P^3: A Privacy-Preserving Perception Framework for Building Vehicle-Edge Perception Networks Protecting Data Privacy

Tianyu Bai, Danyang Shao, Ying He, Song Fu, and Qing Yang
Department of Computer Science and Engineering
University of North Texas, Denton, Texas
{TianyuBai, DanyangShao, YingHe}@my.unt.edu, {Song.Fu, Qing.Yang}@unt.edu

2023 32nd International Conference on Computer Communications and Networks (ICCCN) | 979-8-3503-3618-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCCN58024.2023.10230191

***Abstract*—With the wider adoption of edge computing services, intelligent edge devices, and high-speed V2X communication, compute-intensive tasks for autonomous vehicles, such as perception using camera, LiDAR, and/or radar data, can be partially offloaded to road-side edge units. However, data privacy becomes a major concern for vehicular edge computing, as sensor data with sensitive information from vehicles can be observed and used by edge servers. We aim to address the privacy problem by protecting both vehicles' sensor data and the detection results. In this paper, we present a privacy preserving perception (P^3) framework which provides a secure version of every commonly used layers in various perception CNN networks. They server as the building blocks to facilitate the construction of a privacy preserving CNN for any existing or future network. P^3 leverages the additive secret sharing theory to develop secure functions for perception networks. A vehicle's sensor data is split and encrypted into multiple secret shares, each of which is processed on an edge server by going through the secure layers of a detection network. The detection results can only be obtained by combining the partial results from the participating edge servers. We present two use cases where the secure layers in P^3 are used to build privacy preserving both single-stage and two-stage object detection CNNs. Experimental results indicate data privacy for vehicles is protected without comprising the detection accuracy and with a reasonable amount of performance degradation. To the best of our knowledge, this is the first work that provides a generic framework to ease the development of vehicle-edge perception networks protecting data privacy.**

***Index Terms*—Edge Computing, Vehicle Computing, Data Privacy, Deep Learning, Perception, Connected and Autonomous Vehicles.**

I. Introduction

Autonomous vehicles (AVs) have been attracting more and more attention and interest in both industry and academia. AVs rely on various sensors, e.g., camera, LiDAR, radar, GPS, IMU, etc., to perceive the surrounding environment and plan movement and routes [1]. To achieve autonomous driving, objects on the road should be detected accurately and quickly. The latest perception (object classification, object detection, and segmentation) methods or systems mostly use deep learning for detection. Although they are more accurate, deep learning networks are compute intensive and require powerful computing capacity on a vehicle. Furthermore, perception network is only one of the many deep learning networks that are run on a vehicle for various autonomous driving tasks.

To provide reliable computing power for delay-sensitive applications, edge computing [2] offers a cost-effective and scalable way to execute part of those deep learning workloads for nearby vehicles. This vehicle-edge computing paradigm is attractive and practical for both existing ego AVs and future connected AVs.

Privacy, however, is a big concern in vehicle-edge collaboration, as the sensor data containing sensitive information leave a vehicle and are processed on an edge server. Both the sensitive information in the input data and the object results from the detection network can be accessed and used by the edge server. Recent studies focus on designing specific operations or deep learning networks to process encrypted data, such as [3]–[8]. They protect data at the price of a prohibitive computational overhead and/or a comprised perception accuracy.

To address these issues, we propose a framework that provides building blocks to facilitate the construction of privacy-preserving perception networks in vehicle-edge systems. Our proposed P^3 framework explores the additive secret sharing theory to achieve secure operations in various layers of a CNN. Secret shares of the vehicle's sensor data are encrypted and distributed to two or more edge servers, each executing a secure CNN on a secret share. Results from those edge nodes are combined on a destination vehicle to obtain the perception results. P^3 is generic as it includes a secure version of every commonly used layer in various object detection CNNs, e.g., secure convolution, secure (leaky) ReLU, secure RPN, secure (max, average, and ROI) pooling, secure fully connected layer, secure batch normalization, etc. Developing a privacy-preserving counterpart for a new perception network is reduced to piecing together the corresponding secure layers from P^3, which is more cost-effective.

We present two use cases where the secure functions and layers in our P^3 framework are used to build a two-stage privacy-preserving perception network (PP Faster R-CNN) and a single-stage network (PP YOLO). We have implemented a prototype of P^3 and evaluated its performance on a vehicle-edge testbed. Experimental results show the secure functions on secret shares protect data privacy and do not compromise the perception accuracy, but prolong the detection time. Compared with a slowdown of four orders of magnitude with homomorphic encryption on CNN, P^3 achieves a significant speed-up, which makes privacy-preserving perception networks practical for real-world applications.

The main contributions of this paper are as follows.

- we present a novel framework for building perception networks aiming at protecting both vehicles' sensor data and

perception results from being exposed to edge servers. Our P^3 framework is generic and applicable to any existing or future perception network.

- Our design of the P^3 framework is rooted in the well-established additive secret sharing theory. Secret shares of a vehicle's sensor data are processed on two or more edge servers. Furthermore, secure functions and protocols minimize and protect the exchange of parameters between edge servers.
- P^3 provides a holistic solution to privacy-preserving perception in vehicle-edge systems, including object classification, object detection, and segmentation. To the best of our knowledge, P^3 is the first of its kind that provides a generic framework to ease the development of vehicle-edge perception networks protecting data privacy. The use cases and performance results indicate that P^3 is easy to use and cost-effective for privacy-preserving perception.

The rest of the paper is organized as follows. Section II discusses the related research. Section III provides an overview of our P^3 framework. The secure operations and design of various secure layers are detailed in Sections IV and V. Two use cases of P^3 are presented in Section VI. The performance of privacy-preserving perception networks is evaluated in Section VII. Section VIII concludes this paper with remarks on future research.

II. Related Work

Privacy protection for deep learning is an important research topic and has attracted more attention. A number of techniques have been proposed in the literature, such as homomorphic encryption and secure multi-party computation. In this section, we discuss the related research.

Leboe-McGowan et al. [9] proposed a heuristic privacy-preserving CNN for image classification. Instead of pursuing perfect ciphertext-based non-linear operations, they applied a rough approximation to evaluate CNN's non-linear transformation layers. It achieved a promising performance since the complex ciphertext-based computation was avoided. However, the accuracy of object classification was compromised, i.e., a 4% degradation of classification accuracy. P^3 targets the more comprehensive and complex perception tasks and expects to achieve the same perception accuracy as the original deep learning networks.

