SPOT: Structure Patching and Overlap Tweaking for Effective Pipelining in Privacy-Preserving MLaaS with Tiny Clients

Xiangrui Xu
Department of Computer Science
Old Dominion University
Norfolk, USA
xxu002@odu.edu

Qiao Zhang
Department of Computer Science
Chongqing University
Chongqing, China
qiaozhang@cqu.edu.cn

Rui Ning
Department of Computer Science
Old Dominion University
Norfolk, USA
rning@cs.odu.edu

Chunsheng Xin
Department of Electrical & Computer Engineering
Old Dominion University
Norfolk, USA
cxin@odu.edu

Hongyi Wu
Department of Electrical & Computer Engineering
University of Arizona
Tucson, USA
mhwu@arizona.edu

***Abstract*—Machine Learning as a Service (MLaaS) has paved the way for numerous applications for resource-limited clients, such as IoT/mobile users. However, it raises a great challenge for privacy, including both the data privacy of clients and model privacy of the server. While there have been extensive studies on privacy-preserving MLaaS, a direct adoption of current frameworks leads to intractable efficiency bottleneck for MLaaS with resource constrained clients. In this paper, we focus on MLaaS with resource constrained clients and propose a novel privacy-preserving framework called *SPOT* to address a unique challenge, the memory constraint of such clients, such as IoT/mobile devices, which results in significant computation stalls at the server in privacy-preserving MLaaS. We develop 1) a novel *structure patching* scheme to enable independent computations for sequential inputs at the server to eliminate the computation stall, and 2) a *patch overlap tweaking* scheme to minimize overlapped data between adjacent patches and thus enable more efficient computation with flexible cryptographic parameters. SPOT demonstrates significant improvement on computation efficiency for MLaaS with IoT/mobile clients. Compared with the state-of-the-art framework for privacy-preserving MLaaS, SPOT achieves up to 2× memory utilization boost and a speedup up to 3× on computation time for modern neural networks such as ResNet and VGG.**

***Index Terms*—Mobile Computing, Privacy-preserving, Machine Learning as a Service, Structure Patching, Homomorphic Encryption.**

I. Introduction

The prevalence and widespread adoption of Deep Learning (DL) techniques are evident in various domains, e.g., telehealth [1], [2], where patients can conveniently upload their pathology images for diagnosis. However, designing and training deep neural network models usually require substantial expertise in DL and significant data and computational resources, posing technical barriers for most end-users. To address such challenges, cloud providers have introduced MLaaS [3], as illustrated in Fig. 1, where a proprietary DL model is well-trained and hosted on the cloud, and clients only need to submit queries (i.e., inference requests) to the cloud and receive the inference results (i.e., model outputs) through a web portal.

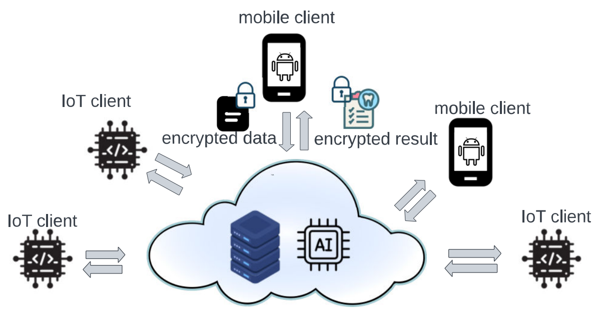


Fig. 1. System architecture of MLaaS with mobile clients.

While MLaaS is a valuable tool for efficiency and productivity, privacy has emerged as a fundamental concern for both clients and servers. From the clients' standpoint, there is an urgent need to safeguard their sensitive information, such as a patient's medical records, against unauthorized access by any entity, including the server. On the other hand, servers strive to prevent the disclosure of proprietary model parameters, developed through significant investments, to clients. Legal frameworks such as the General Data Protection Regulation (GDPR) in the European Union and the Personal Data Protection Act (PDPA) in Singapore mandate the protection of data from unauthorized disclosure. The Health Insurance Portability and Accountability Act (HIPAA) specifically removes sensitive information to protect clients' privacy. While these efforts are instrumental for privacy protection, they may potentially sacrifice valuable information, thus degrading performance. Moreover, recent studies have show that even under the protection of these regulations, attackers may still infer privacy-sensitive data by exploiting available plaintext information [4]. Consequently, there is a pressing requirement to establish secure mechanisms that guarantee the confidentiality of both client's data and server's model parameters in MLaaS.

At the same time, the ubiquity of embedded devices and smartphones makes them ideal devices for end-users in MLaaS. For example, there was over a 101 billion USD market in 2022 and is expected to hit around USD 178.33 billion by 2032 for the embedded system market [5]. A 2020 survey by Doximity also showed that 45% of patients use mobile phones for telehealth services [6]. Various MLaaS applications are often involved with IoT clients (e.g., wearable devices), such as surveillance object detection [7], gesture recognition [8], user verification [9], human activity monitoring [10], and medical health monitoring [11]. To this end, we study MLaaS with IoT/mobile clients in this paper and aim to address the unique challenges raised by IoT/mobile clients in privacy-preserving MLaaS.

A. Our Contributions

To tackle the challenges posed by mobile clients in the context of privacy-preserving MLaaS, we introduce an innovative scheme called SPOT (Structure Patching and Overlap Tweaking). This approach focuses on optimizing the convolution computation, which is widely adopted as a key module in a wide range of modern deep learning models. It not only addresses the challenges of linear computation stall arising from the memory constraints but also enables the utilization of smaller-parameter HE operations tailored for small-footprint clients, as in Sec. II-F. This, in turn, leads to a reduction in computing time under a guaranteed security level.

As illustrated in Fig. 2, SPOT splices the input of linear functions into a number of patches, each of which consists of a portion of all channels. It can adapt the patch size (i.e., length and width) to fit the slot capacity of the HE ciphertext. This is in a sharp contrast to the traditional channel-wise packing in current frameworks, where an entire channel or multiple channels must be packed into a ciphertext. The benefit of our design lies in two folds. First, the convolution-independent nature between the patches enables parallel pipelining using multi-threading, thus mitigating the problem of linear computation stall and achieving high computation efficiency. Second, the ability of flexibly adapting patch size enables SPOT to choose smaller-parameter HE to calculate the linear function, thus further reducing the computation time.

While the proposed patching scheme is promising, it remains nontrivial to implement for achieving optimal performance. First, the computation procedures (such as convolution) must be revamped to effectively leverage the patches to obtain correct results. This will be further discussed in Sec. III-A. Second, the patches must overlap with each other

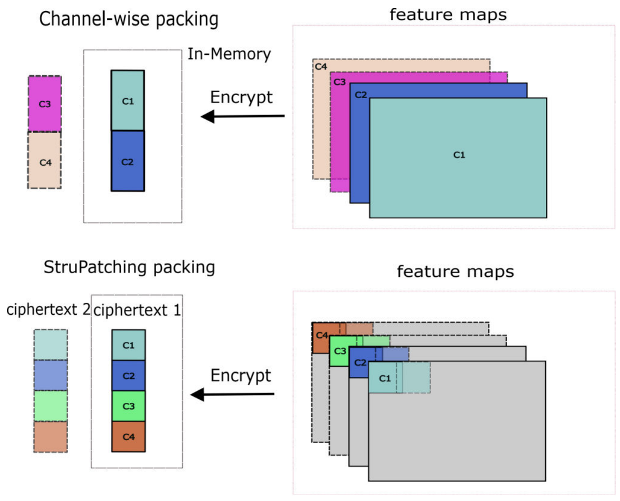


Fig. 2. Channel-wise HE packing versus structure patching based HE packing.

to perform the correct convolution computation, resulting in additional overhead. To mitigate such a problem, we propose tweaking the patch overlaps by carefully crafting a small number of auxiliary ciphertexts that encrypt overlapped data between patches, aiming to reduce the overlap between patches and decrease the cryptographic parameters used for ciphertexts, thus improving the efficiency of the involved HE computation. The details will be discussed in Sec. III-B

We implement SPOT based on the SEAL library for convolution computation and `SCI-Nonlinear` module from CrypTFlow2 for the non-linear computation such as ReLU. We conduct extensive experiments on different small-footprint devices over various neural networks such as ResNet and VGG. SPOT demonstrates up to $3\times$ speedup in inference time and $2\times$ improvements in memory utilization compared to state-of-the-art frameworks.

The rest of the paper is organized as follows. Section II introduces the system framework and cryptographic tools adopted in SPOT. The details of SPOT are elaborated in Section III. The experimental results are illustrated in Section IV. Finally, Section V concludes the paper.

