

Dissertations and Theses

---

11-2019

## Efficient Privacy-Aware Imagery Data Analysis

Yifan Tian

Follow this and additional works at: <https://commons.erau.edu/edt>



Part of the [Electrical and Computer Engineering Commons](#), and the [Privacy Law Commons](#)

---

This Dissertation - Open Access is brought to you for free and open access by Scholarly Commons. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of Scholarly Commons. For more information, please contact [commons@erau.edu](mailto:commons@erau.edu).

EMBRY-RIDDLE AERONAUTICAL UNIVERSITY

DOCTORAL DISSERTATION

---

# Efficient Privacy-Aware Imagery Data Analysis

---

*Author:*

Yifan Tian

*Advisor:*

Dr. Jiawei Yuan

*Committee Members:*

Dr. Radu Babiceanu

Dr. Yantian Hou

Dr. Remzi Seker

Dr. Houbing Song

Dr. Tianyu Yang

*A dissertation submitted in fulfillment of the requirements*

*for the degree of Doctor of Philosophy*

*in Electrical Engineering & Computer Science*

Department of Electrical, Computer, Software, & Systems Engineering

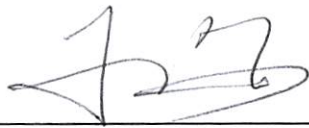
November 2019

# EFFICIENT PRIVACY-AWARE IMAGERY DATA ANALYSIS

By

Yifan Tian

This dissertation was prepared under the direction of the candidate's Dissertation Committee Chair, Dr. Jiawei Yuan and has been approved by the members of the dissertation committee. It was submitted to the College of Engineering and was accepted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Electrical Engineering and Computer Science.



Jiawei Yuan, Ph.D.  
Committee Chair



Radu Babiceanu, Ph.D.  
Committee Member



Yantian Hou, Ph.D.  
Committee Member



Remzi Seker, Ph.D.  
Committee Member



Houbing Song, Ph.D.  
Committee Member



Tianyu Yang, Ph.D.  
Committee Member



Timothy Wilson, Sc.D.  
Department Chair, ECSSE

2019-11-11

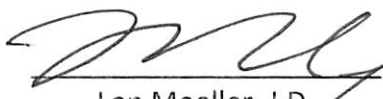
Date



Maj Mirmirani, Ph.D.  
Dean, College of Engineering

11/11/19

Date



Lon Moeller, J.D.  
Senior Vice President for  
Academic Affairs and Provost

11/11/19

Date

*“I am not merely seeking an ‘outcome’ since human beings prefer to find shortcuts to that outcome. Once I am on that shortcut, probably I will miss the truth I am seeking as well as my passion for the truth. The critical thing is the will to seek the truth, even if I fail this time, as long as the will exists, I will find the truth someday because I will always keep going.”*

Leone Abbacchio

## Abstract

The widespread use of smartphones and camera-coupled Internet of Thing (IoT) devices triggers an explosive growth of imagery data. To extract and process the rich contents contained in imagery data, various image analysis techniques have been investigated and applied to a spectrum of application scenarios. In recent years, breakthroughs in deep learning have powered a new revolution for image analysis in terms of effectiveness with high resource consumption. Given the fact that most smartphones and IoT devices have limited computational capability and battery life, they are not ready for the processing of computational intensive analytics over imagery data collected by them, especially when deep learning is involved. To resolve the bottleneck of computation, storage, and energy for these resource constrained devices, offloading complex image analysis to public cloud computing platforms has become a promising trend in both academia and industry. However, an outstanding challenge with public cloud is on the protection of sensitive information contained in many imagery data, such as personal identities and financial data. Directly sending imagery data to the public cloud can cause serious privacy concerns and even legal issues.

In this dissertation, I propose a comprehensive privacy-preserving imagery data analysis framework which can be integrated in different application scenarios to assist image analysis for resource-constrained devices with *efficiency*, *accuracy*, and *privacy* protection. I first identify security challenges in the utilization of public cloud for image analysis. Then, I design and develop a set of novel solutions to address these challenges. These solutions will be featured by strong privacy guarantee, lightweight computation, low accuracy loss compared with image analysis without privacy protection. To optimize the communication overhead and resource utilization of using cloud computing, I investigate

edge computing, which is a promising technique to ameliorate the high communication overhead in cloud-assisted architectures. Furthermore, to boost the performance of my solutions under both cloud and edge deployment, I also provide a set of pluggable enhancement modules to be applied to meet different requirements for various tasks. By exploring the features of edge computing and cloud computing, I flexibly incorporate them as a comprehensive framework to provide privacy-preserving image analysis services.

# *Acknowledgements*

First of all, I would like to thank my Ph.D. advisor, Dr. Jiawei Yuan, for supporting me during the past four years. Dr. Yuan is a professional full of knowledge and wisdom and he's one of the smartest people I know. He is a great advisor who always leads me with patience and also a good and warm friend in daily life. I am very fortunate to have worked with Dr. Yuan and I hope that I could be as lively, enthusiastic, and energetic as him and someday be able to flourish in academia works as he does.

I want to thank all my committee members for years of guidance and collaboration in my Ph.D. life. I am grateful that Dr. Radu Babiceanu provided me kind help with my career as well as his fantastic mentoring of system engineering related topics. Great thanks to Dr. Yantian Hou for his long-time collaboration starting from my very first full paper. I also want to give thanks to Dr. Remzi Seker for his encouragement on my dissertation and our discussion regarding network security and cryptocurrencies. It has been an honor to work with Dr. Houbing Song and his SONG Lab. They extended me a great helping hand when I was stuck on the antenna and hardware issues. I thank God for meeting Dr. Tianyu Yang on my first day at ERAU and being introduced to the local community physically and spiritually.

I also need to express my gratitude to everyone who gave me help in academia and industry. I highly appreciate Dr. Shucheng Yu for his feedbacks on our blockchain project and our discussion on all the other topics. Many thanks to Prof. Farahzad Behi, Dr. Keith Garfield and Dr. Timothy Wilson for the opportunity to work as a lab instructor for CS 225 and I really harvested a lot from the year-long instructing. I owe Jian Wang and Dr. Yongxin Liu a thank you for their "Build UAV from Scratch 101" series and "UAV Resurrection" series after rookie pilot Yifan crashed the experimenting

drone. I will not forget the discussion on those machine learning theories with Renkun Ni, which showed me a different angle of machine learning from a statistic/mathematics perspective. I also want to recognize the best teammate, Ashok Vardhan Raja, best thank you for all your efforts in our Best of Session UAV paper. Kudos to all my other collaborators Dr. Laurent Njilla, Alexander Steinbacher, Thaniel Tong, and Jayson Tinsley in our UAV project. I would like to thank Yushan Jiang as well for his time and our discussion to optimize resource-constrained neural network implementation. Huge credit to Dr. Markus Jakobsson for his advising in our anti-phishing research and his lead during my internship. Cheers to all ACID team members at Agari Data, Inc., Crane Hassold, James Linton and Ronnie Tokazowski for the industry-level projects and insights on cybersecurity and software engineering areas. And I thank Agari and fellow Agarians for offering me an unforgettable internship experience.

I want to thank my mother, Wenwei Yuan, my father, Xiaodong Tian for their selfless support so that I am able to explore the world of cybersecurity and become a better me. I praise the Lord, my Father in heaven for all the wisdom, will and bless he granted me. Finally and most importantly, I would like to thank my wife, Mengxin Cui for all the late nights and early mornings, for all the life routines and occasional surprises, and for always being there for me throughout up and downs in my life. I want to dedicate this milestone of my life to them for their unconditional love and support.



# Contents

<b>Abstract</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>v</b>
<b>Contents</b>	<b>vii</b>
<b>List of Figures</b>	<b>x</b>
<b>List of Tables</b>	<b>xi</b>
<b>Abbreviations</b>	<b>xii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Challenges . . . . .	2
1.3 Contributions . . . . .	6
1.4 Roadmap . . . . .	7
<b>2 Problem Formulation</b>	<b>9</b>
2.1 Cloud-assisted Privacy-preserving Descriptor Based Image Analysis . . . . .	9
2.1.1 System Model . . . . .	9
2.1.2 Threat Model . . . . .	10
2.2 Edge-assisted Privacy-preserving Deep Learning Based Image Analysis . . . . .	11
2.2.1 System Model . . . . .	11
2.2.2 Threat Model . . . . .	12
<b>3 Privacy Protection for Descriptor Based Image Analysis</b>	<b>13</b>
3.1 Introduction . . . . .	13
3.2 Related Works . . . . .	15
3.3 Preliminaries . . . . .	17
3.3.1 Image Descriptor Extraction . . . . .	17
3.3.2 Integer Vector Encryption (IVE) . . . . .	18
3.4 Privacy-preserving Distance Comparison . . . . .	19
3.4.1 Image Similarity Measurement . . . . .	19
3.4.2 <i>PL1C</i> : Privacy-preserving $L_1$ Distance Comparison . . . . .	20
3.4.3 <i>PKLC</i> : Privacy-preserving KL-Divergence Comparison . . . . .	22

3.5	Cloud Assisted Privacy-preserving Image Annotation . . . . .	23
3.5.1	System Setup . . . . .	24
3.5.2	Dataset Encryption . . . . .	25
3.5.3	Secure Annotation Request . . . . .	25
3.5.4	Privacy-preserving Annotation on Cloud . . . . .	26
3.5.5	Final Keyword Selection . . . . .	27
3.6	Preliminaries of CPAR . . . . .	28
3.6.1	k-dimension Tree . . . . .	29
3.6.2	Order-preserving Encryption . . . . .	29
3.7	Privacy-preserving Distance Comparison with Randomized k-d Forest . . . . .	29
3.7.1	Randomized k-d Forest Search . . . . .	30
3.7.2	<i>PL1C-RF</i> : Privacy-preserving $L_1$ Distance Comparison for Randomized k-d Forest . . . . .	32
3.7.3	<i>PKLC-RF</i> : Privacy-preserving KL-Divergence Comparison for Randomized k-d Forest . . . . .	33
3.8	Cloud-Assisted Privacy-preserving Image Annotation with Randomized k-d Forest . . . . .	36
3.8.1	Detailed Construction of CPAR . . . . .	36
3.8.1.1	System Setup . . . . .	36
3.8.1.2	RKDF Encryption . . . . .	37
3.8.1.3	Secure Annotation Request . . . . .	37
3.8.1.4	Privacy-preserving Annotation on Cloud . . . . .	38
3.8.1.5	Final Keyword Selection . . . . .	40
3.9	Conclusion . . . . .	40
<b>4</b>	<b>Evaluation of Privacy Protection Modules for Descriptor Based Image Analysis</b> . . . . .	<b>43</b>
4.1	Security Analysis . . . . .	43
4.1.1	Security of Outsourcing $\mathbf{S}_{L1}^T \mathbf{S}_{s,L1}$ and $\mathbf{S}_{KL}^T \mathbf{S}_{s,KL}$ . . . . .	44
4.1.2	Known Ciphertexts-Image Pairs . . . . .	45
4.1.3	Request Unlinkability . . . . .	45
4.2	Evaluation of CAPIA . . . . .	47
4.2.1	System Setup and Dataset Encryption . . . . .	48
4.2.2	Real-time Image Annotation . . . . .	49
4.2.3	Communication Cost and Storage Overhead . . . . .	51
4.3	Evaluation of CPAR . . . . .	52
4.3.1	RKDF Construction and Encryption . . . . .	53
4.3.2	Real-time Image Annotation . . . . .	53
4.4	Conclusion . . . . .	58
<b>5</b>	<b>Privacy Protection for Deep Learning Based Image Analysis</b> . . . . .	<b>59</b>
5.1	Introduction . . . . .	59
5.2	Related Work . . . . .	63
5.3	Preliminaries - Convolutional Neural Network . . . . .	64
5.4	Privacy-preserving Compute-intense Layers . . . . .	66
5.4.1	<i>PPCL</i> : Privacy-preserving Convolutional Layer . . . . .	66
5.4.2	<i>PPFL</i> : Privacy-preserving Fully-connected Layer . . . . .	68

5.5	Edge-Assisted CNN Inference over Encrypted Imagery Data . . . . .	69
5.5.1	Offline Phase . . . . .	70
5.5.2	Online Phase . . . . .	72
5.6	Discussion - Storage and Update of Pre-computed Keys . . . . .	72
5.7	Discussion - Offloading Pooling Layers . . . . .	74
5.8	Enhancement - Integrity Check . . . . .	76
5.9	Enhancement - Fast Convolution . . . . .	78
5.10	Enhancement - Matrix Compression . . . . .	80
5.11	Conclusion . . . . .	81
<b>6</b>	<b>Evaluation of Privacy Protection Modules for Deep Learning Based Image Analysis</b>	<b>82</b>
6.1	Security and Performance Analysis . . . . .	82
6.1.1	Security Analysis . . . . .	82
6.1.2	Numerical Analysis . . . . .	85
6.1.2.1	Computational Cost . . . . .	85
6.1.2.2	Communication Cost . . . . .	90
6.1.2.3	Storage Overhead . . . . .	91
6.1.2.4	Resource Consumption of Integrity Check . . . . .	91
6.1.2.5	Analysis of Fast Convolution . . . . .	94
6.2	Evaluation . . . . .	97
6.2.1	Efficiency - Offline Phase . . . . .	97
6.2.2	Efficiency - Online Phase . . . . .	97
6.2.3	Energy Consumption . . . . .	101
6.2.4	Accuracy . . . . .	101
6.2.5	Evaluation of Sample Rate in Integrity Check . . . . .	102
6.2.5.1	Evaluation of Matrix Compression . . . . .	103
6.3	Conclusion . . . . .	104
<b>7</b>	<b>Future Works and Conclusion</b>	<b>107</b>
7.1	Extension of Descriptor Based Image Analysis . . . . .	107
7.2	Extension of Deep Learning Based Image Analysis . . . . .	108
7.3	A Privacy-preserving Hybrid Cloud-Edge Framework for Image Analysis . . . . .	109
7.4	Conclusion . . . . .	110

# List of Figures

3.1	Randomized k-d Forest . . . . .	30
3.2	Construction of $PLIC - RF$ . . . . .	34
3.3	Construction of $PKLC - RF$ . . . . .	35
4.1	Error rate of Approximation and Dimension of Approximated Vector ( $PCA - 32$ ) . . . . .	46
4.2	(a) System Setup and Encryption Cost (b) Request Generation Cost (c) Distance Comparison Candidate Generation Cost on Cloud . . . . .	46
4.3	Precision of CAPIA and Annotation without Encryption . . . . .	50
4.4	Recall of CAPIA and Annotation without Encryption . . . . .	50
4.5	Privacy-preserving Annotation Cost on Cloud with Different Approxima- tion Power . . . . .	54
4.6	Speedup Rate with Different Approximation Power . . . . .	54
4.7	Accuracy (Recall) of CPAR with Different Approximation Power . . . . .	55
4.8	Speedup rate of CPAR with Different Accuracy Compared with CAPIA . . . . .	56
5.1	Examples of a Convolutional Layer and a Fully-connected Layer . . . . .	65
5.2	Key Update for Power Connected Devices . . . . .	73
5.3	Example of Pooling Layers . . . . .	75
6.1	Evaluation of Sample Rate $r$ and Error Detection Rate . . . . .	102
6.2	Evaluation of Sample Rate $r$ and Returned Data Size . . . . .	103
6.3	Evaluation of Convolutional Layers and Offloaded Computation Percentage	103

# List of Tables

4.1	Sample Annotation Results	52
4.2	Sample Annotation Comparison between CAPIA and CPAR	57
5.1	Summary of Notations	67
6.1	Numerical Analysis Summary	86
6.2	Example Numerical Analysis on AlexNet	88
6.3	Example Numerical Analysis on FaceNet	89
6.4	Numerical Analysis of Integrity Check	92
6.5	Example Comparison with/without Integrity Check	93
6.6	Efficiency Enhancement Analysis on AlexNet	95
6.7	Efficiency Enhancement Analysis on FaceNet	96
6.8	Experimental Evaluation Results on AlexNet	98
6.9	Comparison Between My Scheme and CryptonNets in First Convolutional Layer of AlexNet	99
6.10	Experimental Evaluation Results on Integrity Check	100
6.11	Power and Energy Consumption Evaluation	101
6.12	Communication Enhancement on AlexNet	104
6.13	Communication Enhancement on FaceNet	105

# Abbreviations

<b>ANN</b>	<b>Artificial Neural Network</b>
<b>CNN</b>	<b>Convolutional Neural Network</b>
<b>CAPIA</b>	<b>Cloud-Assisted Privacy-preserving Image Annotation</b>
<b>CPAR</b>	<b>Cloud-assisted Privacy-preserving Image Annotation using Randomized k-d Forest</b>
<b>DNN</b>	<b>Deep Neural Network</b>
<b>FLOP</b>	<b>Float Operation</b>
<b>FMV</b>	<b>Full-Motion Video</b>
<b>GAN</b>	<b>Generative Adversarial Network</b>
<b>HE</b>	<b>Homomorphic Encryption</b>
<b>IoT</b>	<b>Internet of Thing</b>
<b>IVE</b>	<b>Integer Vector Encryption</b>
<b>JL</b>	<b>Johnson-Lindenstrauss</b>
<b>KL</b>	<b>Kullback-Leibler</b>
<b>LWE</b>	<b>Learn with Error</b>
<b>LSTM</b>	<b>Long Short-Term Memory</b>
<b>MEC</b>	<b>Mobile Edge Computing</b>
<b>MPC</b>	<b>Multi-party Computation</b>

---

<b>OPE</b>	<b>Order-preserving Encryption</b>
<b>OT</b>	<b>Oblivious Transfer</b>
<b>PAHE</b>	<b>Packed Additive Homomorphic Encryption</b>
<b>PCA</b>	<b>Principle Component Analysis</b>
<b>PKLC</b>	<b>Privacy-preserving KL-Divergence Comparison</b>
<b>PKLC-RF</b>	<b>Privacy-preserving KL-Divergence Comparison for Randomized k-d Forest</b>
<b>PL1C</b>	<b>Privacy-preserving L1 Distance Comparison</b>
<b>PL1C-RF</b>	<b>Privacy-preserving L1 Distance Comparison for Randomized k-d Forest</b>
<b>PPCL</b>	<b>Privacy-preserving Convolutional Layer</b>
<b>PPFL</b>	<b>Privacy-preserving Fully-connected Layer</b>
<b>PPT</b>	<b>Probabilistic Polynomial Time</b>
<b>ReLU</b>	<b>Rectified Linear Unit</b>
<b>RCNN</b>	<b>Regional Convolutional Neural Network</b>
<b>RKDF</b>	<b>Randomized k-d Forest</b>
<b>SE</b>	<b>Searchable Encryption</b>
<b>SMID</b>	<b>Single Instruction Multiple Data</b>
<b>SVM</b>	<b>Support Vector Machine</b>
<b>UAV</b>	<b>Unmanned Aerial Vehicle</b>

*To my parents and my wife*



# Chapter 1

## Introduction

### 1.1 Motivation

From the first digital picture was taken, various research efforts have been spent on image analysis, since imagery data contain a great amount of rich information. Image analysis techniques have been adopted in a spectrum of scenarios, including classification [1–5], annotation [6–8], segmentation [9–12], object detection [13–16], etc. These techniques greatly fulfill the semantic gap between low level image pixels and high level human perceivable concepts.

Meanwhile, the widespread use of smartphones brings the explosive growth in the number of pictures taken. Forever’s Strategy & Business Development team [17] has pointed out that the number of photos taken by smartphone is estimated to be 8.8 trillion in 2018. Alongside the thriving mobile computing and Internet of Things (IoTs), the demands of effective image analysis have become stronger ever. To perform effective imagery data analysis, there are two major types of approaches: *Descriptor Based Approach* and *Deep Learning Based Approach*.