Xie et al. [10] and Erkin et al. [11] applied homomorphic encryption to deep learning on network-connected servers. The former devised a secure CNN model to process encrypted data and generate results in cipher text that only the owner of the data can decrypt. The latter focused on secure image classification CNNs using fully homomorphic encryption. These approaches aim to protect the input data and inference results. However, homomorphic encryption causes prohibitive computation overhead and drastic performance degradation. For example, an implementation of the fully homomorphic encryption for deep learning suffered from a slowdown by four orders of magnitude [12]. The preceding works are for image classification. The performance degradation of homomorphic encryption for object detection is even worse, making it impractical for real-world deployment.

Secure machine learning [13] provides secure protocols for training ciphertext-based CNN models using linear regression and logistic regression. A privacy-preserving deep learning framework [14] includes secure protocols for connected servers to share model parameters. In the framework, no encryption or decryption was conducted on data, and there was no guarantee that adversaries could not use those crucial parameters to attack the system.

Perception is vital for autonomous driving. Privacy protection for perception in a vehicle-edge environment has not been well studied. In this paper, we tackle this new problem and present our P^3 framework aiming to facilitate the development of privacy-preserving perception networks which are run on edge servers to process secret shares from vehicles to achieve enhanced data privacy and uncompromised perception accuracy.

III. A Privacy-Preserving Perception (P^3) Framework on the Edge

A. System Architecture and Threat model

A targeted system consists of the following entities, i.e., autonomous vehicles (denoted by V), sensor data collected by a vehicle (denoted by M), edge servers (denoted by E), and attackers (denoted by A). We employ an Honest But Curious model [15] where V is trusted, E is untrusted, and A is an attacker. A aims to obtain and reveal M, and E performs deep learning inferences to detect objects and may obtain M in the process.

In the vehicle-edge environment, we focus on the privacy of the vehicle's sensor data M. The sensor data (such as 2D images captured by on-vehicle cameras) may contain sensitive information, e.g., human faces, badges with name and/or ID, license plate numbers, locations, etc. This information may expose the identity and/or location of vehicles, persons, and other objects. Furthermore, sensor data captured along the routes of a vehicle can reveal the travel patterns and frequently visited places that the passengers want to keep private.

Traditionally, an autonomous vehicle V collects sensor data M and sends M to a roadside edge unit E for edge-assisted perception through a public wireless network; on E, it processes M and returns the detection results to V. There exist major security vulnerabilities in such cases. First, an attacker A can eavesdrop on network traffic and obtain M. Second, an edge server E can easily obtain M while performing perception computations on M. To address these vulnerabilities and protect vehicle sensor data, in this paper, we present the privacy-preserving perception (P^3) framework, which consists of secure functions and layers as building blocks for any existing and future deep learning-based perception networks.

B. Key Components of the P^3 Framework

Figure 1 depicts the key components and execution flow of our P^3 framework. It includes autonomous vehicles (e.g., V_1, V_j, V_k), edge servers (e.g., E_1, E_2), and a trusted server

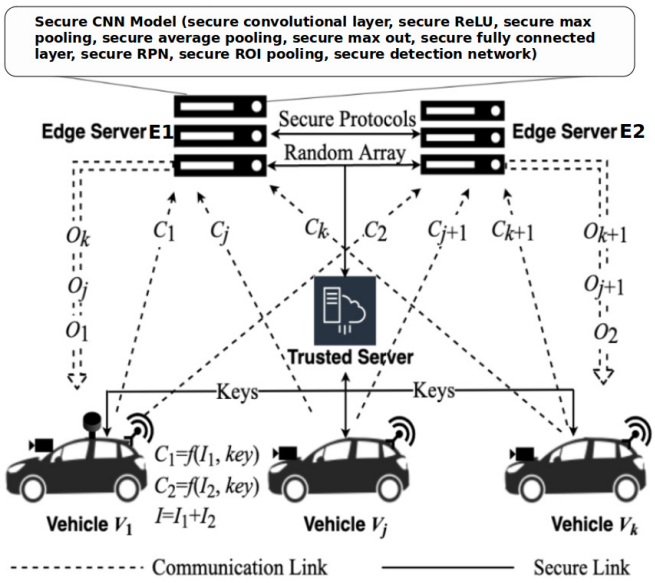

Fig. 1: Key Components and Their Interactions in the Privacy-Preserving Perception (P^3) Framework.

(TS). TS is responsible for encryption key management between AV and edge server and for secure parameter exchanges between edge servers in the execution of privacy-preserving perception networks. The perception network run on an edge server consists of a series of layers with secure functions. In P^3, secure functions have been developed for almost all types of layers used in convolutional neural networks, e.g., secure convolution, secure (leaky) ReLU, secure RPN, secure (max, average, and ROI) pooling, secure fully connected layer, secure batch normalization, etc.

Before the execution of a perception network on the edge, an autonomous vehicle (AV) and edge servers connect to the trusted server TS to obtain encryption and decryption keys used for AV-edge communications and the parameters used by secure functions on the edge. Once an AV (e.g., V_1) decides to offload its perception tasks to the edge, V_1 partitions its sensor data I into multiple secret shares each of which contains random data following the additive property $I = I_1 + ... + I_n$. These secret shares are encrypted and sent to multiple edge servers. An edge server receives one or more (not all) secret shares. As an illustration, In Figure 1, V_1 sends C_1 to E_1 and C_2 to E_2. E_1 and E_2 execute a privacy-preserving perception network CNN_{sec} on C_1 and C_2 and output O_1 and O_2, respectively. By combining the outputs from edge servers, that is $O = O_1 + O_2$, the AV can obtain the perception results. To assure the correctness of perception, the output O should be the same as that produced from running the original perception network CNN (without secure functions) on the sensor data.

The key challenge is how to design the secure functions for each layer of CNN that can process random secret shares instead of AV's sensor data while still producing correct detection results following the additive property. In the following sections, we detail the design of secure functions and layers in our P^3 framework and present two use cases applying P^3 to build privacy preserving perception networks for both single-stage and two-stage object detection.

IV. Secure Operations for Building P^3 Perception Networks

Our P^3 framework leverages the theory of additive secret sharing to design secure operations for each layer in a perception CNN network. Secret sharing has been used for cryptography in distributed systems, such as Shamir's Secret Sharing [16], Blakley's Secret Sharing [17], and proactive secret sharing [18]. In those secret sharing schemes, a secret is partitioned into multiple pieces by using a dividing function $D(s) = s_1, s_2, ..s_n$. Each piece s_i is called a share of the secret. Each secret share is kept by a different party. Recovering the secret requires at least k shares of the secret where $k <= n$. If a single party can gain trust from at least k owners of secret shares, a combing function is applied to retrieve the secret, i.e., $s = C(s_1, s_2, ..s_k)$. Secret sharing provides a cost-effective approach to protect secrets without relying on keys or complex communications. Additive secret sharing is a specific secret sharing schema where the combing function is addition, and all values are defined in a finite domain, i.e., $s = \sum_{i=1}^{i=n} s_i$. Moreover, in additive secret sharing, the threshold $k = n$, implying that a secret can only be revealed when all the secret shares are obtained by one party.