II. Background and Related Works

Notation. We use C_i and C_o to denote the number of input and output channels, respectively. C_n is the number of channels packed in a single ciphertext. $\langle A \rangle_P$ denotes the additive secret share of message A for party $P \in \{0, 1\}$. $[A]_c$ is the ciphertext of A. S' is the number of slots in a ciphertext under given cryptographic parameters. D is the polynomial modulus degree of HE.

A. System Framework

We focus on the MLaaS system as depicted in Fig. 1. Specifically, the client $\mathcal{C}$ possesses sensitive data, such as pathology images of a patient, while the server $\mathcal{S}$ holds a well-trained DL model and outputs the model prediction given the client's input. As discussed in Section I, privacy concerns arise during the interaction between $\mathcal{C}$ and $\mathcal{S}$. Specifically, the client attempts to prevent any third party, including the server, from accessing its private data, while the server is unwilling to disclose its proprietary model parameters, such as weights and kernels, to the client. As such, privacy-preserving MLaaS aims to ensure that the client's input remains fully protected from the server, while the server's model parameters keep completely concealed from the client. Although the computing efficiency acts as a bottleneck for practical applications of privacy-preserving MLaaS and a series of works have made encouraging progress for efficiency enhancement with powerful client, we have observed two key challenges namely memory constraints and cryptographic parameter selection for mobile clients, as pointed out in Section II-F. In this paper, we aim to address these challenges through the SPOT framework.

As for the proprietary model at $\mathcal{S}$, we concentrate on Convolutional Neural Networks (CNNs) that have exhibited remarkable performance in various deep learning tasks [12]. Generally, a CNN includes a stack of layers to capture intricate properties of the input data, such as the spatial relationships among pixels within an image. A layer always contains linear and non-linear functions. The linear functions include dot product and convolution, while the non-linear ones contain activation functions such as the Rectified Linear Unit (ReLU) and pooling (e.g., max pooling and mean pooling). Since the computation overhead for linear functions dominates the overall cost for privacy-preserving MLaaS with mobile clients, as discussed in Section II-F, our proposed SPOT addresses the efficiency optimization for linear computation in the inference process.

B. Packed Homomorphic Encryption

The Homomorphic Encryption (HE) is a class of cryptographic primitives that allow linear computations on encrypted data without decryption, and is primarily used to compute the linear functions in privacy-preserving MLaaS [13]–[15]. Modern HE techniques [16], [17] are able to pack a vector of values into one ciphertext, and perform HE operations in a Single-Instruction-Multiple-Data (SIMD) manner [18] to amortize operation cost. In this work, we adopt the SIMD-style BFV scheme [16] to compute linear functions in CNNs. The main HE operations include HE Multiplication (Mult), HE Addition (Add), and HE Rotation (Rot). Mult performs multiplication between a ciphertext $[x]_c$ and plaintext y, and produces the ciphertext $[x \odot y]_c = [x]_c \odot y$ where $x = \{x_0, x_1, \cdots, x_{S'-1}\}$, $y = \{y_0, y_1, \cdots, y_{S'-1}\}$, and $\odot$ is the element-wise multiplication between encrypted/plaintext vectors. Add conducts the summation between a ciphertext $[x]_c$ and plaintext y (or ciphertext $[y]_c$), and outputs $[x+y]_c = [x]_c + y$ (or $[x+y]_c = [x]_c + [y]_c$) where $+$ is the element-wise addition between encrypted/plaintext vectors. Rot does cyclic rotation of l positions over ciphertext $[x]_c$ and yields $[\tilde{x}]_c$ where $\tilde{x} = \{x_l, \cdots, x_{S'-1}, x_0, x_{l-1}\}$. The efficiency of HE operations depends on pre-determined cryptographic parameters such as S' and smaller cryptographic parameters enable faster HE operations [19], [20]. Our proposed SPOT features with a flexible adoption of small cryptographic parameters under guaranteed security level to accelerate the linear computation in CNNs.

C. Additive Secret Sharing

Given an original message m at party $P \in \{0, 1\}$, one of the two Additive Secret Shares (ASS) is constructed by uniformly sampling randomness r and setting $\langle m \rangle_P = r$, while the other share is formed as $\langle m \rangle_{1-P} = m - r$. To reconstruct the message, one can simply add two shares $m = \langle m \rangle_P + \langle m \rangle_{1-P}$. In this work, we utilize ASS to share the encrypted output of linear functions within CNNs to enable subsequent OT-based computation for non-linear functions.

D. Threat Model

Similar to the previous works such as CrypTFlow2 [15], GAZELLE [13], GALA [21] and Cheetah [22], SPOT follows the *two-party semi-honest* threat model. To be more precise, the client C and the server S follow the protocol but attempt to infer each other's input, namely the client's input data and the server's model parameters, during the inference process. Our protocol, like CrypTFlow2, demonstrates the security of network framework based on the cryptographic tool of ideal/real security, in which the semantic security of PHE and secret sharing scheme.

E. Deep Learning with Tiny Devices

On-device deep learning aims to enable efficient inference/training process given compact models at IoT/mobile client [23]–[26]. Although it deals with model computation under client's resource limitation, straightforward application to privacy-preserving scenarios encounters fundamental problems. On the one hand, releasing the model to IoT/mobile client invades server's data privacy for model parameters. On the other hand, there is no direct solutions to tackle the two unique challenges for privacy-preserving MLaaS with IoT/mobile client, as in Section II-F. By structure patching and overlap tweaking towards the computation process of linear and non-linear functions in neural models, the proposed SPOT enables efficient privacy-preserving inference with IoT/mobile clients. It is worth mentioning that a Channel-By-Channel Packing approach was introduced in [27]. It focuses on improving the throughput and amortizing the expensive key-switching of batch inference by packing multiple images of same channel into single ciphertext in fully homomorphic encryption scheme where the client is not involved in computation. It is based on different design compared with SPOT. It does not address the computation stall issue caused by output ciphertext dependency, which is the bottleneck in our IoT device-based clients scenario. In addition, it does not consider the speedup benefit of smaller cryptographic parameter while SPOT can further improve its efficiency by more flexible parameter selection.

TABLE I
MICROBENCHMARKS RUNTIME FOR CONVOLUTION LAYERS OF DIFFERENT INPUT DATA SIZES ($w \times h$) AND NUMBER OF INPUT/OUTPUT FEATURE MAPS (C_i, C_o). THE NUMBERS HIGHLIGHTED IN RED INDICATE THE PERCENTAGE INCREASE OF THE RUNTIME BY THE MOBILE CLIENT COMPARED TO THE DESKTOP COUNTERPART.

Conv size ($w\|h\|C_i\|C_o$)	Desktop client	Mobile client		
		3 ciphertext	2 ciphertext	1 ciphertext
56|56|64|256	3.405s	5.141s(51.08%↑)	6.01s(76.5%↑)	7.797s(128.98%↑)
28|28|128|512	7.243s	8.167s(12.75%↑)	8.503s(17.39%↑)	10.073s(39.07%↑)
14|14|256|1024	21.814s	22.234s(1.92%↑)	22.42s(2.77%↑)	23.07s(5.75%↑)
7|7|512|2048	73.245s	73.476s(0.31%↑)	73.497s(0.34%↑)	73.9s(0.89%↑)

F. Challenges in Privacy-Preserving MLaaS with Tiny Clients

To address the privacy concerns of the client and protect the ML models of the server, several privacy-preserving MLaaS frameworks have been proposed [14], [21], [28]–[30] to employ cryptographic primitives such as Homomorphic Encryption [16], [17], Garbled Circuits (GC) [31], Oblivious Transfer (OT) [32], and Secret Sharing (SS) [33], in the computation process of DL models, so that the client data is encrypted and the server conducts computation in the cryptographic domain. Among these cryptographic primitives, HE is widely used for linear functions as it inherently supports linear computation [16], [34], while SS and OT are commonly used to compute nonlinear functions [28].

Observation 1: Memory Constraint of Mobile/IoT Clients. Despite significant progress in improving the computation efficiency of privacy-preserving MLaaS, the existing secure inference protocols assume that the client possesses sufficient computation power. Our preliminary experiments show that the computation efficiency drops significantly if the client is an IoT/mobile device with limited memory capacity. Specifically, we performed an evaluation of the convolution layer, which is a primary building block of many modern DL models, by using a desktop client versus a mobile client (Nexus 6), and their CPUs' clock speed are comparable 3.2GHz (AMD EPYC 7413) and 2.7GHz (Snapdragon 805), respectively. However, the execution time of the mobile client is approximately twice as long as that of the desktop counterpart, as shown in Table I that compares the running time for convolution layers with different sizes in ResNet.