- *Descriptor Based Approach:* In the descriptor based image analysis, a set of features (also known as local descriptors), such as color space features and texture features, will be extracted from images. These features describe images from different perspectives, and can be further analyzed to extract information from images using different algorithms. The descriptor based image analysis has been widely adopted in computer vision tasks such as image classification [18, 19], annotation [6–8], object detection [20, 21] etc. For general image analysis without complex models and subsequent training, a set of global low-level image features including RGB, HSV, LAB, Gabor and Haar are chosen as the baseline image annotation technique [22]. Efficiency enhancement and privacy protection modules for these descriptors will be introduced in Chapter 3.
- *Deep Learning Based Approach:* Unlike the descriptor based approaches that require a specific descriptor, the deep learning [23] based approaches train artificial neural networks (ANNs) to automatically select features for specific image analysis tasks. One of the most commonly used artificial neural networks is convolutional neural network (CNN). Compared with descriptor based approaches, CNN based deep learning models has significantly enhanced the effectiveness in many image analysis application scenarios, including image classification [1–5], object detection [13–16], face recognition [24, 25, 25, 26] etc.

## 1.2 Challenges

While these approaches provide decent functionalities for image analysis, they are resource consuming for complex tasks, especially when being executed on resource-constrained mobile and IoT devices. In the descriptor based approaches, large-scale

datasets are required to ensure the accuracy of image analysis tasks, such as object recognition, image annotation, etc. The involvement of these large-scale datasets inevitably cause high computation and storage cost to fulfill the tasks. In deep learning based approaches, a request image needs to go through each layer of a deep neural network to complete the inference process. In prevailing CNN based deep architectures such as VGG [3] and ResNet [5], the analysis of an image costs billions of float operations (FLOPs). In addition, in these deep architectures, fully connected layers need to store millions of parameters, and thus resulting a considerable storage overhead.

Not only the image analysis task can bring huge burden to mobile and IoT devices, images per se can yield storage problem to these devices. On-board storage for mobile and IoT devices becomes limited as the image resolution grows higher. To facilitate the long-term storage and image analysis task of high-volume photos taken everyday, majority of smartphones today are synchronizing their photo albums with cloud storage, such as Apple's iCloud, Samsung Cloud, and Google Photos. Besides the storage service, these cloud storage platforms also help analyze and provide a few decent features helping their users organize their photos. An example is that a large portion of cloud service providers annotate users' photos with proper keywords, which is the key enabler for users to perform popular keyword based search and organization over their photos. In fact, offloading complex image analysis tasks to public cloud platforms has become a prevailing trend for resource-constrained devices [27, 28]. Public cloud service not only offers sufficient computation and storage resources to guarantee the efficiency of image analysis and storage services, but also provides a higher portability for the deployment of services.

Despite these decent features brought by public cloud service, it also raises privacy concerns. One outstanding challenge with public cloud, however, is on the protection of

sensitive information involved in images while offloading the analysis tasks. As a matter of fact, many images are sensitive by nature and contain various sensitive information, such as financial information, personal identities/locations, and healthcare information [29]. Directly sending images to public clouds can raise not only privacy concerns, but also legal issues [30]. To protect the privacy of photos, encrypting them with standard encryption algorithms, e.g., AES, is still the major approach for privacy protection in cloud storage [31, 32]. However, this kind of encryption also sacrifices many other attractive functionalities of cloud storage, especially for keyword based search and management for imagery files. How to protect the privacy of imagery data while utilizing the fancy features brought by public cloud services becomes an open challenge.

Targeted at privacy-preserving image analysis in public cloud, existing researches have spent a significant amount of efforts to design solutions for descriptor based image analysis [33–35] and deep learning based image analysis [36–44] respectively. While these solutions achieve various functionality of image analysis on public cloud in a privacy-preserving manner, expensive cryptographic primitives utilized in them (e.g., homomorphic encryption and multi-party secure computation) introduce heavy encryption and communication overhead to mobile and IoT devices. As a result, their efficiency is restricted for complex tasks due to the intensive computation cost. Specifically, such a performance limitation makes these solutions far away from practical in support of time-sensitive deep learning inference tasks on IoT devices. For example, a quad-core Raspberry Pi, which outperforms most resource-constrained IoT devices in terms of computational capability, can perform only four Paillier homomorphic encryption per second [45]. Given a single input of a typical deep learning model, AlexNet [46], which has  $227 \times 227 \times 3$  elements, it requires more than 10 hours to complete the encryption, which is impractical for most applications in terms of time efficiency. Moreover, some

existing schemes, e.g. CryptoNets [47], utilize approximation to meet security requirement of homomorphic encryption and thus introducing an accuracy loss to the image analysis task.

In addition to the challenge of balancing privacy and computation costs in utilizing public cloud, another challenge is how to reduce the communication overhead during imagery data processing, especially for devices that requires real-time data analytics. Taking cloud-assisted unmanned aerial vehicles (UAVs) as an example, the real-time transmission of high-definition images or full-motion video (FMV) from UAVs to cloud is bandwidth demanding. This kind of bandwidth demanding constant transmission can quickly drain the battery life of UAVs. Moreover, high network latency is also introduced, which limits their applications in time-sensitive tasks that require fast-response for imagery data analysis, such as disaster detection and search-and-rescue. Motivated by such a fact, recent research introduced “edge-computing” to ameliorate the communication cost when utilizing cloud-assisted architecture [48]. By deploying edge computing resources that are close to devices, initial data processing tasks can be carried out, and only critical information will be transmitted to the cloud. As a result, the “data drowning” issue that causes high network latency can be mitigated in cloud-assisted architectures. Nevertheless, existing researches either do not consider privacy issues when using edge devices, or place full trust on them. There still lacks research efforts to address privacy concerns when third-party edge computing resources is utilized, which is analogous to the use of public cloud. As a matter of fact, mobile-edge computing (MEC) [49] provided by third-party base station is one of the most important edge-computing resources. This is because MEC can easily offer one-hop communication for most mobile and IoT devices, and can host sufficient computational resources for required data processing.

### 1.3 Contributions

In order to address the aforementioned challenges and develop a generic methodology for imagery data analysis under different scenarios, three rubrics, *efficiency*, *accuracy*, as well as *privacy* need to be taken into consideration at the same time. I made my contributions by figuring out a perfect balance among these three rubrics. Due to the nature of some deployment environment (e.g. resource-constrained IoT devices), I also evaluate other rubrics such as storage cost and energy consumption. In this dissertation, I design a few expendable modules to be plugged in to meet different requirements in various situations. With the knowledge that imagery data analysis tasks could be deployed in cloud/edge environments using either descriptor/deep learning based approaches, I apply these modules and investigate their performance in the following research directions. Specifically, I make my contribution by demonstrating the practical use and performance of my modules in these directions.

- *Direction 1: Cloud-assisted Privacy-preserving Descriptor Based Image Analysis.*

Regarding the direction of descriptor based image analysis, I use one of the most important image analysis tasks, i.e., automatic image annotation as an application scenario to my privacy-preserving solution for descriptor based image analysis. Image annotation techniques extract appropriate keywords for an image, which serve as the fundamental part for in-depth image analysis, e.g., object detection and search in images, similarity measurement of images, etc. My privacy-preserving solution enables offloading complex automatic image annotation tasks to public cloud, which will be featured by high efficiency, strong privacy protection, and low accuracy loss. Efficient data structures and indexing techniques are

also explored to further improve the efficiency of privacy-preserving computation.

The results of this research direction are presented in Chapter 3 and Chapter 4.

- *Direction 2: Edge-assisted Privacy-preserving Deep Learning Based Image Analysis.* In order to provide a privacy-preserving solution in the deep learning based image analysis direction, I tailor and deploy CNN models on edge devices to privately assist real-time image analysis on resource-constrained IoT devices. I mainly focus on the inference stage of deep learning analysis, which directly fits real-time imagery data analysis nature for most IoT devices. In this task, I first seek to identify computational and storage intensive layers in the CNN based deep learning. Then, efficient privacy-preserving offloading schemes will be developed for these layers to support image analysis. Privacy protection modules are designed independently for each type of CNN layers, which enables their flexible integration to support different image analysis tasks relied on various CNN structures. The research results of this task are presented in Chapter 5 and Chapter 6.

## 1.4 Roadmap

The rest of this dissertation is organized as follows:

- Chapter 2 defines the overall problem formulation, high level system model and threat model.
- Chapter 3 presents my research results of a cloud-assisted privacy-preserving image annotation scheme as demonstration of my privacy-preserving modules for descriptor based image analysis.

- 
- Chapter 4 analyzes the security of privacy-preserving distance comparison modules under cloud-assisted scenarios presented in Chapter 3 and evaluates their practical performance.
  - Chapter 5 states my research results of an edge-assisted offloading scheme of deep learning models as demonstration of my privacy-preserving modules for deep learning based image analysis.
  - Chapter 6 first discusses the security of edge-assisted privacy-preserving compute-intensive layer modules presented in Chapter 5 and then evaluates their practical performance.
  - Chapter 7 illustrates future research directions and provides a conclusion for this dissertation.



## Chapter 2

# Problem Formulation

In order to dive deep into the two research directions mentioned in the end of last chapter, I formulate the underlying problems and define corresponding system and threat model for each direction.

### 2.1 Cloud-assisted Privacy-preserving Descriptor Based Image Analysis

#### 2.1.1 System Model

In the direction of cloud-assisted descriptor based image analysis, I consider two entities: a *Cloud Server* and a *User*. The user stores his/her images on cloud, and then, based on the extracted and processed image descriptors, the cloud helps the user to analyze his/her images without learning the contents of images. In this scenario, the user first performs a one-time system setup that prepares an encrypted large scale dataset, which is offloaded to the cloud server to assist future privacy-preserving image analysis. Later

on, when the user has a new image to be analyzed, he/she generates an encrypted request and sends it to the cloud. After processing the encrypted request, the cloud returns ciphertexts of analysis results and auxiliary information to the user. Finally, the user decrypts all the data returned from the cloud, and based on the decrypted data, the user is able to generate the final analysis result for the requested image.

### 2.1.2 Threat Model

In this cloud-assisted descriptor based image analysis scenario, I consider the cloud server to be “curious-but-honest”, i.e., the cloud server will follow a designated scheme to perform storage and image analysis services correctly, but it may try to learn sensitive information in user’s data. The cloud server has access to all encrypted images, encrypted descriptors, encrypted auxiliary information, the user’s encrypted requests, and encrypted analysis results. I also assume the user’s devices are fully trusted and will not be compromised. The research on protecting user devices is orthogonal to this research direction. These assumptions are consistent with major research works that focus on search over encrypted data on public cloud [50–52]. This scenario model focuses on preventing the cloud server from learning following information: 1) contents of the user’s images; 2) extracted descriptors and analysis result for each image; 3) request linkability, i.e., tell whether multiple analysis requests are from the same image.

## 2.2 Edge-assisted Privacy-preserving Deep Learning Based Image Analysis

### 2.2.1 System Model

For the research direction of privacy-preserving deep learning based image analysis, in my setting, there are two entities: *IoT Device* and *Edge Computing Device*. The IoT device collects imagery data and needs to utilize CNN inference over the imagery data to get analysis result. The edge device obtains a trained CNN model and contributes its computing capability to the CNN imagery data inference task. There are two scenarios for the offloading of CNN inference according to the provider of the trained model: (1) the data holder deploys its own trained model on a computing service platform and later on submits data for inference tasks [36]; (2) the computing service platform offers the trained model and performs inference on data submitted by the data holder, which is known as “machine learning as a service” [41–44]. When privacy is taken into consideration, both scenarios require the protection of imagery data and inference results against the computing service platform, and the second scenario also needs to prevent the data holder from learning the trained model.

In this dissertation, I focus on the first scenario. To be specific, I wish to design a scheme that the IoT device and edge device engage in, at the end of which the IoT device obtains the CNN inference result over its image analysis request, whereas the edge device only assists the computation without learning the details of the image as well as the analysis result.

### 2.2.2 Threat Model

Similar as the setting of last research direction, I consider the edge device to be “curious-but-honest”, i.e., the edge device adheres to the protocol that describes the computation, communication, and storage tasks, but attempts to infer information about the image input and output of the IoT device’s CNN inference task. Given a CNN based image analysis inference task, the edge device has access to the trained CNN model (offloaded convolutional layers and fully-connected layers only) as well as the encrypted image analysis request and the outputs of all offloaded layers. The IoT device is considered to be fully trusted and will not be compromised.

I aim at preventing the edge device from learning the IoT devices’ inputs and outputs of each offloaded layer. The overall purpose of the inference task is not going to be protected, since the CNN model is known to the edge device. For example, the edge device knows the CNN inference is used for object detection, but shall not learn the input image data and the corresponding detection result. I assume that the CNN model is trained by IoT device owner with data in the clear. To prevent privacy leakage of training data from the CNN model, a statistical database can be used for training as discussed in the differential privacy literature [53, 54]. The research on privacy-preserving training is orthogonal to this work.

## Chapter 3

# Privacy Protection for Descriptor Based Image Analysis

To sail in the first research direction regarding privacy-preserving descriptor based image analysis, I use one of the most important image analysis tasks, i.e., automatic image annotation in a cloud involved environment as an application. In this application, the user offloads his/her image to the cloud to get annotated with a set of keywords. My application scenario and problem formulation is the same as in the corresponding section of Chapter 2, in which the cloud follows the designated algorithm to assist the annotation but is curious about the content of the user uploaded image.

### 3.1 Introduction

Automatic image annotation has been an important and challenging task in computer vision area. One of the critical contributions of image annotation is to establish a semantic link between imagery data and textual description so that such applications

as keyword-based image search on public cloud can become possible. In order to enable keyword-based search and management on encrypted data in cloud, keyword-based searchable encryption (SE) has been widely investigated in recent years [50–52, 55, 56]. An SE scheme typically provides encrypted search indexes constructed based on proper keywords assigned to data files. With these encrypted indexes, the data owner can submit encrypted keyword-based search request to search their data over ciphertexts. Unfortunately, these SE schemes all assume that keywords are already available for files to be processed, which is hard to be true for photos taken by smartphones. Specifically, unlike text files that support automatic keyword extraction from their contents, keywords assignment for imagery files relies on manual description or automatic annotation based on a large-scale pre-annotated image dataset. From the perspective of user experience, manually annotating each image from users’ devices is clearly an impractical choice. Meanwhile, automatic image annotation that involves large-scale image datasets is too resource-consuming to be developed on smartphones. Although currently several cloud storage platforms offer image annotation services [57, 58], these platforms require access to unencrypted images. Therefore, how to provide efficient and privacy-preserving automatic annotation for smartphones’ photos becomes the foundation of SE schemes applications on smartphones.

To address this problem, I introduce a cloud-assisted privacy-preserving image annotation scheme (CAPIA). By tailoring homomorphic encryption over vector space, I first design two privacy-preserving outsourcing schemes for  $L_1$  distance comparison and Kullback-Leibler (KL) Divergence comparison respectively as building modules for CAPIA. As a result, CAPIA is able to offload the image annotation process to the public cloud in privacy-preserving manner. In addition, thanks to the underlying linear operations, CAPIA can be easily parallelized for cloud computing environment. Meanwhile,

CAPIA achieves comparable annotation accuracy compared with existing no-privacy-preserving image annotation approaches. Furthermore, as the same keyword may have different importance for the semantic description of different images, I design a real-time weight to support accurate final keywords selection in the image annotation process.

To turbocharge CAPIA's annotation efficiency with privacy protected, I further design a novel privacy-preserving randomized k-d forest structure for cloud assisted annotation (CPAR). I first combine operations for image annotation with the data search using randomized k-d forest [59]. Then, by proposing a set of privacy-preserving comparison schemes, my scheme enables the cloud server to perform image annotation directly over an encrypted randomized k-d forest structure. Compared with CAPIA, CPAR offers an adjustable speedup rate from  $4\times$  to  $43.1\times$  while achieving 97.7% to 80.3% accuracy of CAPIA. Note that besides imagery data analysis, my privacy-preserving randomized k-d forest design can also be used as independent tools for other related fields, especially for these requiring similarity measurement on encrypted data.

## 3.2 Related Works

To automate the keywords extraction process for images, a number of research works have been proposed with the concept of "automatic image annotation" [22, 60–62]. Chappelle et al. [63] trained support vector machine (SVM) classifiers to achieve high annotation accuracy where the only available image features are high dimensional histograms. In ref [64, 65], SVM was used to learn regional information as well as helped segmentation and classification process simultaneously. Different from SVM which works by finding a hyperplane to separate vector spaces, Bayesian network accomplishes the annotation tasks by modeling the conditional probabilities from training samples. In

ref [66, 67], Bayesian networks were built by clustering global image features to calculate the conditional probabilities. However, all of these image annotation works raise privacy issues when delegated to the cloud since unencrypted images need to be outsourced. Therefore, to address such privacy concerns, I propose CPAR, which utilizes the power of cloud computing to perform automatic image annotation for users, while only providing encrypted image information to the cloud.

As a follow-up issue of automatic image annotation, in order to solve the problem of how to search over encrypted data, the idea of keyword-based searchable encryption (SE) was first introduced by Song et.al in ref [55]. Later on, with the widespread use of cloud storage services, the idea of SE received increasing attention from researchers. In ref [50, 56], search efficiency enhanced SE schemes are proposed based on novel index constructions. After that, SE schemes with the support of multiple keywords and conjunctive keywords are investigated in ref [51], and thus making the search more accurate and flexible. Recently, fuzzy keyword is considered in ref [52], which enables SE schemes to tolerate misspelled keyword during the search process. While these SE schemes offer decent features for keyword-based search, their application to images are limited given the question that how keywords of images can be efficiently extracted with privacy protection. It is impractical for cloud storage users to manually annotate their images.



### 3.3 Preliminaries

#### 3.3.1 Image Descriptor Extraction

According to [22], a common baseline image analysis method is based on Global low-level image features including RGB, HSV, LAB, Gabor and HAAR because they can be applied to general image analysis without complex models and subsequent training. In particular, RGB feature is computed as a normalized 3D histogram of RGB pixel, in which each channel (R,G,B) has 16 bins that divide the color space values from 0 to 255. The HSV and LAB features can be processed similarly as RGB, and thus I can construct three feature vectors for RGB, HSV and LAB respectively as  $\mathbf{V}_{RGB}$ ,  $\mathbf{V}_{HSV}$ , and  $\mathbf{V}_{LAB}$ . Texture features of an image are extracted using Gabor and Haar wavelets. Specifically, an image is first filtered with Gabor wavelets at three scales and four orientations, resulting in twelve response images. Each response image is then divided into non-overlapping rectangle blocks. Finally, mean filter response magnitudes from each block over all response images are concatenated into a feature vector, denoted as  $\mathbf{V}_G$ . Meanwhile, a quantized Gabor feature of an image is generated using the mean Gabor response phase angle in non-overlapping blocks in each response image. These quantized values are concatenated into a feature vector, denoted as  $\mathbf{V}_{GQ}$ . The Haar feature of an image is extracted similarly as Gabor, but based on differently configured Haar wavelets. HaarQ stands for the quantized version of Haar feature, which quantizes Haar features into [0,-1,1] if the signs of Haar response values are zero, negative, and positive respectively. I denote feature vectors of Haar and HaarQ as  $\mathbf{V}_H$  and  $\mathbf{V}_{HQ}$  respectively. Therefore, given an image, seven feature vectors will be extracted as  $[\mathbf{V}_{RGB}, \mathbf{V}_{HSV}, \mathbf{V}_{LAB}, \mathbf{V}_G, \mathbf{V}_{GQ}, \mathbf{V}_H, \mathbf{V}_{HQ}]$ . For more details about the adopted image feature extraction, please refer to ref [22].

### 3.3.2 Integer Vector Encryption (IVE)

In this section, I describe a homomorphic encryption scheme designed for integer vectors [68], which will be tailored in my construction to achieve privacy-preserving image annotation. For expression simplicity, following definitions will be used in the rest of this chapter:

- For a vector  $\mathbf{V}$  (or a matrix  $\mathbf{M}$ ), define  $|\max(\mathbf{V})|$  (or  $|\max(\mathbf{M})|$ ) to be the maximum absolute value of its elements.
- For  $a \in \mathbb{R}$ , define  $\lceil a \rceil$  to be the nearest integer of  $a$ ,  $\lceil a \rceil_q$  to be the nearest integer of  $a$  with modulus  $q$ .
- For matrix  $\mathbf{M} \in \mathbb{R}^{n \times m}$ , define  $\text{vec}(\mathbf{M})$  to be a  $nm$ -dimensional vector by concatenating the transpose of each column of  $\mathbf{M}$ .