We extend the additive secret sharing theory to design secure bit-wise operations. In this section, we explain secure bit array generation, secure bit addition, and secure bit multiplication, which are applied to various layers in perception networks (Section V).

A. Secure Bit Array Generation

An edge server E_i receives a secret share M_i and processes it through a perception network. M_i is transformed to feature map(s) F_i that is passed from one layer to another in the perception network. For a given secret share set $\{F_1, F_2...F_n\}$, where the secret $F = \sum_{i=1}^{n} F_i$, is the feature value of M. Secure bit array generation (i.e., Algorithm 1) replaces each feature value secret share F_i by two bit arrays $\{T_i, r_i\}$, satisfying $\sum_{i=1}^{n} F_i = (T_1 \bigoplus T_2 \bigoplus ... \bigoplus T_n) + (r_1 \bigoplus r_2 \bigoplus ... \bigoplus r_n) = T + r$, where $\bigoplus$ denotes bit-wise addition and $+$ represents numeric addition. $\{r, \tau\}$ are two randomly generated bit arrays by the trusted server for each round of computation, ensuring identical plain text will be transformed into different cipher text. The Trusted Server TS forwards $\{r_i, \tau_i\}$ to edge server E_i. Then, each edge server E_i for $i = 1...(n-1)$ computes $d_i = F_i - \tau_i$ and a random bit array T_i, and sends $\{d_i, T_i\}$ to the edge last edge server E_n which computes $d = \sum_{i=1}^{n} d_i$ and $T_n = d \bigoplus T_1 \bigoplus T_2 \bigoplus .... \bigoplus T_{n-1}$.

By doing so, vehicle's sensor data M is divided into secret shares $\{M_1, M_2, ..., M_n\}$ and each secret share is transformed to two bit arrays $\{T_i, r_i\}$. Without knowing τ_i, which is only accessible by E_i, E_n cannot recover τ and further reconstruct the original sensor data M.

Algorithm 1 Secure Bit Array Generation.

On Edge server E_i, $i \in \{1, ..., n\}$
Input: Vehicle sensor data F_i
Output: Bit array $\{T_i, r_i\}$

Trusted Server TS generates random bit array $\{r, \tau\}$, where $r_1 \bigoplus r_2 \bigoplus ... \bigoplus r_n = \tau_1 + \tau_2 + ... + \tau_n$, and sends $\{r_i, \tau_i\}$ to Edge server E_i;
E_i $(i = 1...n)$ computes $d_i = F_i - \tau_i$ and a random bit array T_i;
since $T = T_1 \bigoplus T_2 ... \bigoplus T_n = F - \tau$
T_i are random bit array that computes from E_i where $i \in \{1, ..., n-1\}$, except for T_n.
E_i, $i \in \{1, ..., n-1\}$ forward d_i, T_i to E_n, E_n computes $d = \sum_{i=1}^{n} d_i$ and and $T_n = d \bigoplus T_1 \bigoplus T_2 .... \bigoplus T_{n-1}$;
E_n computes $d = \sum_{i=1}^{n} d_i$ and $T_n = d \bigoplus T_1 \bigoplus T_2 .... \bigoplus T_{n-1}$;
E_i return $\{T_i, r_i\}$.

Algorithm 1 transforms secret shares and produces random bit arrays. Bit arrays T_i, r_i are input to other secure bit-wise operations. For example, the generated bit arrays are used in secure bit addition. Both the addition and multiplication protocols require limited information shared by two parties to protect the secret share.

B. Secure Bit Addition

Given n (i.e., the number of secret shares) sets of bit array $\{T_i, r_i\}$ from the Secure Bit Array Generation, the Secure Bit Addition algorithm computes a boolean value π indicating whether M is more significant than zero, where $M = \sum_{i=1}^{n} h_i = (T_1 \bigoplus T_2 \bigoplus ... \bigoplus T_n) + (r_1 \bigoplus r_2 \bigoplus ... + \bigoplus r_n)$.

To protect data privacy, edge servers should not exchange the bit arrays directly. Instead of performing numeric additions, the Secure Bit Addition algorithm conducts bit-wise additions by only exchanging intermediate bit-wise products. It achieves the same outputs as numeric additions when the values of two parameters at the same location are not equal to 1 simultaneously. That is no carry to more significant bits. For example, when $n = 2$, $(T_1 \bigoplus r_1) \bigoplus (T_2 \bigoplus r_2) = T \bigoplus r = L$, then $L = T + r$ only if there does not exist two 1s at the same location in the bit array for T and r. Meanwhile, we need to send the carry to a more significant bit to obtain the correct result. To achieve this, we calculate the carry for bit arrays $\{T, r\}$ using Secure Bit Multiplication (Algorithm 3). It returns partial carries as bit arrays $\{p_1, p_2\}$ on edge servers and combines the carries $p_1 \bigoplus p_2$ to produce p which is in a binary representation to indicate the locations where both T and r have 1s.

Left-shifting p_i (Line 12 in In Algorithm 2) is to ensure the generated carry is populated to the next more significant bit's location. In bit wise representation, the bit array's first digit indicates the symbol of integer value where positive numbers have symbol value 0 while negative numbers have symbol value 1. As a result, instead of exchanging the z_i we chose to use π_i to identify the secret's partial symbol value, π indicates the bit wise symbol value of z, which is identical to secret F. At last, the protocols ensure that each side can determine the original secret F's symbol without combining its secret shares.

Algorithm 2 Secure Bit Addition.

On Edge Server E_i, $i \in \{1, ..., n\}$
Input: Bit arrays $\{T_i, r_i\}$
Output: Boolean π indicating the sign (i.e., positive or negative) of $(T + r)$

p_i = Secure_Bit_Multiplication(T_i, r_i);
$z_i = T_i \bigoplus r_i$;
p_i updates value by left-shifting one digit;
Edge servers exchange p_i;
while $(p_1 \bigoplus p_2 \bigoplus ... \bigoplus p_n \neq 0)$ **do**
    Edge server E_i computes $z_i = z_i \bigoplus p_i$;
    p_i = Secure_Bit_Multiplication(z_i, p_i);
    E_i updates p_i by left-shifting one bit and forwards it to other edge servers;
end while
if $(z_i < 0)$ **then**
    $\pi_i = 1$;
else
    $\pi_i = 0$;
end if
Edge servers exchange π_i and compute $\pi = \pi_1 \bigoplus \pi_2 \bigoplus ... \bigoplus \pi_n$;
return π.