Our careful analysis reveals that, given the memory constraint, a mobile client can only hold a limited number of ciphertexts. For example, the average memory budget of a typical Android device like Nexus 6 can be up to 100MB for each running application. However, the size of HE public and secret keys can be substantial, occupying approximately 80.23MB of memory. A ciphertext encrypted in polynomial form is often around 0.7-1.5MB. Considering other necessary memory consumption (usually around 10MB or more), the device can only carry one ciphertext at one time, depending on the fluctuation of other system memory usage. Notwithstanding the IoT devices usually do not have strict memory limits on each application, most IoT devices have much less memory available than mobile phones, which is about 1-2MB of SRAM for holding at most one ciphertext at one time. Fig. 3 illustrates

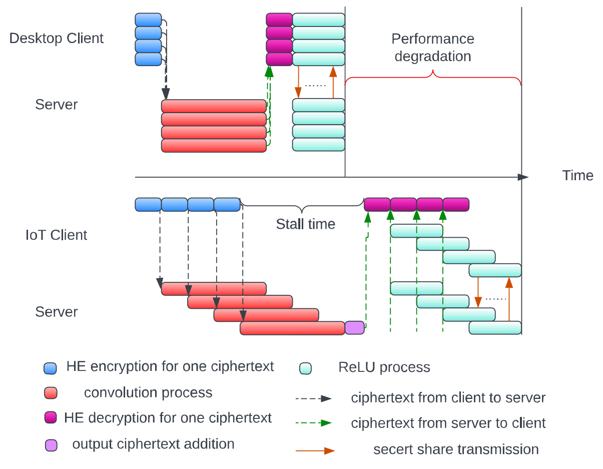

Desktop Client
Server
Performance degradation
Time
IoT Client
Stall time
Server
HE encryption for one ciphertext
convolution process
HE decryption for one ciphertext
output ciphertext addition
ReLU process
ciphertext from client to server
ciphertext from server to client
secert share transmission

Fig. 3. Channel-wise packing on desktop client versus mobile client.

TABLE II
TOTAL EXECUTION TIME ON RESNET-50 ON DESKTOP CLIENT VERSUS IOT CLIENT(DESKTOP CLIENT STATS ARE FROM [22]).

Model	Desktop Client		IoT Client	
	CrypTFlow2	Cheetah (Speedup)	CrypTFlow2	Cheetah (Speedup)
ResNet50	295.7s	80.3s (260%)	428.2s	348.2s (20%)

the impact of memory constraints based on the state-of-the-art frameworks such as CrypTFlow2 [15] and Cheetah [22]. The desktop client, with its abundant memory resources, can generate all input ciphertexts instantly and perform involved computation by leveraging multi-threading [23], [35]. This allows for seamless execution of subsequent Convolution and ReLU operations with minimal additional delay. Similarly, the corresponding decryption for several ciphertexts can be processed by the desktop client at once. In contrast, the mobile client, which is limited in terms of memory, has to generate each input ciphertext sequentially, thus limiting the efficiency of multi-threading since all of the ciphertexts are needed to compute the linear function (e.g., convolution). We name this problem as *linear computation stall*, which results in a significant delay. Although a series of works [15], [22] have been introduced to improve the execution speed, they focus on powerful clients and their the speedup is significantly decreased for a memory-constrained client. As shown in Table II, the speedup of Cheetah over CrypTFlow2 (two recent approaches that are often considered as the state-of-the-art in the literature) is reduced from 260% to only 20% when switching from a desktop client to an IoT device.

Observation 2: Impact of Cryptographic Parameters Selection. We break down the computation time into three parts, shown in Table III: 1) client-HE operations, namely encryption and decryption at the client; 2) server-HE operations, namely HE addition, HE Multiplication, and HE Rotation at the server; and 3) nonlinear or ReLU operations, namely the OT-based

ReLU computation. The results are based on the assumption that the IoT/mobile client holds only one ciphertext. As we can see, the HE operations dominate the computation cost for the whole layer. To accelerate the overall performance, it is key to optimize the efficiency of HE operations.

An important feature of HE operations is that smaller cryptographic parameters (e.g., a smaller number of slots in a ciphertext) enable faster computation [19]. For example, given the 128-bit security level, the cost of one HE operation using the CKKS scheme (to be discussed in Sec II-B) with 4096 slots can be 2 to 4 times faster than that with 8192 slots [16], [19]. Such speedup is particularly valuable for IoT/mobile clients given its limited computation power. However, the state-of-the-art approaches such as CrypTFlow2 [15] do not have much flexibility on cryptographic parameters selection, e.g., the number of slots for the ciphertext cannot be smaller than the size of one input channel, as their design packs one or more channels into a ciphertext to optimize performance.

As the size of one input channel can be very large in various practical applications, the state-of-the-art frameworks have to choose large cryptographic parameters to guarantee a desired security level. For instance, CrypTFlow2 sets its number of slots in BFV no smaller than 8192 [15]. Such constraint on cryptographic parameter selection significantly limits the flexibility of current frameworks. This limitation hinders the ability to efficiently decrease the computation time of MLaaS by incorporating smaller-parameter HE operations.

TABLE III
A BREAKDOWN OF COMPUTATION TIME OF THE CONVOLUTION LAYER INTO THE THREE COMPONENTS OF MLAAS, TOGETHER WITH THE PERCENTAGE AMONG THE TOTAL EXECUTION TIME. THE MOBILE CLIENT MEMORY IS ASSUMED TO BE ABLE TO ACCOMMODATE ONE CIPHERTEXT.

Conv size($w\|h\|C_i\|C_o$)	client-HE	server-HE	ReLU
56|56|64|256	5.376s(61%)	3.03s(34%)	0.29s(3%)
28|28|128|512	2.688s(27%)	6.86s(69%)	0.34s(3%)
14|14|256|1024	1.344s(6%)	21.06s(93%)	0.24s(1%)
7|7|512|2048	0.672s(1%)	72.1s(98%)	0.18s(1%)

III. PROPOSED SPOT SCHEME

In this section, we present the SPOT framework which supports secure inference with resource-constrained clients. First, we introduce the channel-wise packing that is adopted in the state-of-the-art privacy-preserving frameworks. Specifically, C_n out of C_i input feature maps are packed into one ciphertext where $C_n = \lfloor S'/HW \rfloor$, S' is the number of slots in a ciphertext, and H and W are the height and width of a feature map, respectively. Fig. 2 shows an example of channel-wise packing with $C_i = 4$ and $C_n = 2$.

Channel-wise packing needs to pack each entire input feature map into a ciphertext, which requires a large S' for a large input feature map, and thus makes it infeasible for IoT/mobile clients with limited resources. The direct non-overlap patching divides the original feature maps into a series of patches, each with size $H' \times W' \times C_i$ where usually $H' \ll H$ and $W' \ll W$, as shown in Fig. 2. Then each patch is encrypted into one ciphertext. However, as patches must overlap with each other to make sure the following convolution with respect to all patches is equivalent to the result of the original convolution, such vanilla patching leads to a fixed overlap size towards the kernel dimensions, which disables the patching-based packing in layers with large C_i to adopt faster HE with small cryptographic parameters. To make patching-based packing truly advantageous, we first propose in Section III-A a structure-patching-based pipelining that packs N patches into one ciphertext such that $H'W'C_iN \approx S'$ by adapting H', W', and N. Then the overlap tweaking is designed in Section III-B to minimize the overlap size and enable efficient computation with small cryptographic parameters in layers with large C_i.

A. Structure Patching Pipelining

Recall that we focus on privacy-preserving MLaaS with IoT/mobile clients where the linear and non-linear functions in CNN models are computed by packed HE and OT, respectively. Based on the adaptive packing for patches, we intend to compute the convolution efficiently by first performing the proposed structure patching pipelining. For a lucid description, we first present mainstream HE-based convolution based on channel-wise packing namely Single Input and Single Output (SISO), and Multiple Input Multiple Output (MIMO). Direct application of SISO and MIMO to MLaaS with IoT/mobile clients causes linear computation stall as discussed in Section II-F. As such, we propose the structure-patching-based computation to address this challenge.

SISO: SISO computes the convolution with $C_i = C_o = 1$. Specifically, given a kernel with size $k_H \times k_W$, each value of the convolution output is the weighted sum of elements in the input feature map that are within the kernel window. For example, the first value of the convolution output in Fig. 4 is obtained by placing the central element of kernel K namely F5 at the first number of input feature map X namely M1, and the resultant value is the sum of correspondingly multiplied numbers in X within K's window namely (F5M1+F6M2+F8M4+F9M5), so on and so forth. Therefore, the value of the convolution output is the sum of at most $k_H k_W$ ambient numbers of input feature map which are weighted by $k_H k_W$ elements in kernel.