**Encryption:** Given a  $m$ -dimensional vector  $\mathbf{V} \in \mathbb{Z}_p^m$  and the secret key matrix  $\mathbf{S} \in \mathbb{Z}_q^{m \times m}$ , output the ciphertext of  $\mathbf{V}$  as

$$\mathbf{C}(\mathbf{V}) = \mathbf{S}^{-1}(w\mathbf{V} + \mathbf{e})^T \quad (3.1)$$

where  $\mathbf{S}^{-1}$  is the inverse matrix of  $\mathbf{S}$ ,  $T$  is the transpose operator,  $\mathbf{e}$  is a random error vector,  $w$  is an integer parameter,  $q \gg p$ ,  $w > 2|\max(\mathbf{e})|$ .

**Decryption:** Given the ciphertext  $\mathbf{C}(\mathbf{V})$ , it can be decrypted using  $\mathbf{S}$  and  $w$  as  $\mathbf{V} = \lceil \frac{(\mathbf{S}\mathbf{C}(\mathbf{V}))^T}{w} \rceil_q$ .

**Inner Product:** Given two ciphertexts  $\mathbf{C}(\mathbf{V}_1), \mathbf{C}(\mathbf{V}_2)$  of  $\mathbf{V}_1, \mathbf{V}_2$ , and their corresponding secret keys  $\mathbf{S}_1$  and  $\mathbf{S}_2$ , the inner product operation of  $\mathbf{V}_1$  and  $\mathbf{V}_2$  over ciphertexts

can be performed as

$$vec(\mathbf{S}_1^T \mathbf{S}_2) \lceil \frac{vec(\mathbf{C}(\mathbf{V}_1) \mathbf{C}(\mathbf{V}_2)^T)}{w} \rceil_q = w \mathbf{V}_1 \mathbf{V}_2^T + \mathbf{e} \quad (3.2)$$

To this end,  $vec(\mathbf{S}_1^T \mathbf{S}_2)$  becomes the new secret key and  $\lceil \frac{vec(\mathbf{C}(\mathbf{V}_1) \mathbf{C}(\mathbf{V}_2)^T)}{w} \rceil_q$  becomes the new ciphertext of  $\mathbf{V}_1 \mathbf{V}_2^T$ .

More details about this IVE encryption algorithm and its security proof are available in ref [68].

## 3.4 Privacy-preserving Distance Comparison

### 3.4.1 Image Similarity Measurement

In CAPIA, similarity of images is measured by seven low-level color and texture feature vectors  $[\mathbf{V}_{i,RGB}, \mathbf{V}_{i,HSV}, \mathbf{V}_{i,LAB}, \mathbf{V}_{i,G}, \mathbf{V}_{i,GQ}, \mathbf{V}_{i,H}, \mathbf{V}_{i,HQ}]$ . Specifically, given two images  $I_a, I_b$ , their similarity can be computed as a combined distance

$$\begin{aligned} Dis_{ab} = & DL1_{ab}^{RGB} + DL1_{ab}^{HSV} + DL1_{ab}^G + DL1_{ab}^{GQ} \\ & + DL1_{ab}^H + DL1_{ab}^{HQ} + DKL_{ab}^{LAB} \end{aligned}$$

where  $DL1$  and  $DKL$  denote  $L_1$  distance and KL-Divergence of two vectors after data normalization. I consider these seven basic distances contribute equally to the total combined distance  $Dis_{ab}$ . Based on this observation, I first propose two privacy-preserving distance comparison solutions for  $L_1$  (namely,  $PL1C$ ) and KL-Divergence (namely,  $PKLC$ ) respectively, which support two key operations in CAPIA.

### 3.4.2 PL1C: Privacy-preserving $L_1$ Distance Comparison

In *PL1C*, I consider a user has three  $m$ -dimensional integer vectors  $\mathbf{V}_i, i \in \{a, b, c\}$  that will be outsourced to cloud after encryption. The cloud later compares  $L_1$  distances  $DL1_{ac}$  and  $DL1_{bc}$  directly over ciphertexts to figure out which one is smaller.

**Data Preparation:** Given a vector  $\mathbf{V}_i = [v_{i1}, \dots, v_{im}], i \in \{a, b, c\}$ , the user converts it to a  $m\beta$ -dimensional binary vector  $\tilde{\mathbf{V}}_i = [F(v_{i1}), \dots, F(v_{im})]$ , where  $\beta = |\max(\mathbf{V}_i)|$ , and  $F(v_{ij}) = [1, 1, \dots, 1, 0, \dots, 0]$  such that the first  $v_{ij}$  terms are 1 and the rest  $\beta - v_{ij}$  terms are 0. The  $L_1$  distance between  $\mathbf{V}_a$  and  $\mathbf{V}_b$  now can be calculated as

$$DL1_{ab} = \sum_{j=1}^m |v_{aj} - v_{bj}| = \sum_{j=1}^{m\beta} (\tilde{v}_{aj} - \tilde{v}_{bj})^2$$

Then, the user adopts an approximation method introduced in ref [69] to reduce the dimension of  $\tilde{\mathbf{V}}_i$  from  $m\beta$  to  $\hat{m} = \alpha m \log_{\gamma}^{\beta+1}$  based on the Johnson Lindenstrauss (JL) Lemma [70]. By denoting the approximated vector as  $\hat{\mathbf{V}}_i$ , I have  $DL1_{ab} = \sum_{j=1}^{m\beta} (\tilde{v}_{aj} - \tilde{v}_{bj})^2 \approx \sum_{j=1}^{\hat{m}} (\hat{v}_{aj} - \hat{v}_{bj})^2$ .

The correctness and accuracy of such an approximation have been proved in ref [69]. According to my experimental evaluation in Section 6.2, I set  $\alpha = 1$  and  $\gamma = 100$  in my scheme to achieve balanced accuracy and efficiency.

**Data Encryption:** Given an approximated vector  $\hat{\mathbf{V}}_i, i \in \{a, b\}$ , the user appends two elements to it as  $\hat{\mathbf{V}}_i = [\hat{v}_{i1}, \hat{v}_{i2}, \dots, \hat{v}_{i\hat{m}}, r - \frac{1}{2} \sum_{j=1}^{\hat{m}} \hat{v}_{ij}^2, \epsilon_i]$ , where  $r$  is a random number and  $\epsilon_i$  is a small random noise. Then, the user encrypts  $\hat{\mathbf{V}}_i$  using the **Encryption** algorithm of IVE presented in Section 3.3.2 as

$$\mathbf{C}_i = \mathbf{S}^{-1}(w\hat{\mathbf{V}}_i + \mathbf{e}_i)^T \quad (3.3)$$

where  $\mathbf{S}$  is the secret matrix,  $\mathbf{e}_i$  is an error vector, and  $w$  is an integer parameter.  $\mathbf{C}_a$ ,  $\mathbf{C}_b$ , and  $w$  are outsourced to the cloud.

**Request Generation:** Given the approximated vector  $\hat{\mathbf{V}}_c$ , the user selects a positive random number  $r_c$  and applies it to  $\hat{\mathbf{V}}_c$  as  $\hat{\mathbf{V}}_c = [r_c \hat{v}_{c1}, \dots, r_c \hat{v}_{c\hat{m}}, r_c, 1]$ .  $\hat{\mathbf{V}}_c$  is then encrypted as  $\mathbf{C}_c = \mathbf{S}_c^{-1}(w\hat{\mathbf{V}}_c + \mathbf{e}_c)^T$ , where  $\mathbf{S}_c$  is the secret key generated for  $\hat{\mathbf{V}}_c$ .  $\mathbf{C}_c$  and  $\mathbf{S}^T \mathbf{S}_c$  are sent to the cloud as request.

**Distance Comparison:** On receiving the request, the cloud computes  $\lceil \frac{vec(\mathbf{C}_a \mathbf{C}_c^T)}{w} \rceil_q$ ,  $\lceil \frac{vec(\mathbf{C}_b \mathbf{C}_c^T)}{w} \rceil_q$ , and decrypts them using  $vec(\mathbf{S}^T \mathbf{S}_c)$  to obtain  $\hat{\mathbf{V}}_a \hat{\mathbf{V}}_c^T$  and  $\hat{\mathbf{V}}_b \hat{\mathbf{V}}_c^T$  as Eq.3.2.

Finally, the approximated  $L_1$  distance comparison is performed as

$$\begin{aligned}
& \hat{\mathbf{V}}_b \hat{\mathbf{V}}_c^T - \hat{\mathbf{V}}_a \hat{\mathbf{V}}_c^T & (3.4) \\
& = r_c \sum_{j=1}^{\hat{m}} \hat{v}_{bj} \hat{v}_{cj} - \frac{r_c}{2} \sum_{j=1}^{\hat{m}} \hat{v}_{bj}^2 + r_c r + \epsilon_b \\
& \quad - (r_c \sum_{j=1}^{\hat{m}} \hat{v}_{aj} \hat{v}_{cj} - \frac{r_c}{2} \sum_{j=1}^{\hat{m}} \hat{v}_{aj}^2 + r_c r + \epsilon_a) \\
& = \frac{r_c}{2} \left( \sum_{j=1}^{\hat{m}} (\hat{v}_{aj} - \hat{v}_{cj})^2 - \sum_{j=1}^{\hat{m}} (\hat{v}_{bj} - \hat{v}_{cj})^2 \right) + (\epsilon_b - \epsilon_a) \\
& \approx \frac{r_c}{2} (DL1_{ac} - DL1_{bc}) + (\epsilon_b - \epsilon_a)
\end{aligned}$$

It is worth to note that  $PL1C$  is only interested in which distance is smaller during the comparison. Therefore, instead of letting the cloud get exact  $L_1$  distances for comparison,  $PL1C$  adopts approximated distance comparison result scaled and obfuscated by  $r_c$  and  $\epsilon_b - \epsilon_a$  as shown in Eq.3.4. As  $r_c$  is a positive random number, the sign of  $\frac{r_c}{2} (DL1_{ac} - DL1_{bc})$  is consistent with  $DL1_{ac} - DL1_{bc}$ . Meanwhile, since  $r_c \gg \epsilon_b - \epsilon_a$ , the added noise term has negligible influence to the sign of  $DL1_{ac} - DL1_{bc}$  unless these two distances are very close to each other. Fortunately, instead of finding the most

related one, my CAPIA design will utilize *PLIC* to figure out top 10 related candidates during the comparison. Such a design makes important candidates (say top 5 out of top 10) not be bypassed by the error introduced in  $\epsilon_b - \epsilon_a$ . This hypothesis is further validated by my experimental results in Section 6.2.

### 3.4.3 PKLC: Privacy-preserving KL-Divergence Comparison

In *PKLC*, I consider a user has three  $m$ -dimensional vectors  $\mathbf{V}_i, i \in \{a, b, c\}$ , and wants to outsource the comparison of  $DKL_{ac}$  and  $DKL_{bc}$  to the cloud without disclosing the content of  $\mathbf{V}_i, i \in \{a, b, c\}$ . The definition of KL-Divergence for two vectors  $\mathbf{V}_a, \mathbf{V}_b$  is:

$$\begin{aligned} DKL_{ab} &= \sum_{j=1}^m v_{aj} \times \log\left(\frac{v_{aj}}{v_{bj}}\right) \\ &= \sum_{j=1}^m v_{aj} \times \log(v_{aj}) - \sum_{j=1}^m v_{aj} \times \log(v_{bj}) \end{aligned} \quad (3.5)$$

where  $\log\left(\frac{v_{aj}}{v_{bj}}\right) = \log(v_{aj}) = \log(v_{bj}) = 0$  if  $v_{aj} = 0$  or  $v_{bj} = 0$ .

**Data Encryption:** The user first appends  $m + 2$  elements to  $\mathbf{V}_i, i \in \{a, b\}$  as  $\mathbf{V}_i = [v_{i1}, v_{i2}, \dots, v_{im}, v_{i1} \times \log(v_{i1}), \dots, v_{im} \times \log(v_{im}), r, \epsilon_i]$ , where  $r$  is a random number and  $\epsilon_i$  is a small random noise. Then,  $\mathbf{V}_i, i \in \{a, b\}$  are encrypted with the **Encryption** algorithm of IVE as

$$\mathbf{C}_i = \mathbf{S}^{-1}(w\mathbf{V}_i + \mathbf{e}_i)^T \quad (3.6)$$

$\mathbf{C}_a$  and  $\mathbf{C}_b$  are outsourced to the cloud.

**Request Generation:** The user processes  $\mathbf{V}_c$  to generate a privacy-preserving KL-Divergence comparison request as

- Replace elements  $v_{cj}$  with  $-r_c \times \log(v_{cj})$ , and append  $m+2$  elements to  $\mathbf{V}_c$  as  $\mathbf{V}_c = [-r_c \times \log(v_{c1}), \dots, -r_c \times \log(v_{cm}), F(v_{c1}), \dots, F(v_{cm}), r_c, -1]$ , where  $F(v_{cj}) = \begin{cases} r_c, v_{cj} \neq 0 \\ 0, v_{cj} = 0 \end{cases}$ ,  $r_c$  is a positive random number changing for every request.
- Encrypt  $\mathbf{V}_c$  as  $\mathbf{C}_c$  using the **Encryption** algorithm of IVE as  $\mathbf{C}_c = \mathbf{S}_c^{-1}(w\mathbf{V}_c + \mathbf{e}_c)^T$ .

$\mathbf{C}_c$  and  $\mathbf{S}^T \mathbf{S}_c$  are sent to the cloud as request.

**Distance Comparison:** On receiving the request, the cloud first computes  $\lceil \frac{vec(\mathbf{C}_a \mathbf{C}_a^T)}{w} \rceil_q$ ,  $\lceil \frac{vec(\mathbf{C}_b \mathbf{C}_b^T)}{w} \rceil_q$  and decrypts them using  $vec(\mathbf{S}^T \mathbf{S}_c)$  to get  $\mathbf{V}_a \mathbf{V}_c^T$  and  $\mathbf{V}_b \mathbf{V}_c^T$  as Eq.3.2.

Then, the cloud compares  $DKL_{ac}$  and  $DKL_{bc}$  by computing

$$\begin{aligned}
 & \mathbf{V}_a \mathbf{V}_c^T - \mathbf{V}_b \mathbf{V}_c^T & (3.7) \\
 = & r_c(r + \sum_{j=1}^m v_{aj} \times \log(v_{aj}) - \sum_{j=1}^m v_{aj} \times \log(v_{cj})) - \epsilon_a \\
 - & r_c(r + \sum_{j=1}^m v_{bj} \times \log(v_{bj}) - \sum_{j=1}^m v_{bj} \times \log(v_{cj})) + \epsilon_b \\
 = & r_c(DKL_{ac} - DKL_{bc}) + (\epsilon_b - \epsilon_a)
 \end{aligned}$$

Similar to my *PL1C* construction, I have  $r_c > 0$  and  $r_c \gg (\epsilon_b - \epsilon_a)$ . Therefore, the cloud can figure out which KL-Divergence is smaller based on the scaled and obfuscated comparison result.

### 3.5 Cloud Assisted Privacy-preserving Image Annotation

After the introduction of my privacy-preserving distance comparison design, in this section, I illustrate CAPIA by integrating *PL1C* and *PKLC* in image annotation task with cloud deployment. My designs consists of five major procedures. In the *System*

*Setup*, the user selects system parameters, extracts and pre-processes feature vectors of images in a pre-annotated dataset. Then, the user executes the *Data Encryption* procedure to encrypt these processed feature vectors. Both the *System Setup* procedure and the *Data Encryption* procedure are one-time cost in CAPIA. Later on, the user can use the *Secure Annotation Request* procedure to generate an encrypted annotation request. On receiving the request, the cloud server performs the *Privacy-preserving Annotation on Cloud* procedure to return encrypted keywords for the requested image. At the end, the user obtains final keywords by executing the *Final Keyword Selection* procedure.

### 3.5.1 System Setup

To perform the one-time setup of CAPIA system, the user first prepares a pre-annotated image dataset with  $n$  images, which can be obtained from public sources, such as IAPR TC-12 [71], LabelMe [72], etc. For each image  $I_i$  in the dataset, the user extracts seven feature vectors  $[\mathbf{V}_{i,RGB}, \mathbf{V}_{i,HSV}, \mathbf{V}_{i,LAB}, \mathbf{V}_{i,G}, \mathbf{V}_{i,GQ}, \mathbf{V}_{i,H}, \mathbf{V}_{i,HQ}]$ . Compared with other five feature vectors that have dimension up to 256,  $\mathbf{V}_{i,H}$  and  $\mathbf{V}_{i,HQ}$  have a high dimension as 4096. To guarantee the efficiency while processing feature vectors, Principal Component Analysis (PCA) [73] is utilized to reduce the dimension of  $\mathbf{V}_{i,H}$  and  $\mathbf{V}_{i,HQ}$ . According to my experimental evaluation in Section 4.3.2, PCA based dimension reduction with proper setting can significantly improve the efficiency of CAPIA with slight accuracy loss. After that,  $L_1$  normalization will be performed for each feature vector, which normalizes elements in these vectors to  $[-1,1]$ . Besides  $\mathbf{V}_{i,LAB}$ , the user also increases each element in  $\mathbf{V}_{i,k}, k \in \{RGB, HSV, G, GQ, H, HQ\}$  as  $v_{i,k,j} = v_{i,k,j} + 1$  to avoid negative values. Next, each element in all feature vectors are scaled by the same value. Given three processed vectors  $\mathbf{V}_i, i \in \{a, b, c\}$ , it is



easy to verify that the sign of  $L_1$  distance comparison result  $DL1_{ab} - DL1_{ac}$  and KL-Divergence comparison result  $DKL_{ab} - DKL_{ac}$  with processed vectors remain the same as that using original vectors. Six feature vectors that use  $L_1$  distance for similarity measurement are concatenated as a  $m_{L1}$ -dimensional vector  $\mathbf{V}_{i,L1}$ .  $\mathbf{V}_{i,LAB}$  is denoted as a  $m_{KL}$ -dimensional vector  $\mathbf{V}_{i,KL}$  for expression simplicity. It is easy to verify that  $DL1_{ab}^{L1} = DL1_{ab}^{RGB} + DL1_{ab}^{HSV} + DL1_{ab}^G + DL1_{ab}^{GQ} + DL1_{ab}^H + DL1_{ab}^{HQ}$ .

### 3.5.2 Dataset Encryption

Given an image  $I_i$  in the pre-annotated dataset, its keywords  $\{K_{i,t}\}$  are first encrypted using AES. Then, feature vectors  $\mathbf{V}_{i,L1}$  and  $\mathbf{V}_{i,KL}$  are encrypted as  $\mathbf{C}_{i,L1}$  and  $\mathbf{C}_{i,KL}$  using the *Data Encryption* methods in my proposed *PL1C* and *PKLC* schemes respectively. During the encryption, same secret keys  $\mathbf{S}_{L1}$ ,  $\mathbf{S}_{KL}$ , public parameter  $w$ , and random number  $r$  will be used for all images. However, different error vector  $\mathbf{e}_i$  and noise term  $\epsilon_i$  are generated for each image  $I_i$  correspondingly. The user also computes  $\mathbf{S}_{L1}^T \mathbf{S}_{s,L1}$  and  $\mathbf{S}_{KL}^T \mathbf{S}_{s,KL}$ , in which  $\mathbf{S}_{s,L1}$  and  $\mathbf{S}_{s,KL}$  are secret keys for the encryption of later annotation requests. These  $\mathbf{C}_{i,L1}$ ,  $\mathbf{C}_{i,KL}$  and encrypted keywords of each image  $I_i$ , as well as  $\mathbf{S}_{L1}^T \mathbf{S}_{s,L1}$  and  $\mathbf{S}_{KL}^T \mathbf{S}_{s,KL}$  are outsourced to the cloud.

### 3.5.3 Secure Annotation Request

When the user has a new image  $I_s$  for annotation, he/she first extracts seven feature vectors as  $\mathbf{V}_s, s \in [RGB, HSV, LAB, G, GQ, H, HQ]$ . These vectors will be normalized and scaled to output  $\mathbf{V}_{s,L1}$  and  $\mathbf{V}_{s,KL}$  as that in the *System Setup* procedure. Then, the user processes and encrypts  $\mathbf{V}_{s,L1}$  and  $\mathbf{V}_{s,KL}$  as  $\mathbf{C}_{s,L1}$  and  $\mathbf{C}_{s,KL}$  using the *Request Generation* methods in my *PL1C* and *PKLC* schemes respectively. For each annotation

request, the user generates a new positive random number  $r_s$  and a new error vector  $\mathbf{e}_s$ .