Algorithm 3 Secure Bit Multiplication.

On Edge Server E_i, $i \in \{1, ..., n\}$
Input: bit arrays $\{T_i, r_i\}$
Output: partial carry p_i from $T \bigotimes r$

The Trusted Server TS randomly generates 3 sets of secure bit arrays: $n = n_1 \bigoplus n_2 \bigoplus ... \bigoplus n_n, \beta = \beta_1 \bigoplus \beta_2 \bigoplus ... \bigoplus \beta_n, h = h_1 \bigoplus h_2 \bigoplus ... \bigoplus h_n$, $h = \beta \bigotimes n, \chi = rand\{1, n\}$ where $\chi \in N^*$;
Edge Server E_i computes $T_i \bigoplus \beta_i, r_i \bigoplus n_i$ and exchanges the results with other edge servers, then combines to get $A = T \bigoplus n, B = r \bigoplus \beta$;
$Q_i = \beta_i \bigotimes A$, $W_i = n_i \bigotimes B$, $\theta = A \bigotimes B$;
if $(\chi == i)$ **then**
    $p_i = h_i \bigoplus Q_i \bigoplus W_i \bigoplus \theta$;
else
    $p_i = h_i \bigoplus Q_i \bigoplus W_i$;
end if
return p_i.

C. Secure Bit Multiplication

secure bit multiplication receives two bit arrays $\{T_i, r_i\}$ and produces single bit array p_i where $i \in \{1, n\}$. Combining p_i by XOR should retrieve bit wise multiplication of T and r which

is the carrying value required in secure bit addition. secure bit multiplication requires additional sets of random parameters n, β, h, χ to facilitate computing the multiplication of $\{T, r\}$. The correctness of the algorithm can be proved by applying the distribution law over conjunction and disjunction for logical operations.

In Algorithm 3, χ is a random integer between 1 and n and it is used to determine if θ is used to calculate p_i. We note that $\theta = A \bigotimes B$ combined with $n \bigoplus B$ is eliminated in calculating $p_1 \bigoplus p_2 ... \bigoplus p_n$. Thus, using θ on either side does not affect the output. Here, $\bigotimes$ denotes bit-wise multiplication.

V. Building Blocks in P^3: Secure Functions and Layers

P^3 incorporates a comprehensive set of secure functions/layers that process secret shares (input) to produce feature maps (output), yielding the same detection results as the original perception networks. The inference process of a perception network can be expressed as $v = L_i(L_{i-1}(\dots(L_1(x))))$, where each L_i conduct a set of linear or nonlinear operations in the perception network.

Our design of P^3 assures that the output of every layer in a perception network follows the additive secret sharing requirement. More specifically, as the sensor data M is divided into n secret shares, the outputs of any layer on participating edge servers satisfy $(v_1 + v_2 + ... + v_n)$ equals to the output of that layer in the original perception network run on M, where v_i denotes the output of the secure layer executed on an edge server E_i.

For a linear transformation (denoted by G_{linear}) on an input feature map F, $G_{linear}(F) = G_{linear}(F_1 + F_2 + ... + F_n) = G_{linear}(F_1) + G_{linear}(F_2) + ... + G_{linear}(F_n)$. That is the transformation does not alter the additive property of secret shares. We use the original operations in a layer to process a secret share. However, this does not hold for a non-linear transformation and special design and operations are needed to build a secure counterpart. We have developed the secure versions for all the known types of layers used in various perception networks. In this section, we explain our design for a number of major layers.

A. Secure Convolution, Secure Fully Connected Layer, and Layers Containing Only Linear Transformations

The convolutional layer, fully connected layer and detection network consist of linear transformations only. Thus, we can take the same computations in their secure counterparts. The only difference is that the input is replaced by the secret shares of the original input.

B. Secure Max Pooling and Secure Average Pooling

Max pooling selects the most significant features from a feature map for a given stride. The maximum operation on a set of features is nonlinear. Finding the maximum value from multiple secret shares is challenging as the additive relation among secret shares does not mean that a feature in one share is greater than its counterparts in other shares and the values of features in different secret shares cannot be exchanged between edge servers for privacy protection. To address this challenge, we design an iterative difference evaluation method sketched in Algorithm 4. In Lines 10 and 11, each edge server computes the difference between secret shares corresponding to index (α, β), then exchanges the differences to calculate the feature difference I. With I, each edge server can locate the feature having the greatest value.

Average pooling produces the average value of features in a region of the input feature map. To achieve secure average pooling, we combine the means from secret shares (without revealing a secret share itself) and divide the sum by the number of shares, i.e., $M = (m_1 + m_2 + \dots + m_n)/n$.

Algorithm 4 Secure Max Pooling (S-MAXPOOL).

1: On Edge Server E_i, $i \in \{1, 2, ...n\}$
2: Input: feature map secret share F_i
3: Output: partial maximum feature values
4:
5: **for** each channel j in feature map F_i **do**
6: F_{ij} = the feature map F_i in the j_{th} channel;
7: w, h are max pooling strides;
8: **for** each feature max pool stride region R in F_{ij}, w in $[0, 1]$, h in $[0, 1]$ **do**
9: Pooling index $\alpha = 0, \beta = 0$;
10: $I_i = R[\alpha][\beta] - R[w][h]$;
11: E_i, E_q exchange I_i, I_q and compute $I = I_i + I_q$;
12: **if** $I < 0$ **then**
13: $\alpha = w, \beta = h$;
14: **end if**
15: **end for**
16: return $R[\alpha][\beta]$.
17: **end for**

C. Secure Region Proposal Network

A region proposal network (RPN) takes feature maps as input and predicts object bounds and objectness scores at each position. In the secure RPN (Algorithm 5), the input is a secret share of feature map. The anchor box generation part remains unchanged as the locations of anchor points are fixed. Each edge server generates a large number of anchor boxes, with each individual box having identical coordinates across all secret shares. Note secure RPN only produces partial anchor box offsets and credit scores. Thus, edge servers exchange and combine those partial results to obtain the actual scores and offsets of the generated boxes.

In addition, the non-maximum suppression (NMS) requires the credit score of each bounding box to remove overlapping boxes. Thus, the combined credit score from edge servers is used by NMS to filter out overlapping bounding boxes and keep those with reliable credit scores.