As such, we get the convolution output of SISO by rotating $[X]_c$ multiple times, to produce $k_H k_W$ ciphertexts such that the $k_H k_W$ ambient numbers from $[X]_c$, which correspond to the i-th value of the convolution output, are able to appear at the i-th location of those $k_H k_W$ rotated ciphertexts, as shown in Fig. 4. By properly assigning the to-be-multiplied kernel values for the number at the i-th location in each of those $k_H k_W$ ciphertexts, which results in $k_H k_W$ kernel plaintexts, the convolution output is obtained by multiplying each of $k_H k_W$ pairs of rotated ciphertext and kernel plaintext, and then summing these $k_H k_W$ multiplied ciphertexts. For example, the fifth value of the convolution output in Fig. 4 is the sum of F1M1, ..., F9M9. In order to get this sum at the fifth location

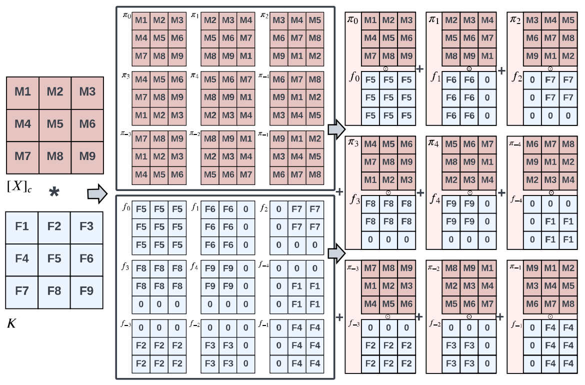


Fig. 4. Computation of SISO.

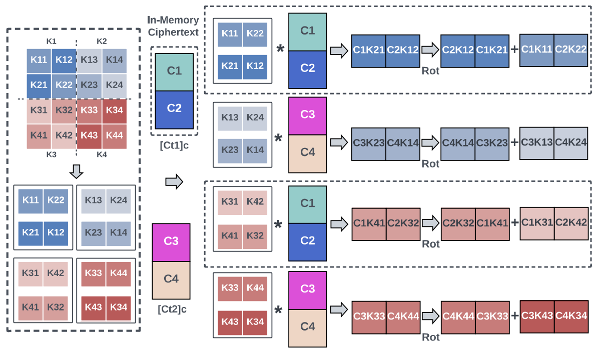


Fig. 5. Computation of MIMO.

in $[X]_c$, $[X]_c$ is rotated to form $k_H k_W = 9$ ciphertexts, each of which makes one of nine values in the kernel, namely M1 to M9, located at the fifth location in X. By multiplying those nine rotated ciphertexts with nine value-assigned kernel plaintexts and summing up the multiplied ciphertexts, the fifth value of the convolution output is obtained. Since each rotation works on all numbers of $[X]_c$, the i-th ($i \neq 5$) value of convolution output is obtained simultaneously in that resulted ciphertext.

MIMO: MIMO deals with more general convolution where C_i and C_o are larger than one. In such case, the kernel is with size $k_H \times k_W \times C_i \times C_o$, and each of the C_o convolution outputs is the sum of C_i SISO convolution. MIMO first packs C_n out of the C_i feature maps in one ciphertext which produces $\lceil C_i/C_n \rceil$ ciphertexts in total. For each of those $\lceil C_i/C_n \rceil$ ciphertexts, it produces partial SISO for all of the C_o convolution output and the final convolution is obtained by adding SISO corresponding to the same output channels. The computation is described as follows. 1) For the i-th $(1 \leq i \leq (\lceil C_i/C_n \rceil))$ input ciphertext, it packs $(i-1)C_n$-th to $(iC_n - 1)$-th feature maps. 2) For each group of C_n convolution output, C_n SISO ciphertexts are produced between i-th input ciphertext and each of C_n diagonally-formed kernel sets. 3) Such C_n SISO ciphertexts are rotated to make them correspond to the same output channels, and these rotated ciphertexts are added to form partial SISO for a group of C_n output convolution. 4) All partial SISO corresponding to the same output channels are summed up to finally obtain C_n out of C_o convolution.

Fig. 5 shows an example of MIMO with $C_i = C_o = 4$ and $C_n = 2$. As for the ciphertext $[\mathrm{Ct}_1]_c$ containing input feature maps C1 and C2, it first produces $C_n = 2$ SISO ciphertexts with diagonally-formed kernel sets $\{\mathrm{K11}, \mathrm{K22}\}$ and $\{\mathrm{K21}, \mathrm{K12}\}$. Since the ciphertext with respect to $\{\mathrm{K11}, \mathrm{K22}\}$ corresponds to partial SISO for the first and the second convolution output while the other one with respect to $\{\mathrm{K21}, \mathrm{K12}\}$ corresponds to partial SISO for the second and the first convolution output, the latter ciphertext is rotated to make the rotated ciphertext correspond to partial SISO for the first and the second convolution output. After that, the rotated ciphertext is added with the ciphertext with respect to $\{\mathrm{K11}, \mathrm{K22}\}$ to form the partial SISO for the the first and the second convolution output. $[\mathrm{Ct}_1]_c$ similarly forms partial SISO for the third and the fourth convolution output with diagonally-formed kernel sets $\{\mathrm{K31}, \mathrm{K42}\}$ and $\{\mathrm{K41}, \mathrm{K32}\}$. Similar logic is applied to ciphertext $[\mathrm{Ct}_2]_c$ with diagonally-formed kernel sets $\{\mathrm{K13}, \mathrm{K24}\}$ and $\{\mathrm{K23}, \mathrm{K14}\}$, and $\{\mathrm{K33}, \mathrm{K44}\}$ and $\{\mathrm{K43}, \mathrm{K34}\}$. Finally, the partial SISO for the same output channels are added together to get the desired convolution.

Structure Patching: Since MIMO needs to add all SISO ciphertexts for the same output channels to finally obtain the desired result, as described in step 4 above, it poses noticeable stall time to get those SISO ciphertexts with a tiny client because it is not feasible to encrypt all input ciphertexts simultaneously given the memory constraint of tiny clients, as discussed in Section II-F. Such memory constraint forces tiny clients to generate and send $\lceil C_i/C_n \rceil$ input ciphertexts to the server sequentially, as shown in the left part of Fig. 6, and the server thus has to sequentially get needed SISO ciphertexts to obtain the final convolution output, which leads to stall inevitably.

Such stall time is due to the dependency between input ciphertexts to get SISO ciphertexts that correspond to the same output channel. Therefore we are motivated to remove such dependency to reduce the stall time. Having observed the issue of incomplete channels within each ciphertext leading to dependency, we slice the input with size $H \times W \times C_i$ into a series of smaller patches with size $H' \times W' \times C_i$. Since each patch contains values from all input channels, the convolution with a single patch is able to obtain a group of final values in an output channel, while the stall can be eliminated as this operation can be completed within the memory constraint of a mobile client by selecting a sufficiently small patch size $H' \times W' \times C_i$ to fit the memory size. Moreover, the convolutions with different patches are independent from each other.

Next, we first discuss the convolution where $C_n^{'} = C_i$ namely each ciphertext packs one patch. This can be divided

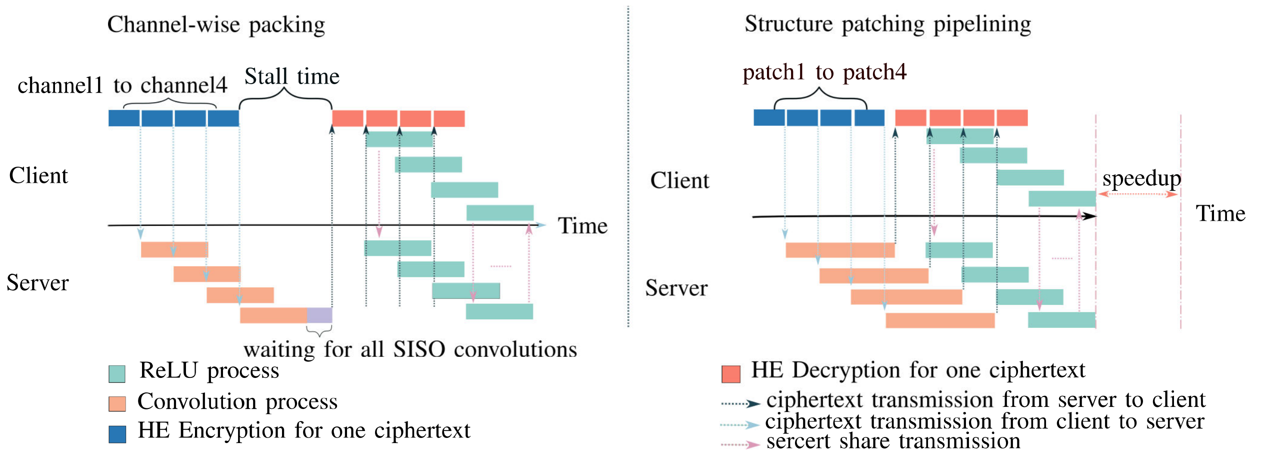
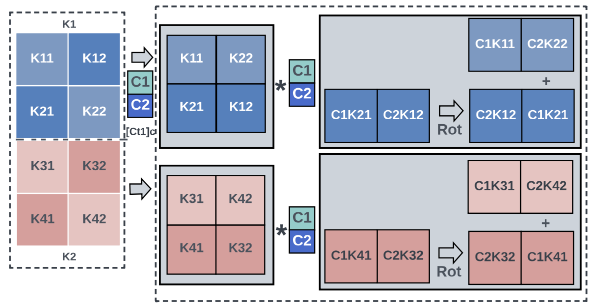
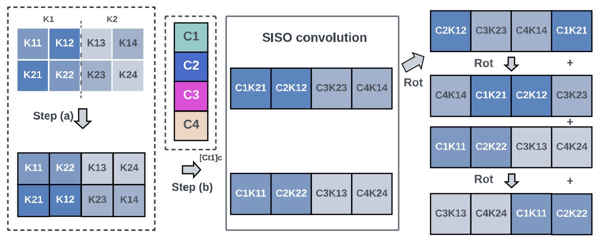