$\mathbf{C}_{s,L1}$  and  $\mathbf{C}_{s,KL}$  are sent to the cloud as the annotation request.

### 3.5.4 Privacy-preserving Annotation on Cloud

On receiving the request, the cloud first outputs  $\mathbf{V}_{i,L1}\mathbf{V}_{s,L1}^T$  and  $\mathbf{V}_{i,KL}\mathbf{V}_{s,KL}^T$  for each image in the pre-annotated dataset as

$$\mathbf{V}_{i,L1}\mathbf{V}_{s,L1}^T = \text{vec}(\mathbf{S}_{L1}^T\mathbf{S}_{s,L1}) \left\lceil \frac{\text{vec}(\mathbf{C}_{i,L1}\mathbf{C}_{s,L1}^T)^T}{w} \right\rceil_q \quad (3.8)$$

$$\mathbf{V}_{i,KL}\mathbf{V}_{s,KL}^T = \text{vec}(\mathbf{S}_{KL}^T\mathbf{S}_{s,KL}) \left\lceil \frac{\text{vec}(\mathbf{C}_{i,KL}\mathbf{C}_{s,KL}^T)^T}{w} \right\rceil_q \quad (3.9)$$

where  $1 \leq i \leq n$ . Then, the cloud ranks all the images according to their combined distances to the request image  $I_s$ . Specifically, a distance comparison candidate  $Comp_i = -2(\mathbf{V}_{i,L1}\mathbf{V}_{s,L1}^T) + \mathbf{V}_{i,KL}\mathbf{V}_{s,KL}^T$  can be generated for each image  $I_i$ . Given  $I_a$  and  $I_b$  for example, the cloud can rank them as

$$\begin{aligned} & Comp_a - Comp_b & (3.10) \\ & = 2(\mathbf{V}_{b,L1}\mathbf{V}_{s,L1}^T - \mathbf{V}_{a,L1}\mathbf{V}_{s,L1}^T) \\ & + \mathbf{V}_{a,L1}\mathbf{V}_{s,KL}^T - \mathbf{V}_{b,KL}\mathbf{V}_{s,KL}^T \\ & = r_s(DL1_{as}^{L1} - DL1_{bs}^{L1}) + 2(\epsilon_b - \epsilon_a) \\ & + r_s(DKL_{as}^{LAB} - DKL_{bs}^{LAB}) + (\epsilon_b - \epsilon_a) \\ & = r_s(Dis_{as} - Dis_{bs}) + 3(\epsilon_b - \epsilon_a) \end{aligned}$$

As  $r_s$  is a positive value and  $r_s \gg (\epsilon_b - \epsilon_a)$ , the cloud can figure out which image is more relative to  $I_s$  according to the above distance comparison result. According to the ranking of all pre-annotated images, the cloud outputs top related images to  $I_s$  and denotes them as a set  $RST$ . Finally, the cloud returns distance comparison candidates  $Comp_i, i \in RST$  as well as corresponding encrypted keywords back to the user.

### 3.5.5 Final Keyword Selection

In this stage, the user first decrypts encrypted keywords and obtains  $K_{i,t}, i \in RST$ , where  $K_{i,t}$  is the  $t$ -th pre-annotated keyword in image  $I_i$ . Then, the user computes distances  $Dis_{is}, i \in RST$  as

$$\begin{aligned} Dis_{is} &= (2r + \sum_{j=1}^{m_{L1}} v_{s,L1,j}^2) + \frac{Comp_i}{r_s} \\ &= (2r + \sum_{j=1}^{m_{L1}} v_{s,L1,j}^2) + \frac{-2(\mathbf{V}_{i,L1} \mathbf{V}_{s,L1}^T) + \mathbf{V}_{i,KL} \mathbf{V}_{s,KL}^T}{r_s} \end{aligned}$$

To achieve higher accuracy in keywords selection, I consider that keywords in images that have smaller distance to the requested one are more relevant. Thus, I define a real-time weight  $W_t$  for each keyword based on distances  $Dis_{is}$  as

$$W_{I_i} = 1 - \frac{Dis_{is}}{\sum_{i \in RST} Dis_{is}} \quad (3.11)$$

$$W_t = \sum W_{I_i}, \text{ for } I_i \text{ contains } K_{i,t} \quad (3.12)$$

Specifically, I first figure out the weight  $W_{I_i}$  of each image according to their distance based similarity. As my definition in Eq.3.12, images with smaller distance will receive a

larger weight value. Then, considering the same keyword can appear in multiple images, the final weight  $W_t$  of a keyword  $K_{i,t}$  is generated by adding weights of images that contain this keyword. Finally, the user selects keywords for his/her image according to their ranking of weight  $W_t$ .

### 3.6 Preliminaries of CPAR

In CAPIA, for every single annotation request, linear processing of all encrypted records in a large-scale dataset is required, which hence becomes the performance bottleneck for practical usage. In order to bypass the latency brought by this linear processing strategy, I leverage randomized k-dimension forest (RKDF), a member from space partitioning tree family, as the parallel search index to boost CAPIA's performance in terms of efficiency. Different from many other index structures that are only efficient for low-dimensional data, RKDF is featured by its performance in handling high-dimensional data. In CAPIA, data vectors are over 1300-dimension and thus making RKDF an effective selection. To understand how the randomized k-dimension forest works, I first investigate its basic component, k-dimension tree (k-d tree), a space partitioning tree structure which boosts up nearest vector search speed [74]. I tailored it in my construction to achieve efficient vector search in privacy-preserving manner. Due to the top-down traversal algorithm underlying k-d trees, several single-element comparisons between a few vectors are required. In order to protect the privacy of these single elements, order-preserving encryption (OPE) is applied in CPAR.

### 3.6.1 k-dimension Tree

A k-d tree, is a data structure used for organizing some number of points in a space with k dimensions. It is a binary search tree with other constraints imposed on it. k-d trees are very useful for range and nearest neighbor searches. Each level of a k-d tree splits all children along a specific dimension, using a hyperplane that is perpendicular to the corresponding axis. At the root of the tree all children will be split based on the first dimension. Each level down in the tree divides on the dimension with the highest deviation. This procedure is performed recursively on both the left and right sub-trees until the max trees are only composed of one element. More details about this k-dimension tree structure and its evaluation are available in ref [74].

### 3.6.2 Order-preserving Encryption

Order-preserving symmetric encryption is a deterministic encryption scheme whose encryption function preserves numerical ordering of the plaintexts. Given two integers  $a$  and  $b$  in which  $a < b$ , by encrypting with OPE, the order of  $a$  and  $b$  is preserved as  $OPE(a) < OPE(b)$ . More details about this OPE encryption scheme and its security proof are available in ref [75, 76].

## 3.7 Privacy-preserving Distance Comparison with Randomized k-d Forest

In this section, I first introduce the top-down traversal and back trace search in an unprotected randomized k-d forest. In addition, in order to address the privacy concerns

when integrating the randomized k-d forest in the automatic image annotation, I propose two privacy-preserving distance comparison scheme with RKDF.

### 3.7.1 Randomized k-d Forest Search

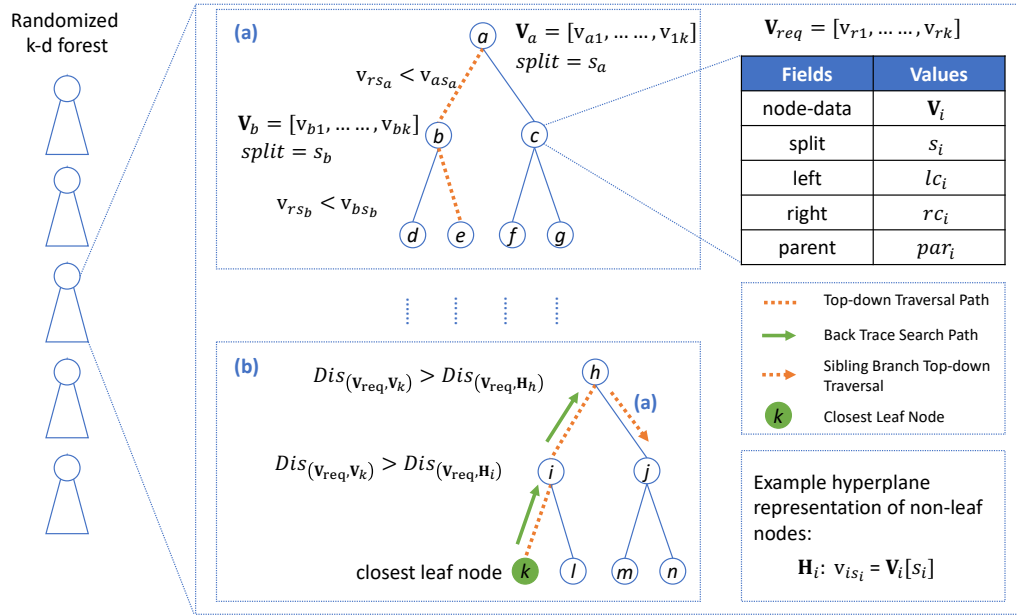


FIGURE 3.1: Randomized k-d Forest

$\mathbf{V}_{req}$  is the request vector and each  $\mathbf{V}_i$  is stored in each tree node  $i$ .  $\text{Dis}(\cdot)$  is an arbitrary distance calculation function and  $\text{Dis}(\mathbf{V}_{req}, \mathbf{H}_i)$  is the distance between the request vector  $\mathbf{V}_{req}$  and  $\text{Node}_i$ 's hyperplane.  $\mathbf{V}_{qL}$  is the least closest vector to  $\mathbf{V}_{req}$  in priority queue  $Queue$ . (a) represents top-down traversal; (b) represents back trace search and (c) represents queue push process.

As depicted in Figure 3.1, a RKDF is composed of a set of parallel k-d trees. For each  $\text{Node}_i$  in a k-d tree [74], it stores a feature vector  $\mathbf{V}_i$  of dataset image  $I_i$ . In addition, each non-leaf node also stores a *split* field  $s_i$  to generate a hyperplane that divides the vector space into two parts. Each  $\text{Node}_j$  in left sub-tree of  $\text{Node}_i$  has  $\text{Node}_j[s_i] \leq \text{Node}_i[s_i]$  and vice versa, as described in ref [74]. To search nodes that store vectors with top-smallest distances to a request vector  $\mathbf{V}_{req}$ , a parallel search among all trees in the forest is performed. Specifically, each tree is traversed in a top-down manner by comparing the *split* field values of  $\mathbf{V}_{req}$  and the vector  $\mathbf{V}_i$  stored in each  $\text{Node}_i$  as an example shown in Fig.3.1(a). The traversal selects the left branch to continue if  $\mathbf{V}_{req}[s_i] \leq \mathbf{V}_i[s_i]$  and vice

versa. Once the traversal reaches a leaf node, the vector stored in that leaf node is pushed into a priority queue *Queue* as a current close candidate to  $\mathbf{V}_{req}$ . The queue push process is shown in Fig.3.1(c). Note that during the search process, this *Queue* keeps updating to hold  $L$  closest vectors to  $\mathbf{V}_{req}$  and is shared by all trees in the forest. After that, a back trace search starts by iterating all the nodes in the path from the parent of the current node to the root node as an example shown in Fig.3.1(b). When reaching a  $Node_i$  during the back trace, a same queue push is executed to judge whether to add  $Node_i$  to *Queue* as illustrated in Fig.3.1(c). For each  $Node_i$  in this path, a distance comparison between  $Dis(\mathbf{V}_{req}, \mathbf{H}_i)$  and  $Dis(\mathbf{V}_{req}, \mathbf{V}_{qL})$  is compared, where  $Dis(\mathbf{V}_{req}, \mathbf{H}_i)$  is the distance between  $\mathbf{V}_{req}$  and a  $Node_i$ 's hyperplane.  $\mathbf{H}_i$  can be considered as the projection vector of  $\mathbf{V}_{req}$  on  $Node_i$ 's hyperplane.  $\mathbf{V}_{qL}$  is the  $L$ th vector in *Queue* which meets  $Dis(\mathbf{V}_{req}, \mathbf{V}_{qi}) \leq Dis(\mathbf{V}_{req}, \mathbf{V}_{qL}), \forall \mathbf{V}_{qi} \in Queue$ . If  $Dis(\mathbf{V}_{req}, \mathbf{H}_i) > Dis(\mathbf{V}_{req}, \mathbf{V}_{qL})$ , the back trace continues to the next node in this path. Otherwise, the sibling branch of  $Node_i$  needs to be searched using the top-down traversal. In RKDF, once a node has been searched in one k-d tree, it will be marked and does not need to be checked again in the other trees. To further enhance the search efficiency of a RKDF, approximated search strategy can be adopted. In particular, based on the hypothesis that feature vectors of similar images are likely to be grouped in the same branch, there is a high probability that the targeted optimal top similar vectors will be visited well before visiting all nodes in each k-d tree. In Section 6.2, I will evaluate the relationship among the approximation strength, accuracy, and efficiency. The detailed search of a RKDF is provided in Algorithm 1. For more details about the RKDF, please refer to ref [59].

To protect the privacy of user's data during the cloud-based annotation, the image data associated with the RKDF need to be encrypted. Furthermore, these encrypted data shall support corresponding search operations in RKDF, which include:

- The comparison between  $\mathbf{V}_{req}[s_i]$  and  $\mathbf{V}_i[s_i]$  in the top-down traversal for path selection.
- The comparison between  $Dis(\mathbf{V}_{req}, \mathbf{H}_i)$  and  $Dis(\mathbf{V}_{req}, \mathbf{V}_{qL})$  during the back trace process.
- The comparison between  $Dis(\mathbf{V}_{req}, \mathbf{V}_a)$  and  $Dis(\mathbf{V}_{req}, \mathbf{V}_b)$ , i.e., distances from the request vector to two different images' feature vectors, which is used in the queue push process.

In order to support these operations, a challenge needs to be resolved: The original privacy-preserving comparison scheme for  $L_1$  distance (*PL1C*) and KL-Divergence (*PKLC*) in CAPIA cannot be simply re-used in CPAR. That's because *PL1C* and *PKLC* can only support the privacy-preserving distance comparison between two vectors. However, while searching in a RKDF, the distance comparison between a vector and a hyperplane needs to be supported in the back trace process and queue push process of RKDF. In order to resolve this issue, I re-design *PL1C* and *PKLC* to get *PL1C - RF* and *PKLC - RF*, standing for *PL1C* and *PKLC* for RKDF. *PL1C - RF* and *PKLC - RF* enable the aforementioned privacy-preserving distance comparison between two vectors as well as between one vector and one hyperplane. In addition, I integrate order-preserving encryption [75, 76] into CPAR to protect the comparison of *split* field values in the top-down traversal of RKDF.

### 3.7.2 *PL1C - RF*: Privacy-preserving $L_1$ Distance Comparison for Randomized k-d Forest

In *PL1C - RF*, I consider two types of  $L_1$  distance comparison that are required in the queue push and back trace process of RKDF: 1)  $DL1_{ac}$  and  $DL1_{bc}$  for three image



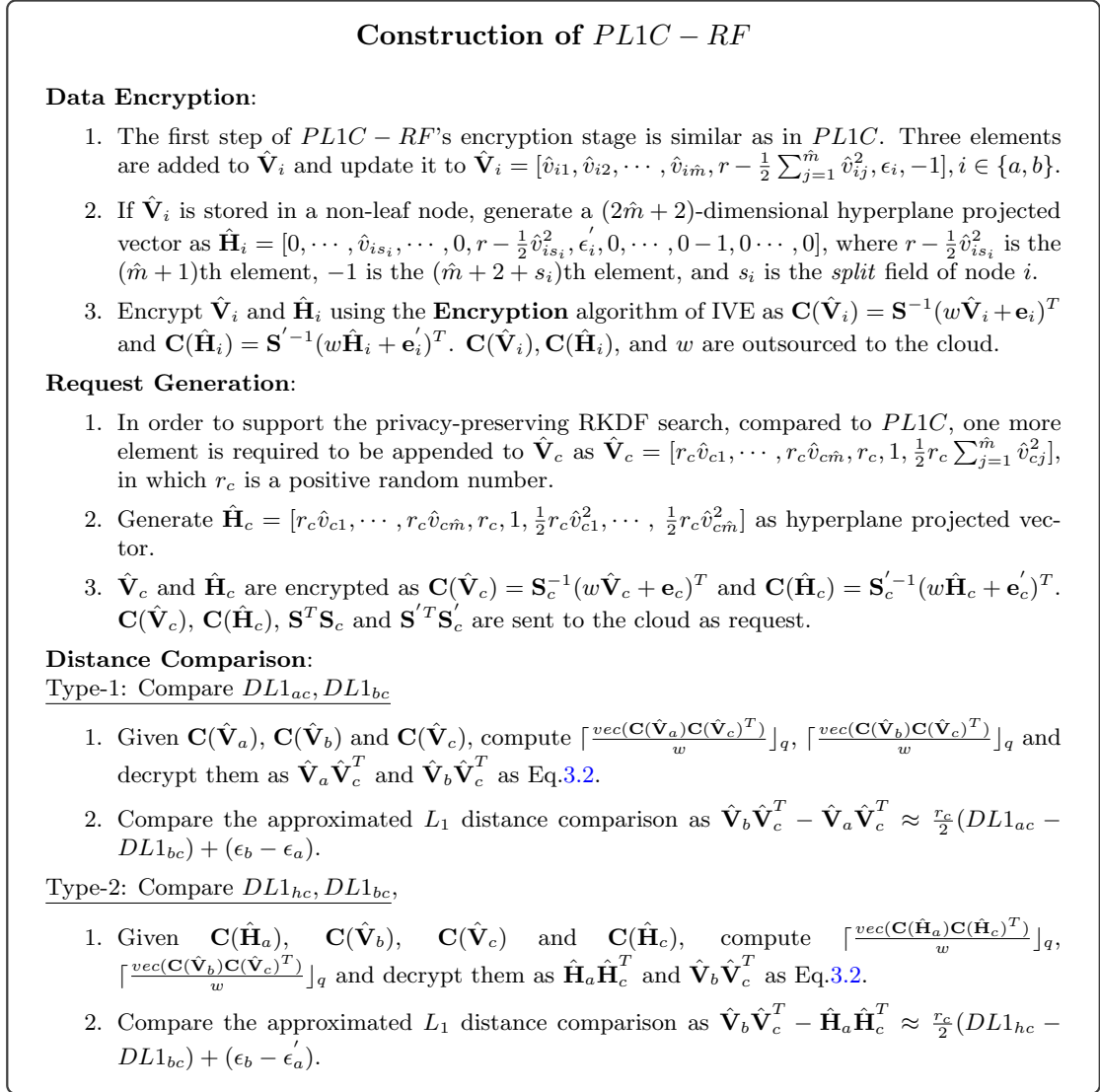
feature vectors  $\mathbf{V}_i, i \in \{a, b, c\}$ ; 2)  $DL1_{hc}$  and  $DL1_{bc}$  for a hyperplane projected vector  $\mathbf{H}_a$  and two image feature vectors  $\mathbf{V}_b, \mathbf{V}_c$ .  $DL1_{hc}$  is measured by the  $L_1$  distance between  $\mathbf{H}_a[s_a]$  and  $\mathbf{V}_c[s_a]$ , where  $s_a$  is the *split* field of the  $Node_a$ . To be more specific,  $DL1_{hc}$  is calculated by projecting  $\mathbf{V}_c$  on  $Node_a$ 's hyperplane and then calculating the  $L_1$  distance between  $\mathbf{V}_c$  and the projected vector  $\mathbf{H}_a$ .

**Data Preparation:** The data preparation for  $PL1C - RF$  is the same as  $PL1C$  for the reason that both of them use the same local descriptor in [22] and the JL-Lemma based approximation in [70]. Given three  $m$ -dimensional integer vectors  $\mathbf{V}_i, i \in \{a, b, c\}$ , an approximated vector  $\hat{\mathbf{V}}_i = [\hat{v}_{i1}, \hat{v}_{i2}, \dots, \hat{v}_{im}], i \in \{a, b\}$  is generated using JL-Lemma. The detailed construction of the rest stages in  $PL1C-RF$  is presented in Fig.3.2. The user first encrypts the image feature vectors and its corresponding hyperplane projected vector (if exists), and then stores them in the cloud. Later on the user can generate encrypted  $L_1$  distance comparison request and ask the cloud to conduct privacy-preserving comparison. On receiving the request, the cloud can conduct two types of  $L_1$  distance comparison using ciphertext only according to user's request.