D. Secure Region of Interest Pooling

Region of interest (ROI) pooling converts all the proposals to fixed shape as required by a detection network. Secure

Algorithm 5 Secure Region Proposal Network (S-RPN).

On Edge Server E_i, $i \in \{1, 2, ...n\}$
Input: feature map secret share F_i
Output: partial region proposals

E_i generates the anchor boxes set B_i for F_i;
for each anchor box b_i in B_i **do**
    E_i computes partial credit score and offset $c_i, o_i = Lin(b_i)$;
    Edge servers exchange $\{c_i, o_i\}$, and compute $c = \sum_{i=1}^{n} c_i$, $o = \sum_{i=1}^{n} o_i$;
    Proposals are produced by applying offset o to anchor boxes B_i;
    NMS identifies and removes overlapped anchor boxes $B_i = NMS(B_i, c)$;
end for
return B_i, c.

ROI pooling takes a secret share of feature map from a backbone network and region proposals from RPN as input. Region proposal-based feature extraction is performed using the secret share to obtain proposal patches P_i. As shown in Algorithm 6, each patch p is flattened to a one-dimensional array with size $(w(p) * h(p))$. A difference table T_i with size $(w(p) * h(p))^2$ is created. The table $T_i[m][n]$ stores the difference η for $p[m]$ and $p[n]$ on an edge server E_i. Edge servers exchange their difference tables through TS and then calculate $T = T_1 + T_2 + \cdots + T_n$ which represents the difference of feature values from F on the patch p (i.e., $T[m][n] = p[m] - p[n]$). If all the values in row v of T are positive, then the corresponding value in $p[v]$ is greater than others.

E. Secure ReLU and Secure Leaky ReLU

ReLU introduces non-linearity to feature maps by applying a rectifier function. Specifically, ReLU preserves positive features while replacing negative ones by zero. However, for secure ReLU, the input is a secret share of a feature map. We design the secure ReLU by leveraging the secure bit-wise operations presented in Section IV to extract secret's sign (positive or negative). First, we use secure bit generation (Algorithm 1) to produce bit arrays from a secret share. The bit arrays are then used as parameters to the secure bit addition (Algorithm 2) which performs bit-wise addition on the bit arrays with the combined carry produced by the secure bit multiplication (Algorithm 3).

The secure bit addition produces a partial sign π_i satisfying $\pi = \pi_1 \bigoplus \pi_2 \bigoplus ... \bigoplus \pi_n$ has the same sign as that in F, where 1 means negative and 0 is positive. Thus, the secure ReLU can perform the rectifier function correctly even without knowing the actual values of features.

Leaky ReLU is similar to ReLU with the difference in the return value when a feature is negative. Specifically, for a negative feature, leaky ReLU multiplies the feature by a coefficient (e.g., 0.1), rather than simply returning zero. The secure leaky ReLU is designed in a similar way. Due to the space limit, we do not include its pseudocode in this paper.

Algorithm 6 Secure ROI Pooling (S-ROIPOOL).

On Edge Server E_i, $i \in \{1, 2, ...n\}$
Input: feature map secret share F_i, region proposal RP
Output: fixed-size partial maximum features in region proposals

Extract region-based features K_i from F_i with the scaled proposal's coordinates μ, υ generated from the input proposals RP;
Calculate a width stride = width$(K_i)/w$ and a height stride = height$(K_i)/h$;
Divide K_i into patches P_i based on the width stride and height stride;
for each patch p in P_i **do**
    Flatten p into one dimension;
    Create difference table T_i with a size $(w(p) * h(p))^2$;
    $T_i[m][n] = p[m] - p[n]$;
    E_i exchanges the difference table T_i with other edge servers and collaboratively computes the difference table $T = \sum_{i=1}^{n} T_i$;
    for each row in T **do**
        if all the values in that row are greater than 0 **then**
            return the index of the row, v.
        end if
    end for
    return $p[v]$.
end for

Algorithm 7 Secure ReLU (S-ReLU).

On Edge Server E_i, $i \in \{1, 2, ...n\}$
Input: feature map secret share F_i
Output: partial positive features or zero

for each feature f in the secret share F_i **do**
    $\{T_i, r_i\}$ = Secure_Bit_Array_Generation(F_i);
    if (Secure_Bit_Addition$(T_i, r_i) == 1$) **then**
        $f = 0$;
    end if
    return F_i.
end for

VI. Use Cases of P^3

A. Use Case 1: Building Privacy-Preserving Two-Stage Perception Network

The P^3 framework provides the building blocks to facilitate the construction of privacy-preserving CNNs for various perception tasks, such as object classification, object detection, and segmentation. In this case study, we select a two-stage object detection network to illustrate the application of P^3 to real-world systems.

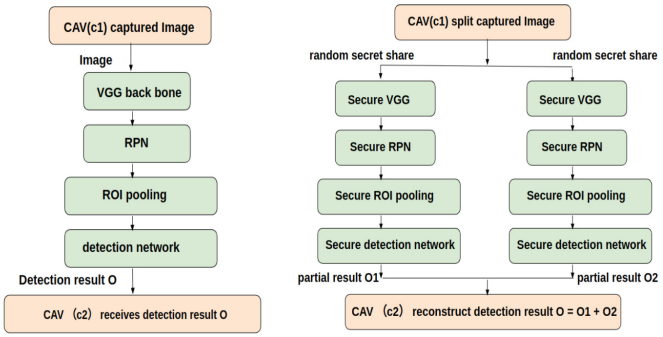

Fig. 2: Use case 1: Building a privacy-preserving two-stage perception network using P^3.

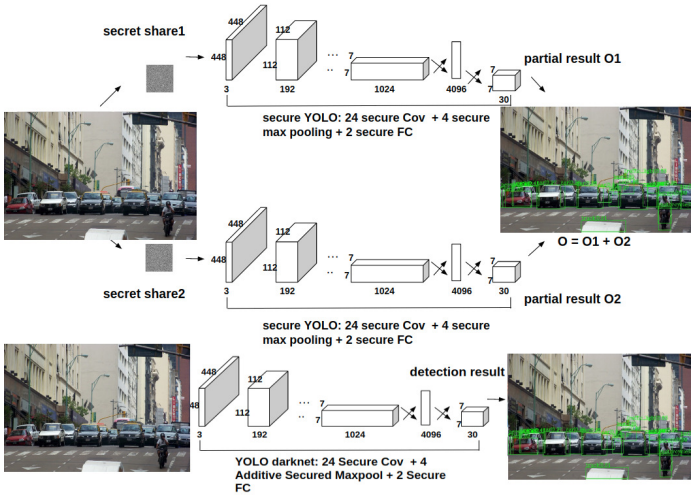

Fig. 3: Use case 2: Building a privacy-preserving single-stage perception network using P^3.