Channel-wise packing
channel1 to channel4
Stall time
Client
Time
Server
waiting for all SISO convolutions
Structure patching pipelining
patch1 to patch4
speedup
ReLU process
Convolution process
HE Encryption for one ciphertext
HE Decryption for one ciphertext
ciphertext transmission from server to client
ciphertext transmission from client to server
sercert share transmission

Fig. 6. Left: channel-wise HE packing with mobile client. Right: Structure patching based HE packing with mobile client. Both cases compute convolution and subsequent ReLU.



(a) Computation in the case $C_o \geq C_i$.



(b) Computation in the case $C_o < C_i$.

Fig. 7. Convolution with structure patching.

into two cases, 1) $C_o \geq C_i$ and 2) $C_o < C_i$. For the case of $C_o \geq C_i$, we revamp the convolution process by dividing the kernels into blocks with a size equal to C_i. We then apply the MIMO logic of convolution between $[\mathrm{Ct}_1]_c$ and kernel sets to obtain the result. The ciphertext $[\mathrm{Ct}_1]_c$, containing all C_i input channels, is the only input ciphertext required to calculate the two output ciphertexts, as shown in Fig. 7 (a). As for the computation with $C_o < C_i$, we design to split the kernels into the blocks with its size equal to C_o. To produce the needed ciphertext, we concatenate all the same direction diagonally-formed kernel sets into one set. After C_o SISO ciphertexts are produced between the input ciphertext and C_o kernel sets, we align and add the rotated SISO ciphertexts to produce the output ciphertext with $C_i - 1$ rotations. An example of the overall process is illustrated in Fig. 7 (b). Note that each row of kernel sets produces one output channel through convolution with the input data, and the number of rows is C_o. The number of elements in a row is equal to C_i, the number of input channels. To fit the convolution with the input ciphertexts, the kernel sets are transformed into a diagonal form and concatenated according to the input channels in input ciphertexts as illustrated in Fig. 7 (b) Step (a). The final convolution result is then obtained through a sequence of SISO Rot for aligning the same row SISO convolutions and Add operations as shown in Step (b).

TABLE IV
CIPHERTEXT SIZE AND OPERATION COST ON DIFFERENT PARAMETER LEVELS.

Parameter level(D)	Ciphertext size (Byte)	Mult cost (s)
16384	789617	0.0015
8192	394865	0.0007
4096	131697	0.00014

In this way, each incoming ciphertext to the server, which represents a patch of all input channels as illustrated in Fig. 2, is eligible to complete the convolution computation to get a group of final values in the output channels, which can be also seen as a ‘patch’ of the output channels, without waiting for other input ciphertexts (i.e., the ciphertexts for other patches) as we use only one input ciphertext to produce output ciphertexts for various kernel blocks. Hence SPOT effectively eradicates the stall time, as demonstrated in the right part of Fig. 6.

Note that we are able to extend above computation for one patch to N patches namely $C_n^{'} = NC_i$. By adapting H', W', and N of the patch size, the slots in each ciphertext can be fully utilized, which contributes to producing fewer ciphertexts and thus reduces computation overhead. Meanwhile, a ciphertext encrypted with smaller cryptographic parameters features faster HE operations. For example, Table IV shows the relationship between different parameter levels and the corresponding cost of BFV in the SEAL library [19]. Here the higher the parameter level is, the larger the associated cryptographic parameters are. Under the 128-bit security level, the HE cost such as Mult with smaller cryptographic parameters is significantly smaller, as listed in Table IV.

Therefore, we are motivated to set smaller cryptographic parameters to enable faster convolution. On the one hand, the channel-wise packing makes it not possible to utilize HE with smaller cryptographic parameters because the input size H and W for practical data often needs large cryptographic parameters such as S'. In contrast, H', W', and N are adjustable in our structure patching pipelining scheme. Thus a smaller S' is possible such that $H'W'NC_i \approx S'$, which further boosts the computation efficiency of the patching-based HE computation. For example, by splitting the input with size 56×56 from ResNet [12] into a series of patches with size 4×4, we are able to reduce the cryptographic parameter S' from $D = 16384$ with $C_n^{'} = 2$ to $D = 2048$ and other corresponding parameters accordingly, which reduces the computation time.

B. Patch Overlap Tweaking

Recall that a convolution is to align the center of a kernel with a certain size, say 3×3, to a specific location in the input feature map, and then perform the corresponding dot-product, followed by a summation. Since each patch contains only part of the input feature map, the convolution corresponding to a boundary location of a patch would run into a problem, as part of the surrounding areas of that location is not in this patch, but in the adjacent patch. We shown an example of the problem in Fig. 8. To generate the convolution result of e, the filter center $K5$ is placed on top of e in patch $[b]_c$ and results in $\{bK_2 + cK_3 + eK_5 + fK_6 + hK_8 + iK_9\}$ in the position of e of the share $\langle b * K - r\rangle_c$, where r denotes the random number share generated by the server. Same for patch $[a]_c$, the convolution result is $\{aK_2 + dK_5 + gK_8\}$ by placing the center of the filter K on top of d. Both convolution results are incorrect since some needed feature map values are missing for a kernel size of 3. It is clear that applying the simple patching scheme does not recover the correct convolution result for the values at the edge of each patch.

In order to get all desired convolution values among all patches, the patches must overlap. For instance, we can set the overlap size, namely the number of overlapped columns/rows between two adjacent patches, to be $\lceil (k_H + s)/2 \rceil$, where s denotes the stride size. In the example shown in Fig. 9, the kernel K has size $k_H = k_W = 3$ with stride $= 1$, which indicates the overlap size of two. In this way, the server performs convolution for each patching-packed ciphertext independently and shares the output with the mobile client, which is able to assemble the received share to get its right share of convolution. Fig. 9 shows an example to get the convolution share with patch overlap. Two adjacent patches,

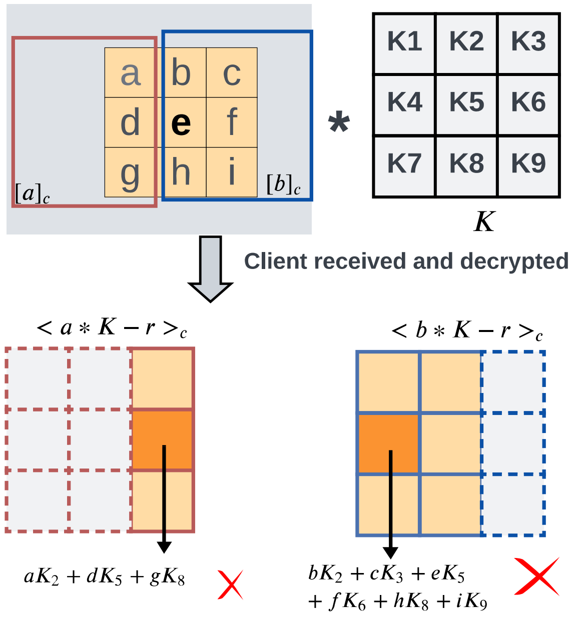


Fig. 8. An example of incorrect results based on non-overlapping patches to compute convolution at e and d with $k_H = k_W = 3$.

$[a]_c$ and $[b]_c$, are encrypted by a mobile client and sent to the server, respectively. The server conducts convolution for each ciphertext and shares the result with the mobile client. Upon receiving the convolution shares of the patches, the mobile client assembles these values to form its final share of convolution by picking out the shares of correct convolution values (e.g., the share at the location of e in $\langle b * K - r\rangle_c$ is chosen as the final share of convolution value at that location rather than the one in $\langle a * K - r\rangle_c$, since the latter is missing some feature map values). Note that there is no need for two patches to overlap when a kernel has a size of 1×1.