Same as  $PL1C$ ,  $PL1C - RF$  maintains its vector privacy protection by keeping the approximated distance comparison result scaled and obfuscated by  $r_c, \epsilon_b - \epsilon_a$  and  $\epsilon_b - \epsilon'_a$  as shown in 3.7.2.

### 3.7.3 $PKLC - RF$ : Privacy-preserving KL-Divergence Comparison for Randomized k-d Forest

In  $PKLC - RF$ , I also consider two types of KL-Divergence comparison similar to  $PL1C - RF$ : 1)  $DKL_{ac}$  and  $DKL_{bc}$  for three image feature vectors  $\mathbf{V}_i, i \in \{a, b, c\}$ ; 2)  $DKL_{hc}$  and  $DKL_{bc}$  for a hyperplane projected vector  $\mathbf{H}_a$  and two image feature vectors

FIGURE 3.2: Construction of  $PL1C - RF$ 

$\mathbf{V}_b, \mathbf{V}_c$ . In addition, the KL-Divergence  $DKL_{hc}$  between a image feature vector and a hyperplane is measured by the KL-Divergence between  $\mathbf{H}_a[s_a]$  and  $\mathbf{V}_c[s_a]$ , where  $s_a$  is the *split* field of  $Node_a$ . Similar as  $PL1C - RF$ ,  $PKLC - RF$  is also calculated by projecting  $\mathbf{V}_c$  on  $Node_a$ 's hyperplane and then calculating the KL-Divergence between  $\mathbf{V}_c$  and the projected vector  $\mathbf{H}_a$ .

The detailed construction of  $PKLC - RF$  is presented in Fig.3.3. In the data encryption stage, the image feature vectors and corresponding hyperplane projected vector (if exists) are encrypted and stored in the cloud. On receiving the encrypted KL-Divergence

comparison request from the user, the cloud conducts two types of privacy-preserving KL-Divergence comparison using ciphertext only according to user's request. Similar to my *PL1C* construction, I have  $r_c > 0$  and  $r_c \gg (\epsilon_b - \epsilon_a)$ . Therefore, the cloud can figure out which KL-Divergence is smaller based on the scaled and obfuscated comparison result.

### Construction of *PKLC* – *RF*

#### Data Encryption:

1. As *PKLC*, *PKLC* – *RF* appends  $m + 2$  elements as  $\mathbf{V}_i = [v_{i1}, v_{i2}, \dots, v_{im}, v_{i1} \times \log(v_{i1}), \dots, v_{im} \times \log(v_{im}), r, \epsilon_i]$ , where  $r$  is a random number and  $\epsilon_i$  is a small random noise. If  $\mathbf{V}_i$  is stored in a non-leaf node in RKDF, its corresponding hyperplane projected vector is processed as  $\mathbf{H}_i = [0, \dots, v_{is_i}, \dots, 0, \dots, v_{is_i} \times \log(v_{is_i}), \dots, 0, r, \epsilon'_i]$ , where  $s_i$  is the *split* field of the node,  $v_{is_i}$ ,  $v_{is_i} \times \log(v_{is_i})$  and  $r$  are the  $s_i$ th,  $(m + s_i)$ th and  $(2m + 1)$ th elements respectively.
2. Encrypt  $\mathbf{V}_i$  and  $\mathbf{H}_i$  with the **Encryption** algorithm of IVE as  $\mathbf{C}(\mathbf{V}_i) = \mathbf{S}^{-1}(w\mathbf{V}_i + \mathbf{e}_i)^T$  and  $\mathbf{C}(\mathbf{H}_i) = \mathbf{S}^{-1}(w\mathbf{H}_i + \mathbf{e}'_i)^T$ .

#### Request Generation:

1. The request generation stage is exactly the same as in *PKLC*. Unlike *PL1C* – *RF* which needs to add one more element for the  $L_1$  distance comparison between a vector and a hyperplane, the data encryption stage in *PKLC* – *RF* does not append additional element compared with *PKLC*. Ciphertext  $\mathbf{C}(\mathbf{V}_c)$  and key  $\mathbf{S}^T \mathbf{S}_c$  are sent to the cloud as request.

#### KL-Divergence Comparison:

Type-1: Compare  $DKL_{ac}, DKL_{bc}$

1. Compute  $\lceil \frac{vec(\mathbf{C}(\mathbf{V}_a)\mathbf{C}(\mathbf{V}_c)^T)}{w} \rceil_q$ ,  $\lceil \frac{vec(\mathbf{C}(\mathbf{V}_b)\mathbf{C}(\mathbf{V}_c)^T)}{w} \rceil_q$  and decrypts them as  $\mathbf{V}_a \mathbf{V}_c^T$  and  $\mathbf{V}_b \mathbf{V}_c^T$  using the **Decryption** of IVE in Section 3.3.2.
2. Compare KL divergence as  $\mathbf{V}_a \mathbf{V}_c^T - \mathbf{V}_b \mathbf{V}_c^T = r_c(DKL_{ac} - DKL_{bc}) + (\epsilon_b - \epsilon_a)$ .

Type-2: Compare  $DKL_{hc}, DKL_{bc}$

1. Compute  $\lceil \frac{vec(\mathbf{C}(\mathbf{H}_a)\mathbf{C}(\mathbf{V}_c)^T)}{w} \rceil_q$ ,  $\lceil \frac{vec(\mathbf{C}(\mathbf{V}_b)\mathbf{C}(\mathbf{V}_c)^T)}{w} \rceil_q$  and decrypts as  $\mathbf{H}_a \mathbf{V}_c^T$  and  $\mathbf{V}_b \mathbf{V}_c^T$  using the **Decryption** of IVE as Eq.3.2.
2. Compare KL divergence as  $\mathbf{H}_a \mathbf{V}_c^T - \mathbf{V}_b \mathbf{V}_c^T = r_c(DKL_{hc} - DKL_{bc}) + (\epsilon_b - \epsilon'_a)$ .

FIGURE 3.3: Construction of *PKLC* – *RF*

## 3.8 Cloud-Assisted Privacy-preserving Image Annotation with Randomized k-d Forest

### 3.8.1 Detailed Construction of CPAR

In this section, I show the detail construction of *CPAR* by adding *PLIC – RF* and *PKLC – RF* to boost the similar images search process during the image annotation task. *CPAR* shares a similar high-level major procedures with *CAPIA*. In the *System Setup*, in addition to system parameters selection, image feature extraction and pre-processing, the user also uses these feature vectors to build a *RKDF*. Then, the user executes the *RKDF Encryption* procedure to encrypt all data associated with nodes in the *RKDF*. Both the *System Setup* procedure and the *RKDF Encryption* procedure are one-time cost in *CPAR*. Then, *Secure Annotation Request* is performed to generate an encrypted annotation request. Followed by that, *Privacy-preserving Annotation on Cloud* and *Final Keyword Selection* find the user a group of top similar images and help the user determine the final keyword set for the requested image.

#### 3.8.1.1 System Setup

The first few steps in system setup procedure follows the same path as *CAPIA* in Section 3.5.1.  $m_{L1}$ -dimensional vector  $\mathbf{V}_{i,L1}$  and  $m_{KL}$ -dimensional vector  $\mathbf{V}_{i,KL}$  are extracted for each image from a pre-defined dataset. After that, a *RKDF* is constructed with feature vector space  $\{\mathbf{V}_i\}_{1 \leq i \leq n}$ , in which each node in a single tree is associated with one  $\mathbf{V}_i$ . For each non-leaf node in *RKDF*, its *split* field element  $\mathbf{V}_i[s_i]$  is stored in a set  $\mathcal{SF}$ . In *CPAR*, the *RKDF* contains ten parallel k-d trees.

### 3.8.1.2 RKDF Encryption

In this stage, CPAR first executes the same process as in Section 3.5.2 to encrypt keywords  $\{K_{i,t}\}$  by AES and get the encrypted feature vectors  $\mathbf{C}(\mathbf{V}_{i,L1})$  and  $\mathbf{C}(\mathbf{V}_{i,KL})$ .  $\mathbf{C}(\mathbf{V}_{i,L1})$  and  $\mathbf{C}(\mathbf{V}_{i,KL})$  are then stored in the corresponding  $Node_i$  of the RKDF. For each non-leaf node, encrypted hyperplane projected vectors  $\mathbf{C}(\mathbf{H}_{i,L1}), \mathbf{C}(\mathbf{H}_{i,KL})$  are generated and added into  $Node_i$  using the data encryption processes described in my *PL1C-RF* and *PKLC-RF*. In addition, for the *split* field element  $\mathbf{V}_i[s_i]$  of each non-leaf node, an order-preserving encryption is executed and the ciphertext  $OPE(\mathbf{V}_i[s_i])$  is stored in  $Node_i$ . After the encryption, each node in the RKDF only contains encrypted data as

- **Non-leaf Node:**  $[\mathbf{C}(\mathbf{V}_{i,L1}), \mathbf{C}(\mathbf{V}_{i,KL}), \mathbf{C}(\mathbf{H}_{i,L1}), \mathbf{C}(\mathbf{H}_{i,KL}), OPE(\mathbf{V}_i[s_i]), AES(\{K_{i,t}\})]$
- **Leaf Node:**  $[\mathbf{C}(\mathbf{V}_{i,L1}), \mathbf{C}(\mathbf{V}_{i,KL}), AES(\{K_{i,t}\})]$

During the encryption process, same secret keys  $\mathbf{S}_{L1}, \mathbf{S}'_{L1}, \mathbf{S}_{KL}$ , public parameter  $w$ , and random number  $r$  will be used for all images. However, different error vector  $\mathbf{e}_i, \mathbf{e}'_i$  and noise term  $\epsilon_i, \epsilon'_i$  are generated for each image  $I_i$  correspondingly. The user also computes  $\mathbf{S}_{L1}^T \mathbf{S}_{s,L1}, \mathbf{S}'_{L1}{}^T \mathbf{S}'_{s,L1}$  and  $\mathbf{S}_{KL}^T \mathbf{S}_{s,KL}$ , in which  $\mathbf{S}_{s,L1}, \mathbf{S}'_{s,L1}$  and  $\mathbf{S}_{s,KL}$  are secret keys for the encryption of later annotation requests. The encrypted RKDF,  $\mathbf{S}_{L1}^T \mathbf{S}_{s,L1}, \mathbf{S}'_{L1}{}^T \mathbf{S}'_{s,L1}$  and  $\mathbf{S}_{KL}^T \mathbf{S}_{s,KL}$  are outsourced to the cloud.

### 3.8.1.3 Secure Annotation Request

CPAR shares the same process for the secure annotation request with CAPIA as stated in 3.5.3. The user can generate the features of the image to be annotated and then encrypt them as  $\mathbf{C}(\mathbf{V}_{s,L1}), \mathbf{C}(\mathbf{H}_{s,L1}),$  and  $\mathbf{C}(\mathbf{V}_{s,KL})$  using the *Request Generation* of *PL1C-RF*

and *PKLC – RF* schemes respectively. For each annotation request, the user generates a new positive random number  $r_s$  and new error vectors  $\mathbf{e}_s, \mathbf{e}'_s$ . Meanwhile, for each element  $sf_j$  in the *split* field set  $\mathcal{SF}$  generated in *System Setup*, the user encrypts  $\mathbf{V}_s[sf_j]$  using order-preserving encryption as  $OPE(\mathbf{V}_s[sf_j])$ .  $\mathbf{C}(\mathbf{V}_{s,L1}), \mathbf{C}(\mathbf{H}_{s,L1}), \mathbf{C}(\mathbf{V}_{s,KL})$  and  $\{OPE(\mathbf{V}_s[sf_j])\}$  are sent to the cloud as the annotation request.

#### 3.8.1.4 Privacy-preserving Annotation on Cloud

On receiving the encrypted request, the cloud first performs a privacy-preserving search over the encrypted RKDF. As described in Algorithm 1, the cloud conducts parallel search over each encrypted tree in the RKDF. There are three places that require the cloud to conduct privacy-preserving computation over encrypted data:

- During the top-down traversal, as the *split* field element of each non-leaf node is encrypted using order-preserving encryption, the cloud can directly compare their ciphertexts (line 7) to determine which node to be checked next.
- In the back trace process, the cloud needs to perform privacy-preserving comparison to determine whether the current node's sibling branch needs to be searched (line 24 to 29). In particular, given  $\mathbf{C}(\mathbf{V}_{s,L1}), \mathbf{C}(\mathbf{H}_{s,L1}), \mathbf{C}(\mathbf{V}_{qL,L1}), \mathbf{C}(\mathbf{H}_{parent,L1}), \mathbf{C}(\mathbf{V}_{s,KL}), \mathbf{C}(\mathbf{V}_{qL,KL}),$  and  $\mathbf{C}(\mathbf{H}_{parent,KL}),$  the cloud first uses type-2 distance comparison in *PL1C – RF* and *PKLC – RF* to compute

$$\begin{aligned} & \mathbf{V}_{qL,L1} \mathbf{V}_{s,L1}^T, & \mathbf{V}_{qL,KL} \mathbf{V}_{s,KL}^T, \\ & \mathbf{H}_{parent,L1} \mathbf{H}_{s,L1}^T, & \mathbf{H}_{parent,KL} \mathbf{V}_{s,KL}^T \end{aligned}$$

Then, the distance comparison is executed as

$$\begin{aligned}
Comp_{qL} &= -2(\mathbf{V}_{qL,L1}\mathbf{V}_{s,L1}^T) + \mathbf{V}_{qL,KL}\mathbf{V}_{s,KL}^T \\
Comp_h &= -2(\mathbf{H}_{parent,L1}\mathbf{H}_{s,L1}^T) + \mathbf{H}_{parent,KL}\mathbf{V}_{s,KL}^T \\
\\
Comp_{qL} - Comp_h & \tag{3.13} \\
&= r_s(DL1_{qL,s}^{L1} - DL1_{parent,s}^{L1}) + 2(\epsilon'_{parent} - \epsilon_{qL}) \\
&+ r_s(DKL_{qL,s}^{LAB} - DKL_{parent,s}^{LAB}) + (\epsilon'_{parent} - \epsilon_{qL}) \\
&= r_s(Dis(\mathbf{V}_{qL}, \mathbf{V}_s) - Dis(\mathbf{H}_{parent}, \mathbf{V}_s)) \\
&+ 3(\epsilon'_{parent} - \epsilon_{qL})
\end{aligned}$$

where  $\mathbf{V}_{qL}$  is the least closest vector to  $\mathbf{V}_{req}$  in priority queue *Queue*. As  $r_s$  is a positive value and  $r_s \gg (\epsilon'_{parent} - \epsilon_{qL})$ , the sign of  $Comp_{qL} - Comp_h$  is consistent with  $Dis(\mathbf{V}_{qL}, \mathbf{V}_s) - Dis(\mathbf{H}_{parent}, \mathbf{V}_s)$ .

- In the *Queue* push process (line 30-37), privacy-preserving distance comparison is needed to determine whether a new node shall be added. Specifically, given  $\mathbf{C}(\mathbf{V}_{s,L1})$ ,  $\mathbf{C}(\mathbf{V}_{Node,L1})$ ,  $\mathbf{C}(\mathbf{V}_{qL}, L1)$ ,  $\mathbf{C}(\mathbf{V}_{s,KL})$ ,  $\mathbf{C}(\mathbf{V}_{Node,KL})$ ,  $\mathbf{C}(\mathbf{V}_{qL}, KL)$ , the cloud use type-1 distance comparison in *PL1C - RF* and *PKLC - RF* to perform distance comparison as

$$\begin{aligned}
Comp_{Node} &= -2(\mathbf{V}_{Node,L1}\mathbf{V}_{s,L1}^T) + \mathbf{V}_{Node,KL}\mathbf{V}_{s,KL}^T \\
Comp_{qL} &= -2(\mathbf{V}_{qL,L1}\mathbf{V}_{qL,L1}^T) + \mathbf{V}_{cur,KL}\mathbf{V}_{s,KL}^T
\end{aligned}$$

$$\begin{aligned}
& Comp_{Node} - Comp_{qL} \tag{3.14} \\
&= r_s(DL1_{Node,s}^{L1} - DL1_{qL,s}^{L1}) + 2(\epsilon_{qL} - \epsilon_{Node}) \\
&+ r_s(DKL_{Node,s}^{LAB} - DKL_{qL,s}^{LAB}) + (\epsilon_{qL} - \epsilon_{Node}) \\
&= r_s(Dis(\mathbf{V}_{Node}, \mathbf{V}_s) - Dis(\mathbf{V}_{qL}, \mathbf{V}_s)) \\
&+ 3(\epsilon_{qL} - \epsilon_{Node})
\end{aligned}$$

To this end, the cloud is able to perform all operations required by a RKDF search in the privacy-preserving manner, and obtain a *Queue* of nodes that stores data of top related images to the request. The cloud returns distance comparison candidates (type-1 distance)  $Comp_i, i \in Queue$  as well as corresponding encrypted keywords back to the user.

### 3.8.1.5 Final Keyword Selection

After the user retrieves the keyword set  $K_{i,t}$  from the the cloud returned data, he/she can computes distance  $Dis(\mathbf{V}_i, \mathbf{V}_s) = (2r + \sum_{j=1}^{m_{L1}} v_{s,L1,j}^2) + \frac{-2(\mathbf{V}_{i,L1} \mathbf{V}_{s,L1}^T) + \mathbf{V}_{i,KL} \mathbf{V}_{s,KL}^T}{r_s}$  as in 3.11. Following 3.12, the user can also figure out the weight ranking  $W_t$  of each keyword  $K_{i,t}$  and selects keywords for the image.

## 3.9 Conclusion

In this chapter, I introduce privacy-preserving distance comparison design *PL1C* and *PKLC* along with their enhanced modules *PL1C - RF* and *PKLC - RF*. I integrate these modules in a practical cloud-assisted image annotation task to show their utility in imagery data analysis. Moreover, my privacy-preserving distance comparison modules



**Input** : Encrypted Search Request (Req) for  $\mathbf{V}_s$ , Encrypted RKDF with a set of Trees  $\{T_k\}$ , approximation power  $\mathcal{AP} - \mathcal{X}$

**Output**: Encrypted Nodes Associated with Top Related Images to the Request.

Initialization  $Queue = []$ ,  $Path = []$  (Searched Path),  $Vis = []$  (Visited Nodes),  $Node_k = T_k.root$ ; Each tree  $T_k$  executes  $topDownTraversal()$  and  $backTraceSearch()$  in parallel,  $Queue$  and  $Vis$  are shared among all trees;

**Function**  $topDownTraversal(Req, Node_k)$ :

```

if  $Node_k$  is not null then
  | return;
end
 $\mathbf{V}_i \leftarrow Node_k.\mathbf{V}_i$ ;
if  $OPE(\mathbf{V}_s[s_i]) \leq OPE(\mathbf{V}_i[s_i])$  then
  |  $Node_k = topDownTraversal(Node_k.left-child)$ ;
else
  |  $Node_k = topDownTraversal(Node_k.right-child)$ ;
end
if  $Node_k \notin Vis$  then
  |  $Vis.push(Node_k)$ ;
  |  $Queue.push(Node_k)$ ;
end
 $Path.push(Node_k)$ ;
return  $Node_k$ ;

```

**Function**  $backTraceSearch(Req, Node_k)$ :

```

if  $Vis.length() > \mathcal{AP} - \mathcal{X} \times \text{Nodes Number}$  then
  | return  $Queue$ ;
end
if  $Path$  is not null then
  |  $parent \leftarrow Path.pop()$ ;
end
if  $parent \notin Vis$  then
  |  $Vis.push(parent)$ ;
  |  $Queue.push(parent)$ ;
end
//Privacy-preserving distance comparison is achieved by  $PL1C - RF$  and  $PKLC - RF$ ,
 $\mathbf{V}_{qL}$  is the least closest vector to  $\mathbf{V}_s$  in  $Queue$ 
if  $Dis(\mathbf{V}_{qL}, \mathbf{V}_s) < Dis(\mathbf{H}_{parent}, \mathbf{V}_s)$  then
  |  $backTraceSearch(Req, parent)$ ;
else
  |  $Node_k = topDownTraversal(Req, Node_k.sibling)$ ;
end
return  $Queue$ ;

```

**Function**  $Queue.push(Node)$ :

```

//Each  $Node_q$  in  $Queue$  are ordered by  $Dis(\mathbf{V}_s, \mathbf{V}_{Node_q})$ 
if  $Queue.length() < \text{Defined Size } L$  then
  | Add  $Node$  into  $Queue$  by order;
else
  | if  $Node_{qL}$  in  $Queue$  has  $Dis(\mathbf{V}_s, \mathbf{V}_{Node}) < Dis(\mathbf{V}_s, \mathbf{V}_{qL})$  then
    | Remove  $Node_{qL}$  from  $Queue$ ;
    | Add  $Node$  into  $Queue$  by order;
  | end
end

```

**Algorithm 1:** Privacy-preserving RKDF Search

and privacy-preserving randomized k-d forest can also be utilized as independent tools for other related fields, especially for efficient similarity measurement on encrypted data. In the next chapter, I will analyze the security of my design and provide experiment results on my prototype implementation.