Two-stage object detectors divide the detection task into two phases, i.e., 1) locating potential objects with bounding boxes, called region proposals, and 2) forwarding the region proposals to a prediction network to compute classification scores. Region-based CNNs, such as R-CNN [19], Fast R-CNN [20], and Faster R-CNN [21], are widely used two-stage object detection networks. R-CNN detects objects by extracting regions of interest with bounding boxes on images. To determine region proposals, a selective search algorithm [22] is used.

The original Faster R-CNN network does not protect input data, which allows edge servers to easily access vehicles' images. To create a privacy-preserving Faster R-CNN (PP Faster R-CNN in short) network, we use the secure functions and layers in P^3 to replace the components in Faster R-CNN, i.e., backbone CNN, Region proposal network (RPN), ROI pooling, and detection network. Figure 2 depicts the workflow of a PP Faster R-CNN compared with the original network.

The VGG 16 backbone includes 13 Convolutional layers, 13 ReLU layers, and 5 Max Pooling layers, which extract features of size 50*50*512 (width, height, channel). RPN produces 2,000 anchor boxes for a feature map. ROI pooling reshapes the feature map combined with region proposals to the same size, and the detection network generates the detection result.

The PP Faster R-CNN takes a similar architecture, i.e., secure VGG backbone, secure RPN, secure ROI pooling, and secure detection network. The secure VGG backbone is composed of 13 secure convolutional layers, 13 secure ReLU layers, and five secure max pooling layers. The P^3 framework provides all of those layers. We simply replace the original layers by the secure versions.

To use the PP Faster R-CNN in a real-world environment, a vehicle creates two or more secret shares $(M_1, M_2, ...)$ for an image from its sensor. These secret shares are transferred to edge servers, protected by encryption with keys obtained from TS. Each edge server processes a secret share through the PP Faster R-CNN. The outputs from the edge servers are combined $(O_1 + O_2 + ...)$ to obtain the detected objects.

B. Use Case 2: Building Privacy-Preserving Single-Stage Perception Network

A single-stage object detector scans an image only once to produce both classifications and bounding boxes of objects. Usually, single-stage detectors are faster than two-stage ones. YOLO [23] is a widely used single-stage detection network. The architecture of YOLO with Darknet is depicted in Figure 3. It consists of 24 convolutional layers, four max pooling layers, and two fully connected layers. Leaky ReLU is used in all layers except for the last activation layer. Darknet computes features from an image and divides into fixed grids and an underlying CNN predicts the confidence score and classification score for each grid cell.

The privacy-preserving YOLO (PP YOLO in short) contains secure convolutional layers, secure max pooling layers, secure leaky ReLU, and secure fully connected layers, as illustrated in Figure 3. The workflow using PP YOLO is similar with that using PP Faster R-CNN, involving partitioning vehicle images, executing PP YOLO on multiple edge servers, and combining outputs.

The YOLO family includes a series of enhanced networks which improve the accuracy of bounding boxs' placement and object classification. For example, YOLO9000 [24] applies batch normalization and anchor boxes in Darknet19, and YOLO3 [25] applies Darknet53 with 53 convolutional layers and uses logistic regression to calculate the objectiveness score of each bounding box. Our P^3 framework provides secure functions/layers to build the privacy-preserving version of those enhanced YOLO networks. Furthermore, the building blocks in P^3 are generic and applicable to future perception networks.

VII. Performance Evaluation

We have implemented a prototype P^3 system and several proof-of-concept privacy-preserving perception networks using P^3. We evaluate their performance on a vehicle-edge

Fig. 4: Privacy-preserving perception network built on P^3 achieves the same accuracy as the original network.

	PP Faster R-CNN(P^3)	PP YOLO (P^3)
Car	3.24 e^{-7}	4.26 e^{-7}
Traffic Light	2.11 e^{-7}	8.71 e^{-8}
Stop Signs	1.19 e^{-7}	3.5 e^{-7}
Pedestrian	6.4 e^{-8}	4.24 e^{-7}
Bicycle	6.9 e^{-7}	1.24 e^{-7}

Fig. 5: MAPE in perception by privacy-preserving (PP) Faster R-CNN and YOLO.

testbed. A Polaris GEM e4 electric vehicle is equipped with a set of Sekonix cameras, one Velodyne LiDAR, one Delphi radar, GPS and IMU sensors. The on-vehicle processing unit is AStuff Spectra. Each edge server has an AMD Ryzen 7 processor with 6 cores at 3.2 GHz and 16 GB DRAM, and runs Ubuntu Linux v20.04 and Python v3.8. We conducted our experiments on the COCO dataset [26], a widely used object-detection dataset. Among the 80+ categories of objects in the COCO dataset, the experiment focuses on transportation-related objects, such as vehicles, traffic signs, and pedestrians. In our experiments, we build privacy-preserving CNNs (PP CNNs in short) using pre-trained models. The PP CNNs are implemented on PyTorch v0.4.0 and Torch v0.20. The models are trained by using 5,000 images from COCO with 135 epochs, 40 batches, and a 0.01 learning rate. 350 images from COCO containing the 15 most frequent street view object classes are selected for inference.

A. Perception Accuracy

The perception results by PP Faster R-CNN and PP YOLO are depicted in Figure 5. In total, 35 object categories are evaluated and PP Faster R-CNN and PP YOLO correctly detect all of them.

We further calculate the mean absolute percentage error (MAPE) between the PP Faster R-CNN and PP YOLO and the original Faster R-CNN and YOLO respectively. Figure 5 lists the MAPE results which indicate PP CNNs achieve the same accuracy as the original CNNs.

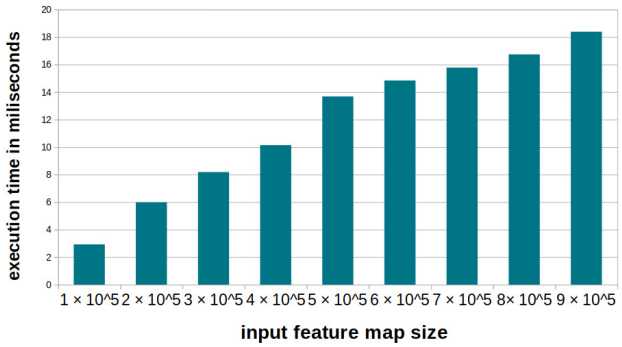

Fig. 6: Performance of secure bit addition.