To fulfill the minimum overlap size requirement among adjacent patches, the minimum patch size should be larger than the minimum overlap size, otherwise the adjacent patches are coincided and do not cover the whole feature map. For the example aforementioned, the minimum patch size $H' = W'$ should be 3. Meanwhile, we observe the efficiency of smaller cryptographic parameters as shown in Table IV and are motivated to choose the smallest practical cryptographic parameter, which is $D = 4096$, to pack each patch for the computation and memory efficiency. However such combination imposes a conflict between available slot number and the number of entry values of one patch for typical input channel in VGG and ResNet, taking the aforementioned example when $C_i = 512$ and a patch with $H' = W' = 3$ (i.e., $3 * 3 * 512 > 4096$), if we pack all C_i to maintain the pipelining efficiency [12], [16], [36]. Compromising to bigger cryptographic parameters such as $D = 8192$ loses around $7\times$ computation efficiency

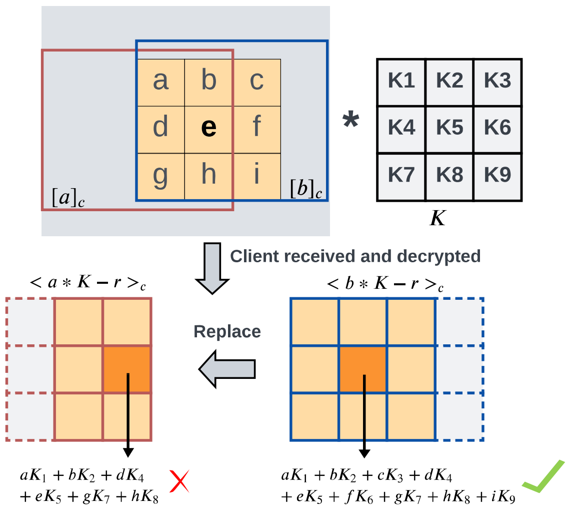

a b c
d e f
g h i
$[a]_c$
$[b]_c$
K1 K2 K3
K4 K5 K6
K7 K8 K9
K
Client received and decrypted
$<a*K-r>_c$
$<b*K-r>_c$
Replace
$aK_1+bK_2+dK_4+eK_5+gK_7+hK_8$
$aK_1+bK_2+cK_3+dK_4+eK_5+fK_6+gK_7+hK_8+iK_9$

Fig. 9. Patch overlapping to compute convolution at e with $k_H = k_W = 3$.

that brings by smaller cryptographic parameters.

To address this challenge, we propose a scheme to minimize the overlap to be as small as one, thus making it possible to fit the patch into a ciphertext with the smallest cryptographic parameters (subject to 128-bit security), to enable faster HE operations. The main idea is to craft auxiliary patches such that the final share of convolution at the mobile client is arithmetically assembled, rather than simply selected, among shares of patch convolution. Under this design, we are able to reduce the overlap size to be one, to get smaller patches. Specifically, Fig. 10 demonstrates such overlap tweaking scheme with a kernel size $k_H = k_W = 3$ which is widely adopted in modern CNN models such as ResNet and VGG [36]. The auxiliary patch C is encrypted by the mobile client as $[C]_c$ and sent to the server along with encrypted patches $[A]_c$ and $[B]_c$. After the mobile client receives shares of $\langle A*K-r\rangle_c$, $\langle B*K-r\rangle_c$, and $\langle C*K-r\rangle_c$ from the server, it gets a share of the desired convolution by summing the corresponding shares from $\langle A*K-r\rangle_c$ and $\langle B*K-r\rangle_c$, and then subtracting the share of $\langle C*K-r\rangle_c$. While an additional ciphertext namely $[C]_c$ is introduced, smaller patches and HE with smaller cryptographic parameters bring more computation efficiency compared with the extra cost. This novel design enables the structure patching in deeper layers of modern CNNs with a large C_i such as ResNet and VGG.

C. Complexity comparison

Table V compares the overall complexity of convolution computations for channel-wise output rotation and patch, where C_m and $C_m^{'}$ denotes the number of input ciphertexts for CrypTFlow2 and SPOT.

IV. Evaluation

A. Experimental Setup

We implement SPOT based on the SEAL library [19] for linear functions such as convolution, and the `SCI-NonLinear` module from CrypTFlow2 [15] for non-linear functions such as ReLU. We test the performance of SPOT with the ImageNet dataset [37] on a series of widely-adopted CNN models such as ResNet-34 [12], ResNet-50 [12], ResNet-101 [12], VGG-11 [36], and VGG-13 [36]. We use Google Nexus 6 and Kinetis K27 microcontroller to serve as mobile and IoT clients, respectively. Nexus 6 is configured with a memory between 64MB and 128MB to run Android applications as well as perform HE operations such as encryption and decryption. The microcontroller is equipped with Cortex-M4 CPU with 1MB SRAM and 2MB flash memory with 80MB SD card ROM. The server runs on Ubuntu and is equipped with an AMD EPYC 7413 24-core Processor 2.65GHz base clock and a 64GB RAM. Similar to current state-of-the-art privacy-preserving frameworks, the 128-bit security level is assumed in our experiments. We select the range of cryptographic parameters for the BFV scheme of the SEAL library [19] subject to this security level constraint, while optimizing the specific parameter values within this range, to enable high slot utilization, and balance the number of ciphertexts and computation overhead. The patch size selection used in experiments corresponding to different cryptographic parameters are shown in Table VIII. In the following, we evaluate SPOT with regard to performance metrics including the mobile/IoT

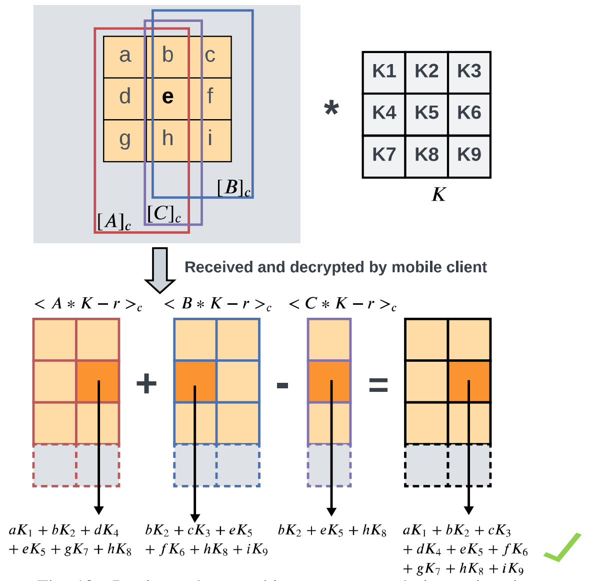

a b c
d e f
g h i
$[A]_c$
$[C]_c$
$[B]_c$
K1 K2 K3
K4 K5 K6
K7 K8 K9
K
Received and decrypted by mobile client
$<A*K-r>_c$
$<B*K-r>_c$
$<C*K-r>_c$
$aK_1+bK_2+dK_4+eK_5+gK_7+hK_8$
$bK_2+cK_3+eK_5+fK_6+hK_8+iK_9$
$bK_2+eK_5+hK_8$
$aK_1+bK_2+cK_3+dK_4+eK_5+fK_6+gK_7+hK_8+iK_9$

Fig. 10. Patch overlap tweaking to get convolution at location e.

TABLE V
Complexity comparison between CrypTFlow2 and SPOT

Method	Permutation	SIMDMulti	Add
CrypTFlow2	$C_m * \frac{C_o}{C_n}(C_n-1)$ $+C_m(K_w*K_h-1)$	$C_m*C_oK_wK_h$	$C_m\frac{C_o}{C_n}(C_nK_wK_h-1)$
SPOT	$C_m^{'}(K_wK_h-1)$ $+C_m^{'}\frac{C_o}{C_i}(C_i-1)$	$Cm^{'}CoK_wK_h$	$C_m^{'}\frac{C_o}{C_i}(C_iK_wK_h-1)$

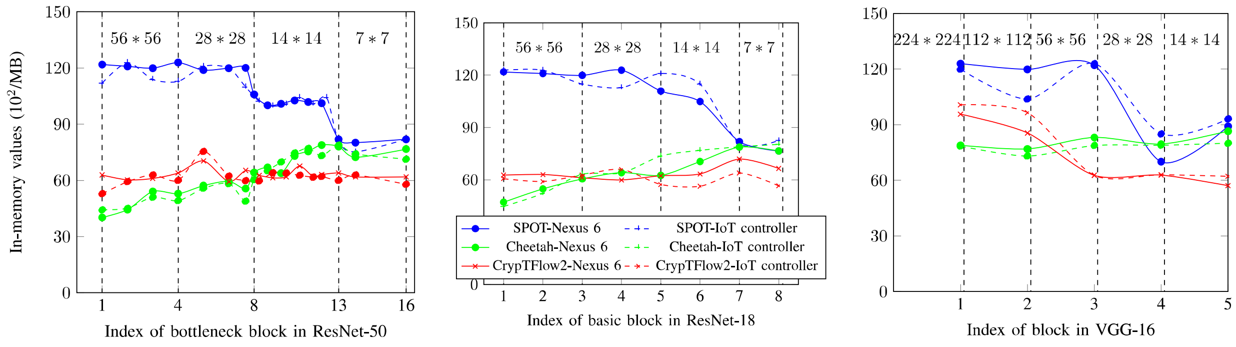


Fig. 11. Memory utilization in various CNN models.

client's memory utilization, the computation cost of the convolution process, and the overall inference time on CNN models, compared with CrypTFlow2 [15] and Cheetah framework [22] in IoT device client settings.