## Chapter 4

# Evaluation of Privacy Protection Modules for Descriptor Based Image Analysis

In this chapter, I first analyze the security of my design against several attacks. Then I provide experimental results to evaluate the practical performance of my privacy protection modules as well as their enhancement for descriptor based image analysis stated in Chapter 3.

### 4.1 Security Analysis

In CAPIA, I have the following privacy related data: feature vectors

$\{\mathbf{V}_{i,L1}, \mathbf{V}_{i,KL}\}_{1 \leq i \leq n}$  and keywords of image  $I_i$  in the pre-annotated dataset; feature vectors  $\mathbf{V}_{s,L1}, \mathbf{V}_{s,KL}$  of the image requested for annotation. As keywords are encrypted using standard AES encryption, I consider them secure against the cloud server as well

as outside adversaries. With regards to  $\mathbf{V}_{i,L1}$ ,  $\mathbf{V}_{i,KL}$ ,  $\mathbf{V}_{s,L1}$ ,  $\mathbf{V}_{s,KL}$ , they are encrypted using the encryption scheme of IVE [68] after pre-processing as presented in my *PL1C* and *PKLC* schemes. The IVE scheme [68] has been proved to be secure based on the well-known Learning with Errors (LWE) hard problem [77]. Thus, given the ciphertexts  $\mathbf{C}_{i,L1}$ ,  $\mathbf{C}_{i,KL}$ ,  $\mathbf{C}_{s,L1}$ ,  $\mathbf{C}_{s,KL}$  only, it is computational infeasible for the cloud server or outside adversaries to recover  $\mathbf{V}_{i,L1}$ ,  $\mathbf{V}_{i,KL}$ ,  $\mathbf{V}_{s,L1}$ ,  $\mathbf{V}_{s,KL}$ .

#### 4.1.1 Security of Outsourcing $\mathbf{S}_{L1}^T \mathbf{S}_{s,L1}$ and $\mathbf{S}_{KL}^T \mathbf{S}_{s,KL}$

As  $\mathbf{S}_{L1}^T \mathbf{S}_{s,L1}$  and  $\mathbf{S}_{KL}^T \mathbf{S}_{s,KL}$  are used in the same manner, I use  $\mathbf{S}^T \mathbf{S}_s$  to denote them for expression simplicity. Different from the original Encryption algorithm of IVE, the user in CAPIA also outsources  $\mathbf{S}^T \mathbf{S}_s$  to the cloud besides ciphertexts  $\mathbf{C}_{i,L1}$ ,  $\mathbf{C}_{i,KL}$ ,  $\mathbf{C}_{s,L1}$ ,  $\mathbf{C}_{s,KL}$ . As all elements in  $\mathbf{S}$  and  $\mathbf{S}_s$  are randomly selected, elements in their multiplication  $\mathbf{S}^T \mathbf{S}_s$  have the same distribution as these elements in  $\mathbf{S}$  and  $\mathbf{S}_s$  [78]. Thus, given  $\mathbf{S}^T \mathbf{S}_s$ , the cloud server is not be able to extract  $\mathbf{S}$  or  $\mathbf{S}_s$  directly and use them to decrypt  $\mathbf{C}_{i,L1}$ ,  $\mathbf{C}_{i,KL}$ ,  $\mathbf{C}_{s,L1}$ ,  $\mathbf{C}_{s,KL}$ . By combining  $\mathbf{S}^T \mathbf{S}_s$  with ciphertexts  $\mathbf{C}_{i,L1}$  and  $\mathbf{C}_{s,L1}$  (same as that for  $\mathbf{C}_{i,KL}$  and  $\mathbf{C}_{s,KL}$ ), the cloud can obtain

$$\begin{aligned} \mathbf{S}^T \mathbf{S}_s \mathbf{C}_{i,L1} &= \mathbf{S}^T \mathbf{S}_s \mathbf{S}^{-1} (w \mathbf{V}_{i,L1} + \mathbf{e}_i)^T \\ \mathbf{S}^T \mathbf{S}_s \mathbf{C}_{s,L1} &= \mathbf{S}^T \mathbf{S}_s \mathbf{S}_s^{-1} (w \mathbf{V}_{s,L1} + \mathbf{e}_i)^T = \mathbf{S}^T (w \mathbf{V}_{s,L1} + \mathbf{e}_i)^T \end{aligned}$$

From the above two equations, it is clear that the combination of  $\mathbf{S}^T \mathbf{S}_s$ ,  $\mathbf{C}_{i,L1}$  and  $\mathbf{S}^T \mathbf{S}_s$ ,  $\mathbf{C}_{s,L1}$  only transfer them to the ciphertexts of  $\mathbf{V}_{i,L1}$  and  $\mathbf{V}_{s,L1}$  that encrypted using the IVE scheme with new keys  $\mathbf{S}^T \mathbf{S}_s \mathbf{S}^{-1}$  and  $\mathbf{S}^T$  respectively. As  $\mathbf{S}^T \mathbf{S}_s \mathbf{S}^{-1}$  and  $\mathbf{S}^T$  are random keys and unknown to the cloud, recovering  $\mathbf{V}_{i,L1}$ ,  $\mathbf{V}_{s,L1}$  from  $\mathbf{S}^T \mathbf{S}_s \mathbf{C}_{i,L1}$ ,  $\mathbf{S}^T \mathbf{S}_s \mathbf{C}_{s,L1}$  still become the *LWE* problem as proved in ref [68]. To this end,  $\mathbf{S}^T \mathbf{S}_s$  only

helps the cloud to perform distance comparison in CAPIA, but does not bring advantages to recover feature vectors compared with the given ciphertexts only scenario.

### 4.1.2 Known Ciphertexts-Image Pairs

I now consider that the cloud server gets a set of ciphertexts-image pairs from the background analysis as  $\{\mathbf{V}_{i,L1}, \mathbf{C}_{i,L1}\}$  ( $\{\mathbf{V}_{s,L1}, \mathbf{C}_{s,L1}\}$ ,  $\{\mathbf{V}_{i,KL}, \mathbf{C}_{i,KL}\}$ ,  $\{\mathbf{V}_{s,KL}, \mathbf{C}_{s,KL}\}$  respectively). In ref [79], a linear analysis attack based on ciphertext-image pairs is introduced to recover vectors from their distance comparison result. In particular, instead of trying to recovering feature vectors or secret keys directly from ciphertexts, such an attack attempts to recover the vectors from the distance comparison result by constructing and solving enough number (i.e., greater than the dimension of vector) of linear equations. To launch this kind of linear analysis attack to CAPIA, there are two necessary requirements that need to be fulfilled simultaneously: 1) The cloud obtains at least  $m$  ciphertext-image pairs, where  $m$  is the dimension of feature vectors; 2) The cloud has access to the exact  $L_1$  distance and KL Divergence comparison results. As shown in Eq.3.4 and Eq.3.7, CAPIA only provides scaled and obfuscated comparison results by adding noise terms  $\epsilon_i$  and random scaling factor  $r_c$ . As a result, the cloud cannot fulfill the second requirement to launch a successful linear analysis attack to CAPIA. To this end, CAPIA is secure even a set of ciphertexts-image pairs are obtained by the cloud server.

### 4.1.3 Request Unlinkability

The request unlinkability in CAPIA is guaranteed by the randomization for each request. Specifically, each query request  $\mathbf{V}_{s,L1}, \mathbf{V}_{s,KL}$  is element-wise obfuscated with different

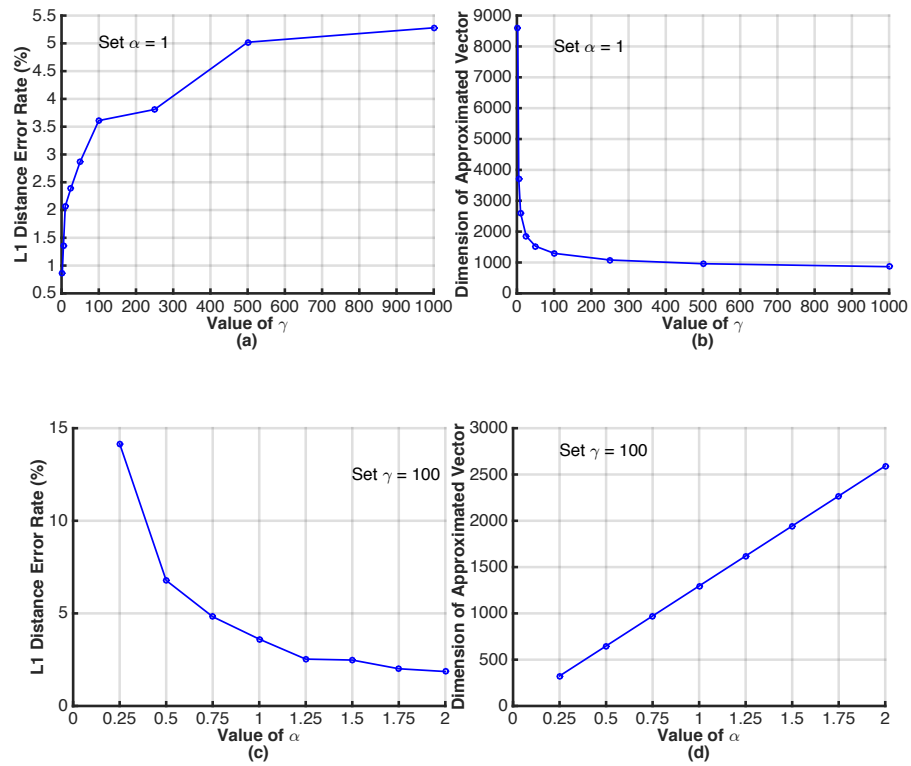


FIGURE 4.1: Error rate of Approximation and Dimension of Approximated Vector (PCA - 32)

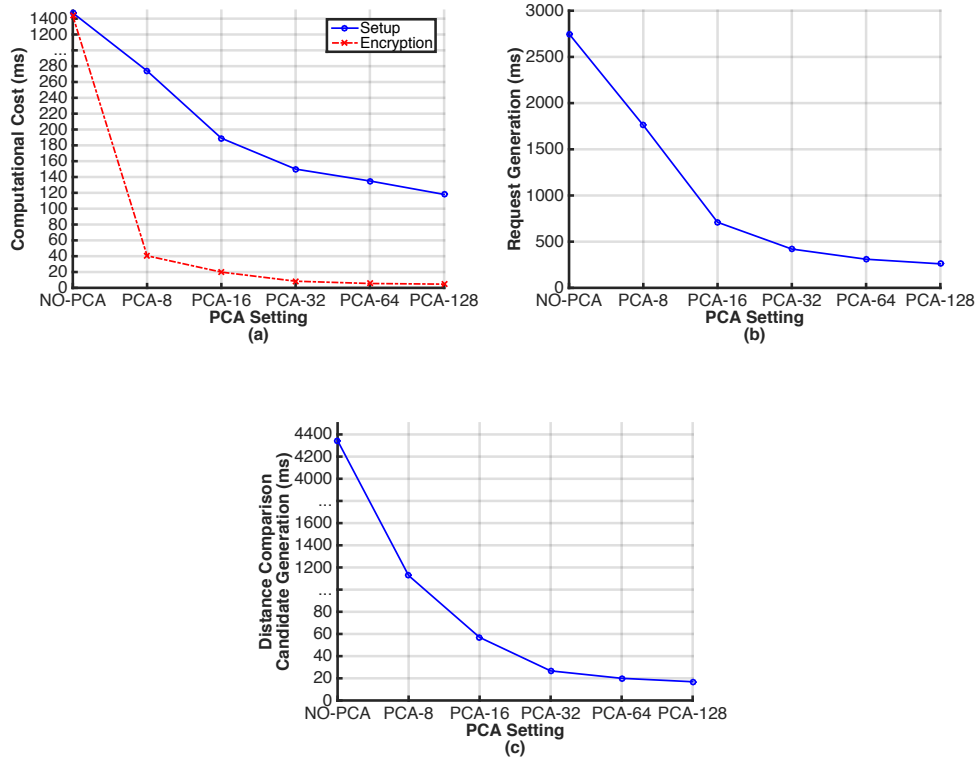


FIGURE 4.2: (a) System Setup and Encryption Cost (b) Request Generation Cost (c) Distance Comparison Candidate Generation Cost on Cloud

random error terms  $\mathbf{e}_s$  and random number  $r_s$  during the encryption, which makes the obfuscated  $\mathbf{V}_{s,L1}, \mathbf{V}_{s,KL}$  have the same distribution as these random values in  $\mathbf{e}_s$  and  $r_c$  [78]. Thus, by changing  $\mathbf{e}_s$  and  $r_c$  during the encryption of different requests, CAPIA outputs different random ciphertexts, even for requests generated from the same image.

## 4.2 Evaluation of CAPIA

To evaluate the performance of CAPIA, I implemented a prototype using Python 2.7. In my implementation, Numpy [80] is used to support efficient multi-dimension array operations. OpenCV [81] is used to extract the color-space features of the images and build the filter kernels to generate the Gabor filter results. Pywt [82] is adopted to perform Haar wavelet and get the corresponding Haar results. Sklearn [83] is used to perform the PCA transformation. I use the well-known IAPR TC-12 [84] as the pre-annotated dataset, which contains 20,000 annotated images and the average number of keywords for each image is 5.7. All tests are performed on a 3.1 GHz Intel Core i7 Macbook Pro with OS X 10.11.6 installed.

In the rest of this section,  $n$  is the total number of images in the pre-annotated dataset;  $m_{L1}$  and  $m_{KL}$  are dimensions of vectors  $\mathbf{V}_{i,L1}$  and  $\mathbf{V}_{i,KL}$  after pre-processing respectively;  $PCA - X$  is used to denote the strength of  $PCA$  transformation applied to  $\mathbf{V}_{i,H}$  and  $\mathbf{V}_{i,HQ}$  in  $\mathbf{V}_{i,L1}$ , which compresses their dimensions from 4096 to  $\frac{4096}{X}$ .  $PCA - 128$ ,  $PCA - 64$ ,  $PCA - 32$ ,  $PCA - 16$ , and  $PCA - 8$  are evaluated in my experiments to balance the efficiency and accuracy of CAPIA. I also use  $DOT_m$  to denote a dot product operation between two  $m$ -dimensional vectors.

### 4.2.1 System Setup and Dataset Encryption

To perform the one-time setup in CAPIA, the user pre-processes feature vectors of each image in the pre-annotated image dataset. Specifically, the user first performs JL-Lemma based approximation over  $\mathbf{V}_{i,L1}$  to make  $\mathbf{V}_{i,L1}$  compatible with my *PL1C*. As discussed in Section 3.4.2, there is a trade-off between the approximation accuracy of  $L_1$  distance and length of the approximated vector that determines efficiency of follow up privacy-preserving operations. To balance such a trade-off, I evaluate different parameters for approximation as shown in Fig.4.1 (a)-(d). According to my results, I suggest to set  $\alpha = 1$  and  $\gamma = 100$  which introduces 3.61% error rate for  $L_1$  distance computation, and extends the dimension of  $\mathbf{V}_{i,L1}$  from 864 to 1296 under the setting of *PCA – 32*. The selection of PCA strength will be discussed and evaluated in Section 4.3.2. Specifically, the error rate drops fast when  $\alpha < 1$  and becomes relative stable when  $\alpha > 1$ . Meanwhile, the dimension of the approximated vector increases linearly to the value of  $\alpha$ . With regard to  $\gamma$ , the dimension of the approximated vector becomes relative stable when  $\gamma > 100$ , however, the error rate still increases when  $\gamma > 100$ . As shown in Fig.4.2 (a), such an approximation setting makes the pre-processing procedure cost 1471ms to 118ms for each image with PCA setting from *No PCA* to *PCA – 128*.

After the pre-processing,  $\{\mathbf{V}_{i,L1}, \mathbf{V}_{i,KL}\}_{1 \leq i \leq n}$  will be encrypted using the *Data Encryption* procedures of my *PL1C* and *PKLC* schemes respectively. As shown in Eq.3.3 and Eq.3.6, the encryption of each  $\mathbf{V}_{i,L1}$  and  $\mathbf{V}_{i,KL}$  requires  $(m_{L1})DOT_{m_{L1}}$  and  $(m_{KL})DOT_{m_{KL}}$  operations respectively. Fig.4.2 (a) shows the total encryption cost for  $\mathbf{V}_{i,L1}$  and  $\mathbf{V}_{i,KL}$  of a pre-annotated image decreases from 1436ms to 4.7ms by increasing the strength of PCA from *No PCA* to *PCA – 128*. This is because the dimension of  $\mathbf{V}_{i,L1}$ , i.e.,  $m_{L1}$ , is determined by the strength of PCA, which is directly correlated



to the encryption cost of  $\mathbf{V}_{i,L1}$ . Same as the system setup, encrypting feature vectors is also a one-time cost, which does not impact the performance of later on real-time privacy-preserving image annotation.

#### 4.2.2 Real-time Image Annotation

**Efficiency:** To annotate a new image in a privacy-preserving manner, the user generates an encrypted request by pre-processing and encrypting feature vectors of the requested image. By varying the PCA strength from *No PCA* to *PCA – 128*, Fig.4.2 (b) shows that the request generation spends from 2775ms to 268ms. On receiving the encrypted request, the cloud first computes distance comparison candidate  $Comp_i$  for each image  $I_i, 1 \leq i \leq n$  in the pre-annotated dataset, which requires a  $(m_{L1} + 1)DOT_{m_{L1}}$  operation and a  $(m_{KL} + 1)DOT_{m_{KL}}$  operation as shown in Eq.3.8 and Eq.3.9. By changing the strength of PCA from *No PCA* to *PCA – 128*, the computational cost for  $Comp_i$  changes from 4334ms to 16.9ms as shown in Fig.4.2 (c). This is because the dimension of  $\mathbf{V}_{i,L1}$ , i.e.,  $m_{L1}$ , is determined by the strength of PCA and  $m_{L1} \gg m_{KL}$  (e.g., 1296 v.s. 48 in *PCA – 32*). Afterwards, the cloud selects encrypted keywords according the ranking of  $Comp_i$  as Eq.3.10. It is worth to note that the annotation process on cloud can be easily parallelized for performance optimization. In particular, computation of  $Comp_i$  for different pre-annotated images are independent with each other, and thus can be easily parallelized in the cloud computing environment.

**Accuracy:** I now evaluate the accuracy of CAPIA. In my evaluation, I use the standard average *precision* and *recall* rates to measure the accuracy of keywords annotation as that in automatic annotation using plaintext images. I use 50 images as annotation requests, and each image will be assigned ten keywords after automatic annotation. Each request has two or more related images in the pre-annotated dataset. I use set

$[K_1, K_2, \dots, \dots K_x]$  to denote distinct keywords annotated for all 50 requested images. The annotation precision and recall rate for a keyword  $K_j, 1 \leq j \leq x$  in these 50 requests are defined as

- $precision_{K_j}$ : number of images assigned  $K_j$  correctly in CAPIA divided by the total number of images assigned  $K_j$  in CAPIA.
- $recall_{K_j}$ : number of images assigned  $K_j$  correctly in CAPIA divided by the number of images assigned  $K_j$  in the ground-truth annotation.