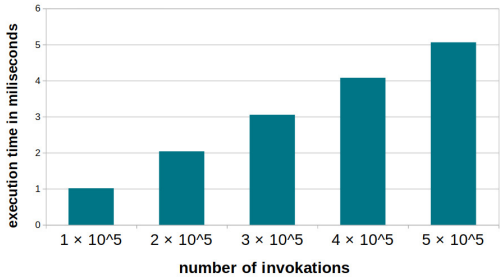

Fig. 7: Performance of secure bit multiplication.

B. Results of Perception Time

In order to understand the speed of a PP CNN built from P^3, we measure the execution time of the major operations and secure layers. These include the secure bit addition (Figure 6), secure bit multiplication (Figure 7), secure convolution layer (Figure 8), secure ReLU (Figure 9), secure Max pooling (Figure 10), secure Average pooling (Figure 11), secure RPN (Figure 12), and secure ROI pooling(Figure 13).

Here are some important findings. The execution time of the secure bit multiplication increases linearly with the number of invocations and the input size, whereas that of the secure bit addition depends mostly on the input size. The execution time of the secure ReLU increases exponentially with the input size caused by the random bit array generation and secure bit addition. The execution times of secure Convolution, Max pooling, and Average pooling are also affected by the input size. The execution time of the secure RPN increases as the number of anchor boxes increases. The secure ROI pooling has a longer execution time caused by the construction of the difference table and comparison, especially for large bounding boxes.

Figure 15 compares the inference time of PP Faster R-CNN and PP YOLO, which is 28.398 seconds and 3.081 seconds, respectively. The results show that most of the degradation comes from the secure functions for non-linear operations. This is due to the expensive bit-wise operations, including secure bit array generation, secure bit addition, and secure bit multiplication.

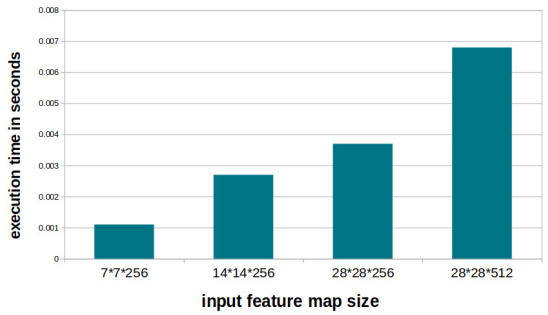

Fig. 8: Performance of secure Convolution.

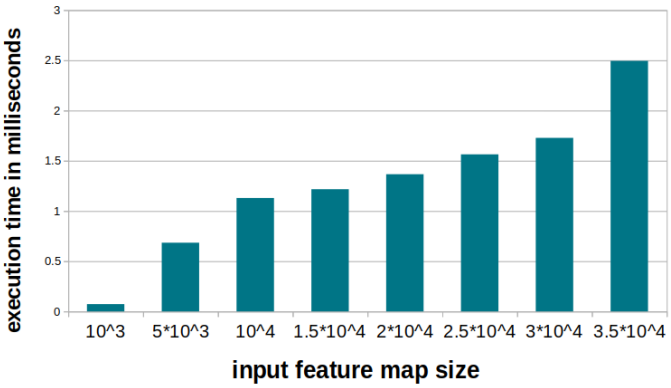

Fig. 9: Performance of secure ReLU.

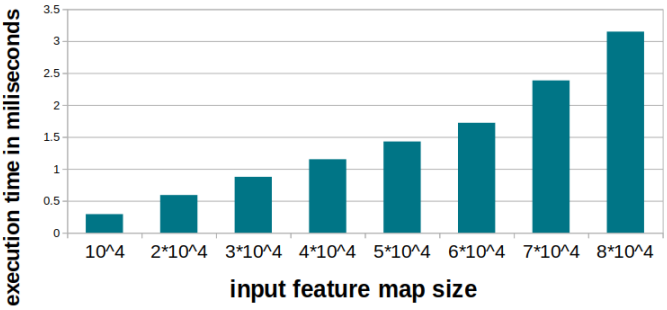

Fig. 10: Performance of secure Max Pooling.

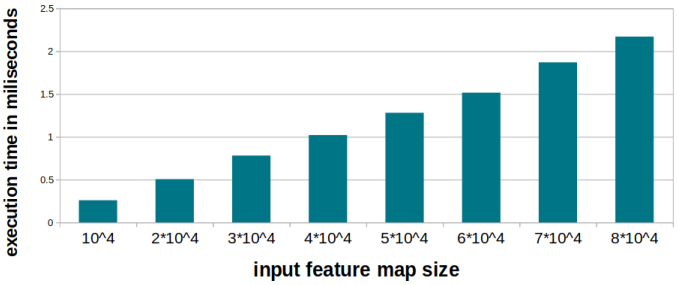

Fig. 11: Performance of secure Average Pooling.

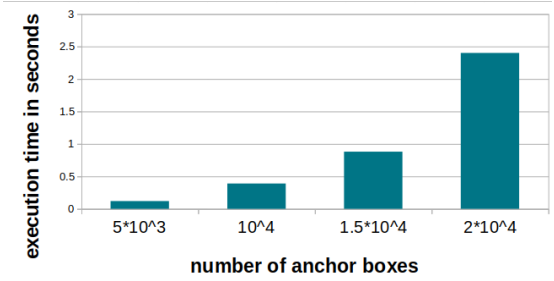

Fig. 12: Performance of secure RPN.

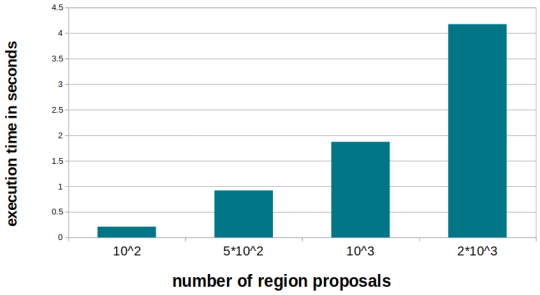

Fig. 13: Performance of secure ROI Pooling.

We also measure the additional storage and network bandwidth used by PP CNNs. The data transfer time by PP Faster R-CNN and PP YOLO is 15.16 seconds and 12.3 seconds, respectively, with a 300 Mbps bandwidth. PP Faster R-CNN displays a longer data transfer time due to the two-stage structure of its perception network which transfers additional data for secure RPN and secure ROI pooling. The amount of data included in secret shares, feature maps, and intermediate data from secure operations is compared in Figure 14. PP YOLO produces a smaller amount of data due to the fewer number of layers and smaller input size.