TABLE VI
PATCH SIZE ($H^{'} * W^{'}$) SELECTION FOR DIFFERENT ENCRYPTION PARAMETERS AND CONVOLUTION LAYERS, WHERE $S^{'}$ IS THE CYCLOTOMIC RING DEGREE AND co_mod DENOTES COEFFICIENT MODULUS SIZE(PLAINTEXT MODULUS = 2^{20}).

Network layers	$S^{'} = 4096$	$S^{'} = 8192$	$S^{'} = 16384$
$(W\|H\|Ci\|Co)$	co_mod=109	$co\_mod = 218$	$co\_mod = 438$
56|56|64|64	8*8	16*8	16*16
28|28|128|128	8*4	8*8	16*8
14|14|256|256	4*4	8*4	8*8
7|7|512|512	2*4	4*4	8*4

B. Memory Utilization at Tiny Client

We define the *in-memory value* as the number of feature map entry values that are stored in per megabyte (MB) memory of the mobile/IoT client, since different packing schemes lead to various amounts of unused slots and the number of loaded ciphertexts. *In-memory value* can reflect the amount of valid entry values loaded into the client's memory. A larger in-memory value indicates a higher slot utilization of each ciphertext and a higher flexibility for structure patching to deal with HE-based computation under resource constraints. Fig. 11 compares in-memory values of SPOT with that of CrypTFlow2 and Cheetah over different CNN models, including ResNet-50 with bottleneck blocks [12], ResNet-18 with basic blocks [12], and VGG-16 with five types of blocks [36]. The bottleneck block includes a stack of convolution layers with different input/output channels in kernel sizes 1×1, 3×3, and 1×1. The basic block contains convolution layers with various input/output channels in kernel size 3×3, and the block in VGG-16 has kernels in size 3×3 with different output channels. We can see from Fig. 11 that SPOT significantly improves the memory utilization of the mobile/IoT client. Specifically, SPOT is capable of handling up to $2\times$ more in-memory values in both Nexus 6 and IoT controller compared to CrypTFlow2 and Cheetah. Such capability is attributed to the flexible patch size which maximizes the utilization of slots in each ciphertext for large input feature map size, as well as the overlap tweaking which enables selection of cryptographic parameters with higher HE efficiency. Meanwhile, channel-wise packing wastes more slots for each ciphertext and has to compromise to bigger ciphertext for more slots to pack two or more channels of large feature map. We notice that Cheetah shows similar high slot utilization as SPOT for encrypting input ciphertexts due to the new encoding method. However, the extraction of output ciphertexts generates a large amount of LWE output ciphertexts with only one useful coefficient in each ciphertext, which deteriorates the total slot and memory utilization. Note that while the memory utilization of SPOT is much higher in most of the blocks, the improvement drops in some deeper blocks due to a larger number of input channels, namely C_i, and the arithmetical computation in overlap tweaking. Mobile clients' performance fluctuates affected by the actual memory availability, compared with IoT devices in different blocks.

TABLE VII
RUNNING-TIME MICROBENCHMARK ON BOTTLENECK BLOCKS IN RESNET-50.

Block type	CrypTFlow2		Cheetah		SPOT	
$(W\|H\|Ci\|Co)$	IoT controller	Nexus 6	IoT controller	Neuxs 6	IoT controller	Nexus 6
56|56|64|256	8.356s	7.797s	9.97s	7.92s	3.54s(2.35×)	2.9s(2.69×)
28|28|128|512	9.73s	10.073s	10.52s	9.95	2.24s(4.34×)	2.6s(3.87×)
14|14|256|1024	22.53s	23.07s	20.93s	22.01s	7.45s(2.80×)	8.618s(2.53×)
7|7|512|2048	72.5s	73.9s	70.51s	71.64s	25.73s(2.74×)	26.147s(2.74×)

TABLE VIII
RUNNING-TIME MICROBENCHMARK ON BASIC BLOCKS IN RESNET-18.

Block type	CrypTFlow2		Cheetah		SPOT	
$(W\|H\|Ci\|Co)$	Nexus 6	IoT controller	Nexus 6	IoT controller	Nexus 6	IoT controller
56|56|64|64	1.41s	1.593s	2.89s	2.962s	0.693s(2.03×)	0.784s(2.03×)
28|28|128|128	2.34s	2.063s	4.01s	3.24s	0.878s(2.66×)	0.918s(2.24×)
14|14|256|256	4.45s	4.804s	4.28s	4.53s	1.507s(2.84×)	1.566s(2.90×)
7|7|512|512	22.14s	22.30s	20.7s	21.94s	7.764s(2.67×)	7.636s(2.87×)

C. Running-time Performance On Convolutional Blocks

We then test the running-time performance over various convolutional blocks to demonstrate the computation efficiency of SPOT compared to channel-wise computation in CrypTFlow2 and Cheetah. Tables VII and VIII compare the running time of SPOT, Cheetah, and CrypTFlow2 on various bottleneck blocks and basic blocks in ResNet models, respectively, and illustrates

TABLE IX
RUNTIME MICROBENCHMARK ON BLOCKS IN VGG-16.

Block type ($W\|H\|Ci\|Co$)	CrypTFlow2		Cheetah		SPOT	
	Nexus 6	IoT controller	Nexus 6	IoT controller	Nexus 6	IoT controller
224|224|64|64	30.83s	31.5s	33.9s	36.2s	8.88s(3.47×)	9.056s(3.47×)
112|112|128|128	18.8s	19.27s	19.6s	21.1s	6.39s(2.94×)	6.798s(2.83×)
56|56|256|256	4.21s	4.281s	5.16s	5.96s	2.55s(1.65×)	2.538s(1.68×)
28|28|512|512	3.12s	3.407s	3.82s	4.24s	2.32s(1.38×)	2.614s(1.30×)
14|14|512|512	4.40s	4.55s	3.92s	3.12s	2.13s(2.06×)	2.266s(2.00×)

TABLE X
TOTAL EXECUTION TIME ON RESNET AND VGG.

Network model	CrypTFlow2		Cheetah		SPOT	
	Nexus 6	IoT controller	Nexus 6	IoT controller	Nexus 6	IoT controller
ResNet-101	811.2s	827.6s	721.6s	882.1s	279.7s(2.58×)	307.3s(2.69×)
ResNet-50	428.2s	435.4s	348.2s	356.8s	153.0s(2.27×)	160.8s(2.21×)
ResNet-34	118.3s	112.3s	80.5s	89.5s	49.53s(1.62×)	41.8s(2.14×)
ResNet-18	101.6s	103.71s	83.1s	111.7s	47.78s(1.74×)	49.19s(2.11×)
VGG-11	65.29s	72.13s	65.8s	69.4s	33.29s(1.97×)	25.31s(2.75×)
VGG-16	151.23s	154.5s	163.2s	159.4s	64.54s(2.34×)	75.05s(2.05×)

the speedup of SPOT. Since the blocks with larger feature map contain more entry values and need more output ciphertexts to be extracted, Cheetah shows less runtime performance boost for mobile/IoT clients in starter blocks. Thus, we compare the best running-time performance with SPOT instead of a specific method. Overall, SPOT achieves up to 4× and 3× speedup compared to CrypTFlow2 and Cheetah. With structure patching and overlap tweaking, SPOT is able to efficiently carry out HE computation under limited resources at mobile/IoT clients, by splitting the input into a series of patches, and minimizing the overlap between two patches to enable small cryptographic parameter selection for faster HE operations. For example, SPOT demonstrates 4× speedup on IoT controller in a bottleneck block with an input size 28×28, and there is nearly 3× speedup in a basic block with the number of input channels $C_i = 512$. At last, a significant speedup of SPOT over CrypTFlow2 and Cheetah is also observed in Table IX for blocks in VGG-16.

