte, P. K. Hanumolu, and R. Kumar, "Guaranteeing local differential privacy on ultra-low-power systems," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, June 2018, pp. 561–574.

[14] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[15] V. Costan and S. Devadas, "Intel SGX explained." *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.

[16] W. Dai, "cuHE: CUDA homomorphic encryption library." [Online]. Available: https://github.com/vernamlab/cuHE

[17] W. Dai and B. Sunar, "cuHE: A homomorphic encryption accelerator library," in *International Conference on Cryptography and Information Security in the Balkans*. Springer, 2015, Conference Proceedings, pp. 169–186.

[18] B. Ding, J. Kulkarni, and S. Yekhanin, "Collecting telemetry data privately," in *Advances in Neural Information Processing Systems*, 2017, pp. 3571–3580.

[19] L. Ducas and D. Micciancio, "FHEW: bootstrapping homomorphic encryption in less than a second," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 617–640.

[20] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[21] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. ACM, 2014, pp. 1054–1067.

[22] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption." *IACR Cryptology ePrint Archive*, vol. 2012, p. 144, 2012.

[23] P. Fenner and E. O. Pyzer-Knapp, "Privacy-preserving Gaussian process regression – a modular approach to the application of homomorphic encryption," 2020.

[24] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, ser. STOC '09. New York, NY, USA: ACM, 2009, pp. 169–178. [Online]. Available: http://doi.acm.org/10.1145/1536414.1536440

[25] C. Gentry, "Computing arbitrary functions of encrypted data," *Commun. ACM*, vol. 53, no. 3, pp. 97–105, Mar. 2010. [Online]. Available: http://doi.acm.org/10.1145/1666420.1666444

[26] C. Gentry, S. Halevi, and N. P. Smart, "Fully homomorphic encryption with polylog overhead," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012, pp. 465–482.

[27] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 75–92.

[28] Z. Ghodsi, A. Veldanda, B. Reagen, and S. Garg, "Cryptonas: Private inference on a relu budget," 2020.

[29] 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, pp. 201–210.

[30] S. Halevi and V. Shoup, "Algorithms in HElib," in *Annual Cryptology Conference*. Springer, 2014, pp. 554–571.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

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

[33] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "Gazelle: A low latency framework for secure neural network inference," *arXiv preprint arXiv:1801.05507*, 2018.

[34] J. Katz, A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1996.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

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

[37] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via MiniONN transformations," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 619–631.

[38] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 1–23.

[39] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella *et al.*, "Fairplay-secure two-party computation system." in *USENIX Security Symposium*, vol. 4. San Diego, CA, USA, 2004, p. 9.

[40] F. Mireshghallah, M. Taram, P. Ramrakhyani, D. Tullsen, and H. Esmaeilzadeh, "Shredder: Learning noise distributions to protect inference privacy," *ASPLOS*, 2019.

[41] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 2505–2522. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/mishra

[42] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017, pp. 19–38.

[43] P. Mohassel and P. Rindal, "ABY3: a mixed protocol framework for machine learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 35–52.

[44] NanotechnologyProductsDatabase, "TSMC 20nm Technology." [Online]. Available: https://product.statnano.com/product/6776/tsmc-20nm-technology

[45] NanotechnologyProductsDatabase, "TSMC 28nm Technology." [Online]. Available: https://product.statnano.com/product/6777/tsmc-28nm-technology

[46] C. Orlandi, A. Piva, and M. Barni, "Oblivious neural network computing via homomorphic encryption," *EURASIP Journal on Information Security*, vol. 2007, no. 1, p. 037343, 2007.

[47] N. Papernot, M. Abadi, U. Erlingsson, I. Goodfellow, and K. Talwar, "Semi-supervised knowledge transfer for deep learning from private training data," *arXiv preprint arXiv:1610.05755*, 2016.

[48] T. Pöppelmann, M. Naehrig, A. Putnam, and A. Macias, "Accelerating homomorphic evaluation on reconfigurable hardware," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015, pp. 143–163.

[49] 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*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1295–1309. [Online]. Available: https://doi.org/10.1145/3373376.3378523

[50] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. ACM, 2018, pp. 707–721.

[51] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "DeepSecure: Scalable provably-secure deep learning," *arXiv preprint arXiv:1705.08963*, 2017.