Figure 16 compares the inference performance of PP YOLO and PP Faster R-CNN built by using our P^3 framework and that of CryptoNets which uses a leveled homomorphic encryption scheme. It is clear that both PP YOLO and PP Faster R-CNN are significantly faster than CryptoNets, i.e., a

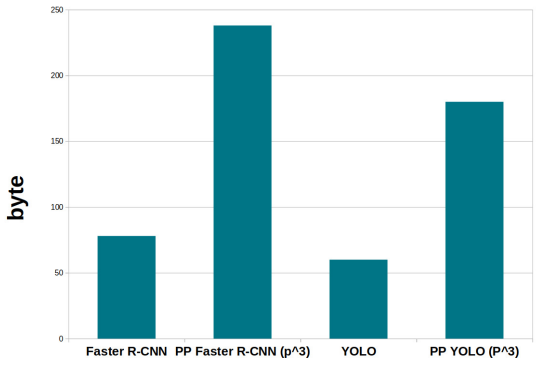

Fig. 14: Comparison of data storage between a perception network and the privacy-preserving version.

	Faster R-CNN		PP Faster R-CNN (P^3)
convolutional	0.42s	secure convolutional	0.44s
ReLU	0.154s	secure ReLU	19.13s
max pooling	0.021s	secure max pooling	1.156s
RPN	1.26s	secure RPN	2.66s
ROI pooling	1.04s	secure ROI pooling	4.174s
detection network	0.792s	secure detection network	0.838s
total	3.687s	total	28.398s

	YOLO		PP YOLO (P^3)
convolutional	0.039s	secure convolutional	0.042s
leaky ReLU	0.012s	secure leaky ReLU	2.814s
max pooling	0.02s	secure max pooling	0.125s
fully connected	0.014s	secure fully connected	0.1s
total	0.085s	total	3.081s

Fig. 15: Execution time of different layers in a perception network vs. those in the privacy-preserving version.

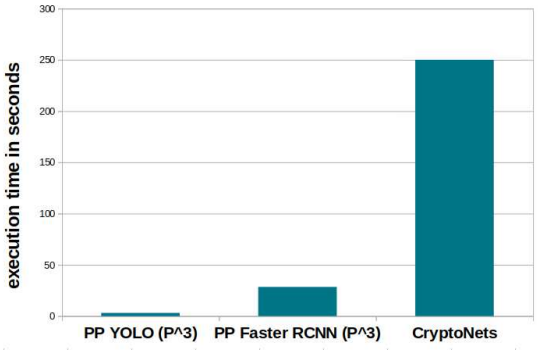

Fig. 16: Performance comparison between privacy-preserving CNNs (using P^3) with CryptoNets.

83.3X and 8.8X speedup, respectively. This shows the privacy-preserving perception networks built from P^3 are more pratical for real-world applications.

VIII. Conclusion

Perception is crucial for autonomous driving. We study the privacy protection of vehicles' sensor data and perception results in vehicle-edge environments. We present a novel framework (P^3) that incorporates secure functions for all the known types of layers used in various perception networks. These building blocks ease the construction of privacy-preserving perception networks for edge servers to process sensor data from vehicles while protecting data privacy. Experimental results show that perception on multiple edge servers achieves the same accuracy with reasonable performance degradation. In our future research, we will optimize and accelerate the secure functions/layers in P^3 to speed up perception for real-time applications.

Acknowledgement

This work has been supported in part by the U.S. National Science Foundation grants CNS-2231519, CNS-2113805, CNS-1852134, OAC-2017564, ECCS-2010332, CNS-2037982, DUE-2225229, and CNS-1828105.

References

[1] S. Campbell, N. O'Mahony *et al.*, "Sensor technology in autonomous vehicles: A review," in *IEEE ISSC*, 2018.
[2] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1697–1716, 2019.
[3] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.
[4] C. Gentry, *A fully homomorphic encryption scheme*. Stanford University, 2009.
[5] D. Stehlé and R. Steinfeld, "Faster fully homomorphic encryption," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2010, pp. 377–394.
[6] N. P. Smart and F. Vercauteren, "Fully homomorphic encryption with relatively small key and ciphertext sizes," in *Workshop on Public Key Cryptography*, 2010.
[7] Y. Sun, R. Lu, X. Lin, X. Shen, and J. Su, "An efficient pseudonymous authentication scheme with strong privacy preservation for vehicular communications," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 7, pp. 3589–3603, 2010.
[8] C. Juvekar, V. Vaikuntanathan *et al.*, "Gazelle: A low latency framework for secure neural network inference," in *USENIX Security*, 2018.
[9] D. Leboe-McGowan, M. M. Al Aziz, and N. Mohammed, "Simple approximations for fast and secure deep learning on genomic data," in *IEEE IEMCON*, 2020.
[10] P. Xie, M. Bilenko, T. Finley, R. Gilad-Bachrach, K. Lauter, and M. Naehrig, "Crypto-nets: Neural networks over encrypted data," *arXiv preprint arXiv:1412.6181*, 2014.
[11] Z. Erkin, T. Veugen, T. Toft, and R. L. Lagendijk, "Generating private recommendations efficiently using homomorphic encryption and data packing," *IEEE Trans. on Information Forensics and Security*, vol. 7, no. 3, pp. 1053–1066, 2012.
[12] N. J. H. Marcano, M. Moller *et al.*, "On fully homomorphic encryption for privacy-preserving deep learning," in *IEEE Globecom*, 2019.
[13] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *IEEE SP*, 2017.
[14] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *ACM CCS*, 2015.
[15] O. Goldreich, *Foundations of cryptography: volume 2, basic applications*. Cambridge University Press, 2009.
[16] M. Naor and A. Shamir, "Visual cryptography," in *Workshop on the Theory and Application of of Cryptographic Techniques*, 1994.
[17] E. F. Brickell, "Some ideal secret sharing schemes," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1989, pp. 468–475.
[18] A. Herzberg, S. Jarecki *et al.*, "Proactive secret sharing or: How to cope with perpetual leakage," in *Crypto*, 1995.
[19] R. Girshick, J. Donahue *et al.*, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *IEEE CVPR*, 2014.
[20] R. Girshick, "Fast r-cnn," in *IEEE ICCV*, 2015.
[21] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, pp. 91–99, 2015.
[22] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International journal of computer vision*, vol. 104, no. 2, pp. 154–171, 2013.
[23] J. Redmon, S. Divvala *et al.*, "You only look once: Unified, real-time object detection," in *IEEE CVPR*, 2016.
[24] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *IEEE CVPR*, 2017.
[25] ——, "Yolo v3: An incremental improvement," *arXiv:1804.02767*, 2018.
[26] T.-Y. Lin, M. Maire *et al.*, "Microsoft coco: Common objects in context," in *ECCV*, 2014.