D. End-to-End Performance on CNNs

We finally evaluate the total execution time on an entire CNN model for SPOT. As shown in Table X, SPOT achieves a speedup of up to $2.5 \sim 3\times$ for the ResNet series, and a speedup of $2.7 \sim 2.8\times$ for the VGG series, compared with Cheetah and CrypTFlow2, respectively. This speedup is consistent with the ones reported for the various individual blocks in the previous subsection. This running time improvement demonstrates the efficiency of the novel design of structure patching and patch overlap tweaking, which work together to significantly reduce the computation time of privacy-preserving MLaaS with memory-limited mobile clients. Even though Cheetah shows large acceleration for desktop clients by avoiding rotations, it still faces linear computation stall problem due to ciphertext dependency. Moreover, it extracts each useful polynomial coefficient into a ciphertext, thus increasing the number of ciphertexts and corresponding processing time. These two bottlenecks prolong the total execution process, making Cheetah's performance improvement negligible compared to CrypTFlow2 in the tiny client setting.

V. CONCLUSION

This paper has introduced SPOT, a novel framework for machine learning as a service (MLaaS) with resource-constrained clients. SPOT features a novel design of structure patching and patch overlap tweaking to resolve the problems of computation stall at the server and inflexible cryptographic parameters selection that are faced by the current state-of-the-art privacy-preserving MLaaS frameworks. SPOT has demonstrated up to 2× higher memory utilization at the clients, and an overall speedup of up to 3× on modern CNN models such as ResNet and VGG.

VI. ACKNOWLEDGEMENT

This work was supported in part by the National Science Foundation under Grant OAC-2320999, CNS-2120279, IIS-2236578, CNS-2153358, and DUE-2153358, DoD Center of Excellence in AI and Machine Learning (CoE-AIML) under Contract Number W911NF-20-2-0277, the National Natural Science Foundation of China under Grant 62302067, the Natural Science Foundation of Chongqing, China under Grant CSTB2022NSCQ-MSX1217, and the Commonwealth Cyber Initiative.

REFERENCES

[1] J.-D. Lin, H.-H. Lin, J. Dy, J.-C. Chen, M. Tanveer, I. Razzak, and K.-L. Hua, "Lightweight face anti-spoofing network for telehealth applications," *IEEE Journal of Biomedical and Health Informatics*, vol. 26, no. 5, pp. 1987–1996, 2022.
[2] Z. Wang, C. Saoud, S. Wangsiricharoen, A. W. James, A. S. Popel, and J. Sulam, "Label cleaning multiple instance learning: Refining coarse annotations on single whole-slide images," *IEEE Transactions on Medical Imaging*, vol. 41, no. 12, pp. 3952–3968, 2022.
[3] W. Wang, S. Wang, J. Gao, M. Zhang, G. Chen, T. K. Ng, and B. C. Ooi, "Rafiki: Machine learning as an analytics service system," *arXiv preprint arXiv:1804.06087*, 2018.
[4] H. Li, Y. He, L. Sun, X. Cheng, and J. Yu, "Side-channel information leakage of encrypted video stream in video surveillance systems," in *IEEE INFOCOM*, pp. 1–9, 2016.
[5] "Embedded computing market." https://www.precedenceresearch.com/embedded-computing-market, May 2022.
[6] "2020 state of telemedicine report." https://c8y.doxcdn.com/image/upload/v1599769894/Press%20Blog/Research%20Reports/2020-state-telemedicine-report.pdf, Sept. 2023.
[7] U. Subbiah, D. K. Kumar, S. K. Thangavel, and L. Parameswaran, "An extensive study and comparison of the various approaches to object detection using deep learning," *2020 Proc.IEEE ICOSEC*, vol. 5, pp. 183–194, Oct 2020.
[8] S. Xue, L. Zhang, A. Li, X.-Y. Li, C. Ruan, and W. Huang, "Appdna: App behavior profiling via graph-based deep learning," in *Proc. IEEE Infocom*, IEEE, 2018.
[9] B. Zhou, J. Lohokare, R. Gao, and F. Ye, "Echoprint: Two-factor authentication using acoustics and vision on smartphones," in *Proc.ACM MobiCom*, 2018.
[10] W. Jiang, C. Miao, F. Ma, S. Yao, Y. Wang, Y. Yuan, H. Xue, C. Song, X. Ma, D. Koutsonikolas, *et al.*, "Towards environment independent device free human activity recognition," in *Proc. ACM MobiCom*, 2018.
[11] H. Zhang, C. Xu, H. Li, A. S. Rathore, C. Song, Z. Yan, D. Li, F. Lin, K. Wang, and W. Xu, "Pdmove: Towards passive medication adherence monitoring of parkinson's disease using smartphone-based gait assessment," *in Proceedings of the ACM on interactive, mobile, wearable and ubiquitous technologies*, vol. 3, no. 3, pp. 1–23, 2019.
[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE CVPR*, 2016.
[13] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "Gazelle: A low latency framework for secure neural network inference," in *Proc. USENIX Security*, 2018.

[14] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, “Cryptflow: Secure tensorflow inference,” in *Proc. IEEE Security and Privacy*, IEEE, 2020.

[15] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, “Cryptflow2: Practical 2-party secure inference,” in *Proc. ACM CCS*, 2020.

[16] J. Fan and F. Vercauteren, “Somewhat practical fully homomorphic encryption,” *Cryptology ePrint Archive*, 2012.

[17] J. H. Cheon, A. Kim, M. Kim, and Y. Song, “Homomorphic encryption for arithmetic of approximate numbers,” in *Proc. ASIACRYPT*, 2017.

[18] Z. Brakerski, C. Gentry, and S. Halevi, “Packed ciphertexts in lwe-based homomorphic encryption,” in *Proc. PKC*, 2013.

[19] H. Chen, K. Laine, and R. Player, “Simple encrypted arithmetic library-seal v2. 1,” in *Proc. Financial Cryptography and Data Security*, 2017.

[20] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, “Homomorphic encryption security standard,” tech. rep., HomomorphicEncryption.org, Toronto, Canada, November 2018.

[21] Q. Zhang, C. Xin, and H. Wu, “GALA: Greedy ComputAtion for Linear Algebra in Privacy-Preserved Neural Networks,” in *Proc. NDSS*, 2021.

[22] Z. Huang, W. jie Lu, C. Hong, and J. Ding, “Cheetah: Lean and fast secure Two-Party deep neural network inference,” in *USENIX Security*, pp. 809–826, 2022.

[23] M. Wang, S. Ding, T. Cao, Y. Liu, and F. Xu, “Asymo: scalable and efficient deep-learning inference on asymmetric mobile cpus,” in *Proc. ACM MobiCom*, 2021.

[24] C. Wang, B. Hu, and H. Wu, “Energy minimization for federated asynchronous learning on battery-powered mobile devices via application co-running,” in *Proc. IEEE ICDCS*, 2022.

[25] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, “Mcunetv2: Memory-efficient patch-based inference for tiny deep learning,” in *Proc. NeurIPS*, 2021.

[26] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proc. CVPR*, 2018.

[27] J. H. Cheon, M. Kang, T. Kim, J. Jung, and Y. Yeo, “High-throughput deep convolutional neural networks on fully homomorphic encryption using channel-by-channel packing.” Cryptology ePrint Archive, Paper 2023/632, 2023.

[28] D. Demmler, T. Schneider, and M. Zohner, “Aby-a framework for efficient mixed-protocol secure two-party computation.,” in *Proc. NDSS*, 2015.

[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 *Proc. ICML*, pp. 201–210, PMLR, 2016.

[30] S. Wagh, D. Gupta, and N. Chandran, “Securenn: 3-party secure computation for neural network training.,” *Proc. Priv. Enhancing Technol.*, 2019.

[31] M. Bellare, V. T. Hoang, and P. Rogaway, “Foundations of garbled circuits,” in *Proc. CCS*, pp. 784–796, 2012.

[32] G. Brassard, C. Crépeau, and J.-M. Robert, “All-or-nothing disclosure of secrets,” in *Proc. CRYPTO*, 1987.

[33] A. Shamir, “How to share a secret,” *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[34] Z. Brakerski, “Fully homomorphic encryption without modulus switching from classical gapsvp,” in *Proc. CRYPTO*, 2012.

[35] I. Jibaja, T. Cao, S. M. Blackburn, and K. S. McKinley, “Portable performance on asymmetric multicore processors,” in *Proc. International Symposium on Code Generation and Optimization*, 2016.

[36] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *Proc. ICLR*, 2015.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Proc. NeurIPS*, 2017.