[52] S. S. Roy, F. Turan, K. Jarvinen, F. Vercauteren, and I. Verbauwhede, "FPGA-based high-performance parallel architecture for homomorphic computing on encrypted data," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 387–398.

[53] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[54] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, "Efficient privacy-preserving face recognition," in *International Conference on Information Security and Cryptology*. Springer, 2009, pp. 229–244.

[55] A. Sanyal, M. J. Kusner, A. Gascón, and V. Kanade, "TAPAS: Tricks to accelerate (encrypted) prediction as a service," *arXiv preprint arXiv:1806.03461*, 2018.

[56] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[57] "Simple Encrypted Arithmetic Library (release 2.3.1)," http://sealcrypto.org, Jun. 2018, microsoft Research, Redmond, WA.

[58] A. Shilov, "TSMC's 5nm EUV Making Progress: PDK, DRM, EDA Tools, 3rd Party IP Ready." [Online]. Available: https://www.anandtech.com/show/14175/tsmcs-5nm-euv-process-technology-pdk-drm-eda-tools-3rd-party-ip-ready

[59] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[60] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," *Designs, codes and cryptography*, vol. 71, no. 1, pp. 57–81, 2014.

[61] tom's Hardware, "Nvidia geforce rtx 3090 founders edition review: Heir to the titan throne," 2020. [Online]. Available: https://www.tomshardware.com/news/nvidia-geforce-rtx-3090-review

[62] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," *arXiv preprint arXiv:1806.03287*, 2018.

[63] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs." in *USENIX Security Symposium*, 2016, pp. 601–618.

[64] V. Wang, "TSMC Fights back Intel in Terms of 16nm FinFET." [Online]. Available: https://en.ctimes.com.tw/DispNews.asp?O=HJY1GE204OGSAA00NK

[65] WikiChip, “Mask / reticle,” 2020. [Online]. Available: https://en.wikichip.org/wiki/mask

[66] C. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang, “Machine learning at facebook: Understanding inference at the edge,” in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 331–344.

[67] D. Wu and J. Haven, “Using homomorphic encryption for large scale statistical analysis,” Technical Report: cs. stanford. edu/people/dwu4/papers/FHESI Report. pdf, Tech. Rep., 2012.

[68] S. Wu, C. Y. Lin, S. H. Yang, J. J. Liaw, and J. Y. Cheng, “Advancing foundry technology with scaling and innovations,” in *Proceedings of Technical Program - 2014 International Symposium on VLSI Technology, Systems and Application (VLSI-TSA)*, April 2014, pp. 1–3.

[69] A. C.-C. Yao, “How to generate and exchange secrets,” in *Foundations of Computer Science, 1986., 27th Annual Symposium on*. IEEE, 1986, pp. 162–167.

[70] G. Yeap, S. Lin, Y. Chen, H. Shang, P. Wang, H. Lin, Y. Peng, J. Sheu, M. Wang, X. Chen, B. Yang, C. Lin, F. Yang, Y. Leung, D. Lin, C. Chen, K. Yu, D. Chen, C. Chang, H. Chen, P. Hung, C. Hou, Y. Cheng, J. Chang, L. Yuan, C. Lin, C. Chen, Y. Yeo, M. Tsai, H. Lin, C. Chui, K. Huang, W. Chang, H. Lin, K. Chen, R. Chen, S. Sun, Q. Fu, H. Yang, H. Chiang, C. Yeh, T. Lee, C. Wang, S. Shue, C. Wu, R. Lu, W. Lin, J. Wu, F. Lai, Y. Wu, B. Tien, Y. Huang, L. Lu, J. He, Y. Ku, J. Lin, M. Cao, T. Chang, and S. Jang, “5nm CMOS production technology platform featuring full-fledged EUV, and high mobility channel FinFETs with densest 0.021mm^2 SRAM cells for mobile SoC and high performance computing applications,” in *2019 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2019.

[71] S. Zahur, M. Rosulek, and D. Evans, “Two halves make a whole,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 220–250.

[72] J. Zhang, Z. Gu, J. Jang, H. Wu, M. P. Stoecklin, H. Huang, and I. Molloy, “Protecting intellectual property of deep neural networks with watermarking,” in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. ACM, 2018, pp. 159–172.

[73] Y. Zhu, G.-Y. Wei, and D. Brooks, “Cloud no longer a silver bullet, edge to the rescue,” *arXiv preprint arXiv:1802.05943*, 2018.