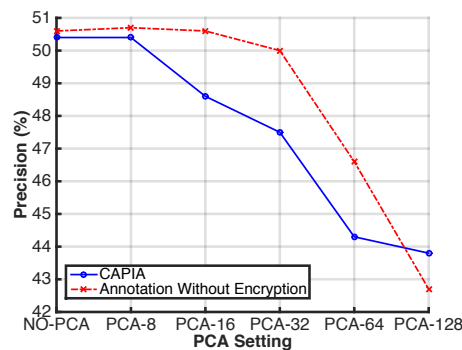


FIGURE 4.3: Precision of CAPIA and Annotation without Encryption

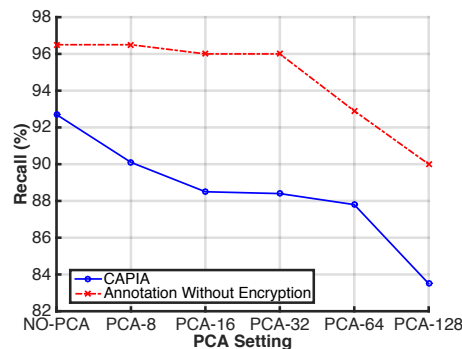


FIGURE 4.4: Recall of CAPIA and Annotation without Encryption

To compare the annotation accuracy of CAPIA, I also evaluate the no-privacy-preserving annotation using the same 50 requests. As shown in Fig.4.3 and Fig.4.4, while providing strong privacy guarantee, CAPIA introduces less than 2.5% and 7.5% accuracy loss in terms of average precision and recall rates with PCA setting from *No PCA* to *PCA* – 128. In addition, Fig.4.3 and Fig.4.4 also demonstrate that the increasing of PCA

strength reduces the annotation accuracy of CAPIA to some extent, especially from  $PCA - 32$  to  $PCA - 64$ . Taking the efficiency enhancement brought by PCA together into consideration, I suggest to use  $PCA - 32$  as an appropriate setting for practical usage. Specifically, Fig.4.2 demonstrates the efficiency improvement from PCA becomes relative stable after  $PCA - 32$ . Meanwhile, the accuracy loss of CAPIA still increases quickly after  $PCA - 32$ .

In Table 4.2, I present samples of automatically annotated images using CAPIA. On one hand, CAPIA is highly possible to assign correct keywords to images compared with human annotation. This observation also confirms the high average recall rate of CAPIA, since these ground-truth annotations are likely to be covered in CAPIA. On the other hand, CAPIA also introduces additional keywords that frequently appear together with these accurate keywords in top related images. These additional keywords are typically not directly included in human annotations, but are potentially related to correct keywords. Such a fact also explains why the average precision rate of CAPIA is relatively low compared with the average recall rate. Overall, my evaluation results demonstrate that although CAPIA cannot provide perfect keywords selection all the time compared with human annotation, it is still promising for automatically assigning keywords to images, and hence fulfilling the fundamental gap between SE schemes and images.

### 4.2.3 Communication Cost and Storage Overhead

The communication cost in CAPIA comes from two major parts: annotation request and encrypted results returned from the cloud server. The encrypted request consists of a  $m_{L1}$ -dimensional vector  $\mathbf{C}_{s,L1}$  and a  $m_{KL}$ -dimensional vector  $\mathbf{C}_{s,KL}$ . In the  $PCA - 32$  setting, the total communication cost for a request is 26KB. Meanwhile, the returned





Image	CAPIA Annotation	Human Annotation
	floor-tennis-court, man, woman	floor-tennis-count, man
	sky-blue, highway, vegetation, ground, bush, trees, lake, ocean	highway, sky-blue, trees, vegetation
	cloud, sky-blue, ground, mountain, horse man, road, grass	ground, cloud sky-blue, mountain snow, grass
	group-of-persons, sky-blue, ground, trees, mountain, ruin-archeological, hat cloud, hill	trees, ground, man, sky-blue, group-of-persons

TABLE 4.1: Sample Annotation Results

result contains encrypted keywords and distance comparison candidates  $Comp_i$  of top 10 related images. Using AES-256 for keywords encryption, the total size for the returned result is 488 Bytes with the average number of keywords for each pre-annotated image as 5.7. With regard to the storage overhead of CAPIA, it includes two parts for each pre-annotated image  $I_i$ : 1) encrypted feature vectors  $\mathbf{C}_{i.L1}$  and  $\mathbf{C}_{i.KL}$ , which are 26KB in total. 2) Encrypted keywords, which are 480 Bytes as average using AES-256 encryption.

### 4.3 Evaluation of CPAR

To evaluate the performance of CPAR, I use Python 2.7 to build the prototype of my privacy-preserving randomized k-d forest. I utilize FLANN library [59] to implement the non-privacy randomized k-d forest for comparison. All tests are performed on a 3.1 GHz Intel Core i7 Macbook Pro with OS X 10.14.2 installed as *User* and a Microsoft Azure cloud E4-v3 VM with Ubuntu 18.04 LTS installed as *Cloud Server*. Other experiment environment configurations, dataset selection and notations stay aligned with CAPIA. In the rest part of this section, I first provide numerical analysis as well as experimental

evaluation for each stage of CPAR. Then, I compare CPAR with CAPIA in terms of efficiency and accuracy.

### 4.3.1 RKDF Construction and Encryption

To construct an encrypted RKDF, the user first constructs an unencrypted RKDF using 20,000 pre-annotated images, and then replaces data of each node in the RKDF with their corresponding ciphertexts. The construction of an unencrypted RKDF with 10 k-d trees costs 28.56 seconds. Then, for the pre-processed feature vectors  $\mathbf{V}_{i,L1}$  and  $\mathbf{V}_{i,KL}$  of each image, the user can encrypt them using *PL1C – RF* and *PKLC – RF* with  $(m_{L1})DOT_{m_{L1}}$  and  $(m_{KL})DOT_{m_{KL}}$  operations respectively, which costs 8.4ms in total in my implementation. If an image is associated with a non-leaf node in any tree of the RKDF, encryption for the hyperplane projected vectors  $\mathbf{H}_{i,L1}$  and  $\mathbf{H}_{i,KL}$  with  $(m'_{L1})DOT_{m'_{L1}}$  and  $(m_{KL})DOT_{m_{KL}}$  operations respectively, which costs 54.7ms in total. In addition, for each non-leaf node, an order-preserving encryption is needed for the *split* field, each of which costs 1.4ms. Therefore, to build a 10-tree encrypted RKDF with a 20,000 pre-annotated image dataset, it takes 74.78 minutes in my implementation. It is noteworthy that the encrypted RKDF construction is one-time offline cost, which does not impact the performance of later on real-time privacy-preserving image annotation.

### 4.3.2 Real-time Image Annotation

*Request Generation:* To annotate a new image in a privacy-preserving manner, the user pre-processes and encrypts its feature vectors  $\mathbf{V}_{s,L1}$  and  $\mathbf{V}_{s,KL}$  using *PL1C – RF* and *PKLC – RF*. Specifically, the encryption of  $\mathbf{V}_{s,L1}$  requires  $(m_{L1})DOT_{m_{L1}} + (m'_{L1})DOT_{m'_{L1}}$  for shown in Fig.3.2, and the encryption of  $\mathbf{V}_{s,KL}$  requires  $(m_{KL})DOT_{m_{KL}}$

operations as shown in Fig.3.3. In addition, for each element  $sf_j$  in the *split* field element set  $\mathcal{SF}$  with size of 348 in my implementation, order-preserving encryption are executed for  $\mathbf{V}_s[sf_j]$ . As a result, the encrypted request can be efficiently generated with only 534.16ms.

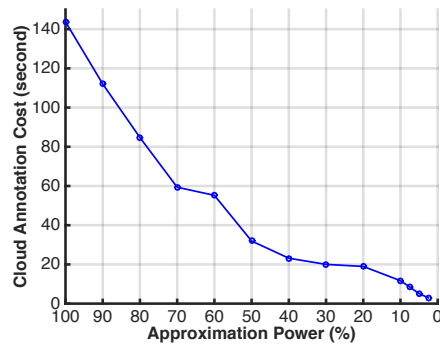


FIGURE 4.5: Privacy-preserving Annotation Cost on Cloud with Different Approximation Power

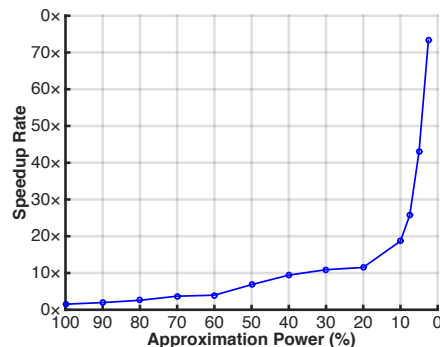


FIGURE 4.6: Speedup Rate with Different Approximation Power

*Privacy-preserving Annotation on Cloud:* On receiving the encrypted request, the cloud performs privacy-preserving RKDF search with top-down traversal, back trace search, and queue push processes. The top-down traversal only requires a direct comparison between the ciphertexts under order-preserving encryption, whose cost is negligible compared with the other two processes. In the back trace search, privacy-preserving type-2 distance comparison needs to be executed using  $PL1C - RF$  and  $PKLC - RF$ . In particular, two comparison candidates  $Comp_{qL}$  and  $Comp_h$  are computed with  $(m_{L1} +$

1)  $DOT_{m_{L1}} + (m_{KL} + 1)DOT_{m_{KL}}$  operations and  $(m'_{L1} + 1)DOT_{m'_{L1}} + (m_{KL} + 1)DOT_{m_{KL}}$  operations respectively. With regards to the queue push process, privacy-preserving type-1 distance comparison are executed using  $PL1C - RF$  and  $PKLC - RF$ , which requires  $2(m_{L1} + 1)DOT_{m_{L1}} + 2(m_{KL} + 1)DOT_{m_{KL}}$  operations in total. Another important parameter that affects the search efficiency is the selection of approximation power  $\mathcal{AP} - \mathcal{X}$ . As depicted in Fig.4.5, by increasing the approximation power from  $\mathcal{AP} - 100$  to  $\mathcal{AP} - 2.5$ , the privacy-preserving annotation using encrypted RKDF reduces from 143.72 seconds to 2.98 seconds. Compared with CAPIA [85] that requires 218.46 seconds for one privacy-preserving annotation on cloud and does not support approximate dataset checking, CPAR can significantly speed it up as depicted in Fig.4.6.

*Final Keyword Selection:* This process only involves AES decryption and the weights generation that only requires a small number of additions. As a result, the final keyword selection can be completed by the user within 318ms.

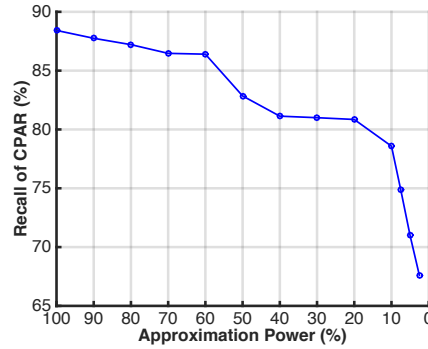


FIGURE 4.7: Accuracy (Recall) of CPAR with Different Approximation Power

*Accuracy:* I use the same average *recall* rates as defined in Section 6.2 to evaluate the accuracy of CPAR. In my evaluation, annotation requests for 50 different images are submitted, in which each requested image has two or more related images in the pre-annotated dataset. As shown in Fig.4.7, the accuracy of CPAR reduces from 88.42% to 67.59% when the approximation power increases from  $\mathcal{AP} - 100$  to  $\mathcal{AP} - 2.5$ . Compared

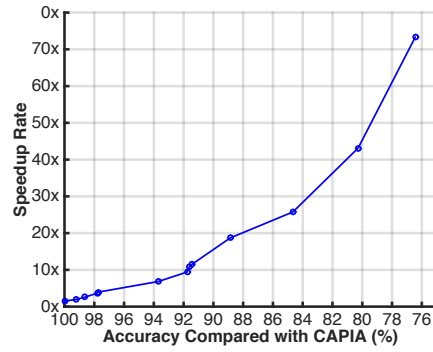


FIGURE 4.8: Speedup rate of CPAR with Different Accuracy Compared with CAPIA





with CAPIA [85] my scheme achieves the same accuracy by setting the approximation power as  $\mathcal{AP} - 100$ . While the increasing of approximation power reduces the accuracy of CPAR to some extent, it also boosts the efficiency significantly as shown in Fig.4.5. Compared with CAPIA, Fig.4.8 shows that CPAR can speed up CAPIA by  $4\times$ ,  $11.5\times$ ,  $18.7\times$ ,  $25.8\times$ ,  $43.1\times$  when achieving 97.7%, 91.4%, 88.9%, 84.7%, 80.3% accuracy of CAPIA respectively. Therefore, CPAR can greatly promote the efficiency the of CAPIA while retaining comparable accuracy. To balance the efficiency speedup and annotation accuracy of CPAR, I suggest to set the approximation power as  $\mathcal{AP} - 10$ , i.e. achieves 88.9% accuracy of CAPIA with  $18.7\times$  speedup.

In Table 4.2, I present samples of automatically annotated images using CAPIA and CPAR with approximation power as  $\mathcal{AP} - 10$ . In the last column I list the human annotation results (ground-truth) for comparison. On one hand, CPAR is highly possible to assign correct keywords to images compared with human annotation. This observation also confirms the high average recall rate of CPAR, since these ground-truth annotations are likely to be covered in CPAR. On the other hand, CPAR also introduces additional keywords that frequently appear together with these accurate keywords in top related images. These additional keywords are typically not directly included in human annotations, but are potentially related to correct keywords. Compared with CAPIA, CPAR



only misses a small portion of ground-truth keywords due to the approximation strategy, which is consistent with my evaluation result in Fig.4.7 and Fig.4.8. Overall, my evaluation results demonstrate that although CPAR cannot provide perfect keywords selection all the time compared with human annotation, it can still maintain comparable accuracy as CAPIA and is promising for automatically assigning keywords to images.

TABLE 4.2: Sample Annotation Comparison between CAPIA and CPAR

Image	CPAR Annotation	CAPIA Annotation	Human Annotation
	<u>floor-tennis-court</u> , <u>man</u> , grass	<u>floor-tennis-court</u> , <u>man</u> , woman	floor-tennis-court, man
	<u>highway, sky-blue,</u> <u>trees, vegetation</u> , ground, ship, sky, ocean, bush	<u>sky-blue, highway,</u> <u>vegetation</u> , ground, bush, <u>trees</u> , lake, ocean	highway, sky-blue, trees, vegetation
	group-of-persons, <u>ground, cloud</u> , man, sky-light, <u>mountain</u> , door, chair, floor-other, column	<u>cloud, sky-blue,</u> <u>ground, mountain</u> , horse man, road, <u>grass</u>	ground, cloud, sky-blue, mountain, snow, grass
	<u>group-of-persons</u> , hat, hill, cloud, <u>sky-blue, ground</u> , sky, fabric, couple-of-persons, grass	<u>group-of-persons</u> , <u>sky-blue, ground</u> , <u>trees</u> , mountain, ruin-archeological, hat, cloud, hill	trees, ground, man, sky-blue, group-of-persons

In each cell of CPAR and CAPIA annotation results, ground-truth human annotation results are underlined and bold out.

*Communication Cost:* The communication cost of CPAR does not have big difference compared with CAPIA. The encrypted request introduces an additional  $m'_{L1}$ -dimensional vector  $\mathbf{C}(\mathbf{H}_{s,L1})$  and a set of encrypted *split* field elements  $\mathcal{SF}$ .  $\mathbf{C}(\mathbf{H}_s)$  and  $\mathcal{SF}$  add 48K and 4KB communication respectively and make the total communication cost for a request to be 80KB. Meanwhile, since the size of the returned results contain encrypted keywords and distance comparison candidates is not changed, the total size for the returned result is maintained at 488 Bytes. Therefore, CPAR does

not introduce heavy communication on the basis of CAPIA and the communication cost for each privacy-preserving annotation can be efficiently handled in today's Internet environment.

## 4.4 Conclusion

In this chapter, I conduct thorough security analysis to show the security of my design. I also provide a set of prototype implementation over the well-known IAPR dataset to demonstrate the practical usage of my privacy-preserving distance comparison design *PL1C* and *PKLC* along with their enhanced modules *PL1C - RF* and *PKLC - RF*. Extensive experiments results show that my modules can achieve promising performance in terms of efficiency and accuracy while protecting the privacy of user in descriptor based image analysis tasks.

## Chapter 5

# Privacy Protection for Deep Learning Based Image Analysis

To continue with the second research direction on privacy-preserving deep learning based image analysis, I apply one of the most well-known deep learning models, convolutional neural network (CNN), in an edge computing setting as an example use case. In this case, the resource-constrained IoT device offloads its image to the edge device to get analyzed. My case scenario and problem formulation is the same as in the corresponding section of Chapter 2, in which the edge device follows the designated algorithm to assist the analysis but is curious about the content of the IoT uploaded image.

### 5.1 Introduction

With the recent advances in artificial intelligence, the integration of deep neural networks and IoT is receiving increasing attention from both academia and industry [86–88]. As

the representative of deep neural networks, convolutional neural network has been identified as an prevailing structure to enable a spectrum of intelligent IoT applications [89], including visual detection, smart security, audio analytics, health monitoring, infrastructure inspection, etc. In these applications, pre-trained CNN models are deployed on IoT devices, with which the corresponding *CNN inference tasks* can be executed when real-time application requests are initiated. Nevertheless, due to the high computation in the inference tasks, deploying CNNs on resource-constrained IoT devices for time-sensitive services becomes very challenging. For example, popular CNN architectures (e.g., AlexNet [46], FaceNet [90], and ResNet [91]) for visual detection require billions of operations for the execution of a single inference task. My evaluation results show that a single inference task using AlexNet can cost more than two minutes on an IoT device with comparable computing capability as a Raspberry Pi (Model A).

To soothe IoT devices from heavy computation and energy consumption, offloading CNN inference tasks to public cloud computing platforms has become a popular choice in the literature. However, this type of “cloud-backed” system may raise privacy concerns by sending sensitive data to remote cloud [92]. Moreover, connecting to the cloud can cause additional latency to the system under network congestion and even make the system dysfunction when network is off [93]. While research efforts have been made towards enabling CNN inference over encrypted data using cloud computing [36–44], expensive cryptographic primitives utilized in them (e.g., homomorphic encryption and multi-party secure computation) introduce heavy encryption and communication overhead to IoT devices. Such a performance limitation makes these solutions far away from practical in support of time-sensitive CNN inference tasks on IoT devices, especially for complex CNN architectures. For example, a quad-core Raspberry Pi, which outperforms most resource-constrained IoT devices in terms of computational capability, can perform only

four Paillier homomorphic encryption per second [45]. Given a single input of AlexNet which has  $227 \times 227 \times 3$  elements, it requires more than 10 hours to complete the encryption, which is impractical for most applications in terms of time delay and energy consumption. Besides, these research adopt batch processing to improve their performance, which is more suitable for the “Data Collection and Post-Processing” routine, while differently, real-time processing is desired for IoT devices to fulfill time-sensitive tasks. To the best of my knowledge, enabling real-time execution of complex CNN inference tasks over encrypted data remains as an open problem.

Besides privacy-preserving real-time CNN inference offloading, one of the other challenges is regarding data integrity. Since the offloaded computation is resource consuming, the edge devices may not be willing to allocate expensive computational resources and may tend to cheat the IoT devices by returning random data with the same size of the desired data. According to [94], in some cases, dishonest edge service may even discard the data to save resources. Unfortunately, resource-constraint IoT devices are not able, or need to pay a considerable computational cost, to judge the correctness of the returned results from the cloud. How to effectively detect such dishonest behaviors while maintaining the lightweight computation on IoT devices and overall performance in time-sensitive CNN inference tasks is also an essential challenge to be solved.

In this chapter, I design privacy-preserving execution module for compute-intense layers to address such a challenging problem. I apply my designed modules in a time-sensitive deep learning based image analysis task to enable a real-time privacy-preserving CNN inference scheme for resource-constrained IoT devices. Different from existing “cloud-backed” designs, my design leverages edge computing to promote the efficiency of offloading IoT data, because it can effectively ameliorate the network latency and availability

issue [95]. More importantly, my privacy-preserving modules include a novel online/offline encryption to assure the real-time CNN inference over encrypted IoT data can be efficiently executed by general edge computing devices (e.g., regular laptop computers), and hence avoiding the reliance on powerful cloud servers for computing capabilities. To be specific, since linear operations of CNNs over input data and random noise are linearly separable, decryption of noise can be conveniently computed offline. In practical CNN architectures such as AlexNet and FaceNet, linear operations are dominant due to their vast number. Therefore, it is rewarding to trade offline computation and storage (of random noise) for online computation to assure the real-time performance of CNN inference tasks. Thanks to the online/offline encryption, IoT devices are able to securely offload over 99% CNN operations to edge devices. In addition, my scheme does not introduce any accuracy loss as compared to CNN inferences over unencrypted data, because it does not utilize any approximation for all required operations. In order to detect dishonest behaviors of edge devices, I design an integrity check mechanism as a pluggable module to help the IoT devices detect incorrect returned results from edge devices with a success rate over 99%. Minor computation overhead (1.1% drop of offload percentage in worst case) is introduced when this integrity check module is plugged. It is also worth to note that my scheme can be customized to support flexible CNN architectures that fulfill the requirements of different applications.

To further boost the efficiency of my scheme, two more pluggable modules can be integrated to enhance the performance in terms of convolution efficiency and communication cost reduction. Convolution operation, which generates the major computation consumption in a CNN model, consists of multiplications mostly. I investigate the fast convolution algorithm in [96] and integrate it to speed up the convolution cost by  $26.89\times$

for AlexNet. I also leverage the fast and efficient floating-point data compression algorithm in [97] and utilize it to reduce my communication cost by over 72% in AlexNet and FaceNet.

## 5.2 Related Work

The problem of privacy-preserving deep learning based image analysis has been studied in recent years under the cloud computing environment [36–44]. These works focus on the “machine learning as a service” scenario, wherein the cloud server has a trained neural network model and users submitted encrypted data for predication. One recent line of research uses somewhat or fully homomorphic encryption (HE) to evaluate the neural network model over encrypted inputs after approximating non-linear layers in the neural network [36–38]. Combining multiple secure computation techniques (e.g., HE, Secure multi-party computation (MPC), oblivious transfer (OT)) is another trend to support privacy-preserving neural network inference [39, 41, 42, 44]. The idea behind these mixed protocols is to evaluate scalar products using HE and non-linear activation functions using MPC techniques. In particular, SecureML [39] utilized the mixed-protocol framework proposed in ABY [42], which involves arithmetic sharing, boolean sharing, and Yao’s garbled circuits, to implement both privacy-preserving training and inference in a two-party computation setting. In [44], MiniONN is proposed to support privacy-preserving inference by transforming neural networks to the corresponding oblivious version with the Single Instruction Multiple Data (SMID) batch technique. Trusted third-party is invoked in Chameleon [43] and hence greatly reducing the computation and bandwidth cost for a privacy-preserving inference. In [41] GAZELLE is proposed

by leveraging lattice-based packed additive homomorphic encryption (PAHE) and two-party secure computation techniques. GAZELLE deploys PAHE in an automorphism manner to achieve fast matrix multiplication/convolution and thus boosting the final run-time efficiency. A multi-server solution, named SecureNN, is proposed in [40], which greatly improves the privacy-preserving inference performance, i.e.,  $42.4\times$  faster than MiniONN [44], and  $27\times$ ,  $3.68\times$  faster than Chameleon [43] and GAZELLE [41].

While the performance of evaluating neural network over encrypted data for image analysis keeps being improved, the existing research works only focus on small-scale neural networks. Taking the state-of-the-art SecureNN [40] as an example, the network-A evaluated (also used by [39]) only requires about 1.2 million FLOPs for an inference, which costs 3.1s with wireless communication in their 3PC setting. As a comparison, the AlexNet evaluated in my scheme contains 2.27 billion FLOPs for one inference, which costs 3.508s in my scheme with similar wireless transmission speed. It is also worth to note that SecureNN utilizes powerful cloud server (36 vCPU, 132 ECU, 60GB memory) for evaluation, whereas the edge computing device in this paper is just a regular laptop computer. Scaling up the network size is not a trivial task. For example, compared with the type-A network in [41], its type-C network with  $500\times$  multiplication increases the computational cost and communication cost to  $430\times$  and  $592\times$ . Therefore, how to support real-time execution of complex CNN inference tasks over encrypted IoT data remains as an open problem.

### 5.3 Preliminaries - Convolutional Neural Network

A CNN contains a stack of layers that transform input data to outputs with label scores. There are four types of most common layers in CNN architectures, including:



*Convolutional Layers, Pooling Layers, Activation Layers, and Fully-connected Layers.*

Convolutional layers extract features from input data. Fig.5.1 depicts an example of convolutional layer that has an input data of size  $n \times n \times D$  and  $H$  kernels, each of size  $k \times k \times D$ . The input will be processed into all  $H$  kernels independently to extract  $H$  different features. Considering the input and each kernel as  $D$  levels, where each level of the input and kernel are a  $n \times n$  matrix and a  $k \times k$  matrix respectively. Each level of a kernel starts scanning the corresponding input level from top-left corner, and then moves from left to right with  $s$  elements, where  $s$  is the stride of the convolutional layer. Once the top-right corner is reached, the kernel moves  $s$  elements downward and scans from left to right again. This convolution process is repeated until the kernel reaches the bottom-right corner of this input level. For each scan, an output is computed using the dot product between the scanned window of input and the kernel as an example shown in Fig.5.1. For each kernel, the output for all  $D$  levels will be summed together.

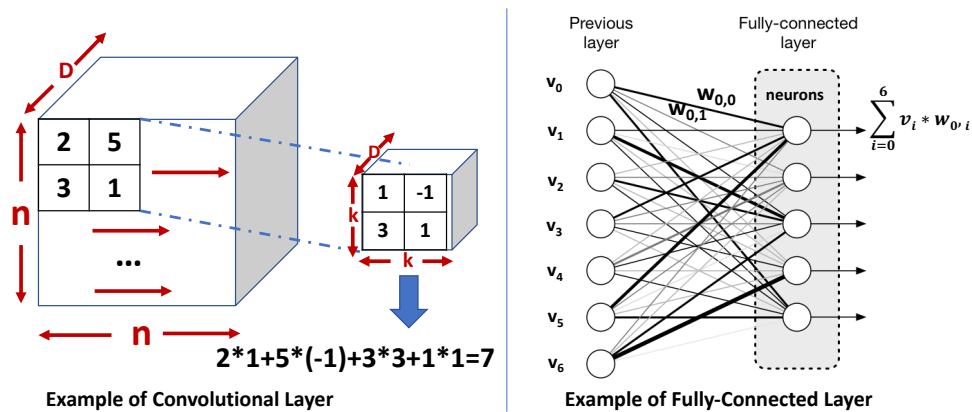


FIGURE 5.1: Examples of a Convolutional Layer and a Fully-connected Layer

Pooling layers and activation layers are usually non-linear layers. A pooling layer is periodically inserted between convolutional layers. Pooling layers progressively reduce the spatial size of outputs from convolutional layers, and thus to make data robust against noise and control overfitting. An activation layer utilizes element-wise activation functions to signal distinct identification of their input data. There are a number of popular

pooling strategies (e.g., max-pooling and average-pooling) and activation functions (e.g., rectified linear units (ReLUs) and continuous trigger functions), which are extremely computational efficient compared with convolutional layers and fully-connected layers. In my scheme, these two efficient layers will be directly handled on the IoT devices.

Fully-connected layers are usually the final layers of a CNN to output the final results of the network. In case of a fully-connected layer, all neurons in it have full connections to all outputs from the previous layer. As an example shown in Fig.5.1, the connection between each neuron and input element has a weight. To obtain the output of a neuron, elements connected to it will be multiplied with their weights and then accumulated.

More details about CNN can be found in ref [98].

## 5.4 Privacy-preserving Compute-intense Layers

In my scheme, the user needs to offload compute-intense convolution and fully-connected layer to the edge device. For the purpose of privacy protection, the offloaded data is encrypted. To enable accurate and efficient convolution/fully-connected layer execution over encrypted data, I design privacy-preserving convolution layer (*PPCL*) module and privacy-preserving fully-connected layer (*PPFL*) module respectively. Before I dive into the details of my modules. I summarize important notation in Table 5.1.

### 5.4.1 *PPCL*: Privacy-preserving Convolutional Layer

In *PPCL*, I consider a general convolutional layer with a  $n \times n \times D$  input, stride as  $s$ , padding as  $p$ , and  $H$  kernels with each size of  $k \times k \times D$ . The  $d_{th}$  level of the input is denoted as a  $m \times m$  matrix  $\mathcal{I}_d$ .

TABLE 5.1: Summary of Notations

$n \times n$	the size of each level of a convolutional layer's input
$D$	the depth of the input of a convolutional layer and kernel
$k \times k$	the size of a convolutional layer's kernel matrix
$H$	the number of kernels of a convolutional layer
$s$	the size of stride used for a convolutional layer
$p$	the size of padding used for a convolutional layer
$\mathcal{R}_{c,d}$ $1 \leq d \leq D$	$n \times n$ random matrices to encrypt the input of a convolutional layer
$\alpha_i$ $1 \leq i \leq H$	$\left(\frac{n-k+2p}{s} + 1\right) \times \left(\frac{n-k+2p}{s} + 1\right)$ matrices to decrypt a convolutional layer's output from $H$ kernels
$m$	the size of a fully-connected layer input vector
$T$	the number of neurons of a fully-connected layer
$\mathcal{R}_f$	a $m$ -dimensional random vector to encrypt the input of a fully-connected layer
$\beta$	a $T$ -dimensional decryption vector to decrypt fully-connected layer outputs

**Input Encryption:** The IoT device encrypts the input using the pre-stored keys  $\{\mathcal{R}_{c,d}\}$  for this convolutional layer as

$$Enc(\mathcal{I}_d) = \mathcal{I}_d + \mathcal{R}_{c,d} \quad (5.1)$$

where  $\{Enc(\mathcal{I}_d)\}, 1 \leq d \leq D$  are sent to the edge device.

**Privacy-preserving Execution:** The edge device takes each  $Enc(\mathcal{I}_d), 1 \leq d \leq D$  as the input of kernels to perform the convolution process. For the  $i_{th}$  kernel, the edge device outputs

$$\begin{aligned} & \sum_{d=1}^D \mathbf{Conv}(Enc(\mathcal{I}_d), i_{th}) \\ &= \sum_{d=1}^D \mathbf{Conv}(\mathcal{I}_d, i_{th}) + \sum_{d=1}^D \mathbf{Conv}(\mathcal{R}_{c,d}, i_{th}) \end{aligned} \quad (5.2)$$

$\sum_{d=1}^D \mathbf{Conv}(Enc(\mathcal{I}_d), i_{th}), 1 \leq i \leq H$  are returned back to the IoT device as intermediate results.

**Decryption and Preparation for the Next Layer:** Given the returned

$\sum_{d=1}^D \mathbf{Conv}(Enc(\mathcal{I}_d), i_{th}), 1 \leq i \leq H$ , the IoT device quickly decrypts them as

$$\sum_{d=1}^D \mathbf{Conv}(Enc(\mathcal{I}_d), i_{th}) - \alpha_i = \sum_{d=1}^D \mathbf{Conv}(\mathcal{I}_d, i_{th}) \quad (5.3)$$

where  $\{\alpha_i = \sum_{d=1}^D \mathbf{Conv}(\mathcal{R}_{c,d}, i_{th})\}, 1 \leq i \leq H$  are the pre-stored decryption keys for this layer. Afterwards, the IoT device performs the activation layer and pooling layer directly over convolutional output, which are extremely compute-efficient. For example, one of the most popular activation layer ReLU only requires translating negative values in the input to 0. The popular max-pooling (or average-pooling) layer simply shrinks the data by taking the max value (or average value respectively) every few values. The output will be encrypted and sent to the edge device using *PPCL* for the next convolutional layer (or *PPFL* respectively for a fully-connected layer).

#### 5.4.2 *PPFL*: Privacy-preserving Fully-connected Layer

In *PPFL*, I consider a general fully-connected layer with  $T$  neurons and takes a  $m$ -dimensional vector  $\mathcal{V}$  as input.

**Input Encryption:** Given the input, the IoT device encrypts it using the pre-stored encryption key  $\mathcal{R}_f$  for this layer as

$$Enc(\mathcal{V}) = \mathcal{V} + \mathcal{R}_f \quad (5.4)$$

$Enc(\mathcal{V})$  is then sent to the edge device.

**Privacy-preserving Execution:** On receiving  $Enc(\mathcal{V})$ , the edge device takes  $Enc(\mathcal{V})$  as the input of the fully-connected layer. Specifically, the encrypted outcome  $Enc(\mathcal{O}[j])$ ,

$1 \leq j \leq T$  of each neuron is computed as

$$Enc(\mathcal{O}[j]) = \sum_{i=1}^m Enc(\mathcal{V})[i] \times w_{i,j} = \mathcal{O}[j] + \beta[j] \quad (5.5)$$

where  $w_{i,j}$  is the weight between the  $i_{th}$  element of input vector and the  $j_{th}$  neuron.

$Enc(\mathcal{O}) = \{Enc(\mathcal{O}[1]), Enc(\mathcal{O}[2]), \dots,$

$Enc(\mathcal{O}[T])\}$  is sent back to the IoT device as intermediate results.

**Decryption and Preparation for the Next Layer:** Given the returned  $Enc(\mathcal{O})$ ,

the IoT device decrypts each  $Enc(\mathcal{O})$  with the pre-stored decryption key  $\beta$  of this layer

as

$$\mathcal{O} = Enc(\mathcal{O}) - \beta \quad (5.6)$$

Then, the IoT device executes the activation layer with  $\mathcal{O}$  as input. The output from the activation layer will be encrypted and sent to edge device using *PPFL* if there are any additional fully-connected layers in the CNN.

## 5.5 Edge-Assisted CNN Inference over Encrypted Imagery

### Data

In this section, I demonstrate the practical application of my *PPCL* and *PPFL* module in an edge-assisted IoT image analysis task. To assure the real-time performance of the image analysis tasks, I use a novel online/offline strategy to design my scheme. The online phase refers to the duration when a CNN inference is being executed for the data collected by an IoT device. The offline phase refers to the “no-inference” status

of the IoT device and time before the IoT device is deployed. Specifically, the owner of an IoT device pre-computes multiple sets of encryption and decryption keys during the offline phase and loads them into the IoT device. In the online phase, the IoT device uses these pre-computed keys to efficiently encrypt data to be offloaded and decrypt results returned by the edge device. In my system, the IoT device offloads expensive convolutional layers and fully-connected layers to the edge device, and only keeps the compute-efficient layers at local. This is motivated by the fact that convolutional and fully-connected layers occupy majority of computation and parameters storage in typical CNNs [99]. All CNN operations performed by the edge device are over encrypted imagery data.

### 5.5.1 Offline Phase

In the offline phase, the user generates encryption and decryption keys for all convolutional layers and fully-connected layers in a trained CNN. I consider each element in the input image matrix of convolutional layers and fully-connected layers is  $\gamma$ -bit long, and  $\lambda$  is the security parameter. To ensure the security  $\frac{1}{2^{\lambda-\gamma-1}}$  shall be a negligible value in terms of computational secrecy [100], e.g.,  $< \frac{1}{2^{128}}$ . Detailed selection of security parameter is discussed in Section 6.1.1.

As described in Algorithm.2, given a convolutional layer with a  $n \times n \times D$  input, stride as  $s$ , padding as  $p$ , and  $H$  kernels ( $k \times k$  matrices), the owner generates  $\{\mathcal{R}_{c,d}, 1 \leq d \leq D\}$  as the encryption keys and  $\{\alpha_i, 1 \leq i \leq H\}$  as its decryption keys, where  $\mathcal{R}_{c,d}$  is a  $n \times n$  random matrix and  $\alpha_i$  is a  $(\frac{n-k+2p}{s} + 1) \times (\frac{n-k+2p}{s} + 1)$  matrix. For expression simplicity, I use  $\mathbf{Conv}(\mathcal{R}_{c,d}, i_{th})$  to denote the convolution operation for the  $i_{th}$  kernel with  $\mathcal{R}_{c,d}$  as input.

Given a fully-connected layer with a  $m$ -dimensional vector as input and  $T$  neurons, the owner first generates a  $m$ -dimensional random vector  $\mathcal{R}_f$ . Then, the owner takes  $\mathcal{R}_f$  as the input of the fully-connected layer to output a  $T$ -dimensional vector  $\beta$ .  $\mathcal{R}_f$  and  $\beta$  are set as the encryption key and decryption key respectively for this layer.

For a CNN with  $x$  convolutional layers and  $y$  fully-connected layers,  $x$  sets of  $\{\mathcal{R}_{c,d}, \alpha_i\}_{1 \leq d \leq D, 1 \leq i \leq H_x}$  and  $y$  sets of  $\{\mathcal{R}_f, \beta\}$  are generated by the owner as a final set of keys  $\{Enc_{key}, Dec_{key}\}$ . **Note that**, each set of keys is only valid for one CNN request in the later online phase. Thus, the owner will generate multiple sets of keys according to the necessity of application scenarios as discussed in Section 5.6.

**Input** : Input size  $n \times n \times D$ , stride  $s$ , padding  $p$ ,  $H$  kernels  
**Output**: Encryption keys  $\mathcal{R}_{c,d}$ ,  $1 \leq d \leq D$ , Decryption keys  $\alpha_i$ ,  $1 \leq i \leq H$   
Generate random  $n \times n$  matrices  $\mathcal{R}_{c,d}$ ,  $1 \leq d \leq D$  ;  
**for**  $1 \leq i \leq H$  **do**  
    **for**  $1 \leq d \leq D$  **do**  
        Take  $\mathcal{R}_{c,d}$  as input for the  $i_{th}$  kernel for convolution and output  $\mathbf{Conv}(\mathcal{R}_{c,d}, i_{th})$ ;  
         $d++$ ;  
    **end**  
    Set  $\alpha_i = \sum_{d=1}^D \mathbf{Conv}(\mathcal{R}_{c,d}, i_{th})$ ;  
     $i++$ ;  
**end**

**Algorithm 2:** Offline Preparation of Convolutional Layer

**Input** : Input Data & Trained CNN  
**Output**: CNN Execution Result  
**Set** the Layer Input  $\mathcal{M} =$  Input Data;  
**Set** Layers = the collection of all Convolutional Layers and Fully-connected Layers in CNN;  
**Set** Layer = the first Layer from Layers;  
**while** Layer is not null **do**  
    **if** Layer = Convolutional Layer **then**  
        Execute the *PPCL* with  $\mathcal{M}$  as input.;  
        Set  $\mathcal{M} =$  output from *PPCL*;  
    **end**  
    **if** Layer = Fully-connected Layer **then**  
        Execute the *PPFL* with  $\mathcal{M}$  as input.  
        Set  $\mathcal{M} =$  output from *PPFL*;  
    **end**  
    **Set** Layer = Layers.next();  
**end**  
**return**  $\mathcal{M}$  as result;

**Algorithm 3:** Online CNN Inference

### 5.5.2 Online Phase

During the online phase, the IoT device can efficiently interact with the edge device to conduct CNN inference over encrypted data. The overall process of my online phase is depicted in Algorithm.3. Specifically, the IoT device offloads encrypted data to the edge device for performing compute-intense convolutional layers and fully-connected layers. Intermediate results are returned back to the IoT device for decryption. Then, these decrypted results are processed with the follow up activation layer and pooling layer (if exists). Outputs are encrypted and offloaded again if the next layer is a convolutional layer or a fully-connected layer. This procedure is conducted iteratively until all CNN layers are executed.

To fulfill these tasks, I apply *PPCL* and *PPFL* to enable the IoT device to efficiently handle each layer in a CNN. Compute-intense convolutional and fully-connected layers are securely offloaded to the edge using *PPCL* and *PPFL*. Since I develop *PPCL* and *PPFL* as independent modules, they can be customized and recursively plugged into any CNN no matter how many different convolutional layers and fully-connected layers it contains.

## 5.6 Discussion - Storage and Update of Pre-computed Keys

Our scheme considers two major types of resource-constrained IoT devices that run CNN-driven applications.

- Type-1: Mobile IoT devices with limited battery life and computational capability, such as drones.



- Type-2: Static devices with power supply but has limited computational capability, such as security cameras.

The type-1 devices are usually deployed to perform tasks for a period time. Therefore, before each deployment, the device owner can pre-load enough keys to support its CNN tasks. With regards to the type-2 devices, the owner can perform an initial key pre-loading and then use remote update to securely add new offline keys as described in Fig.5.2.

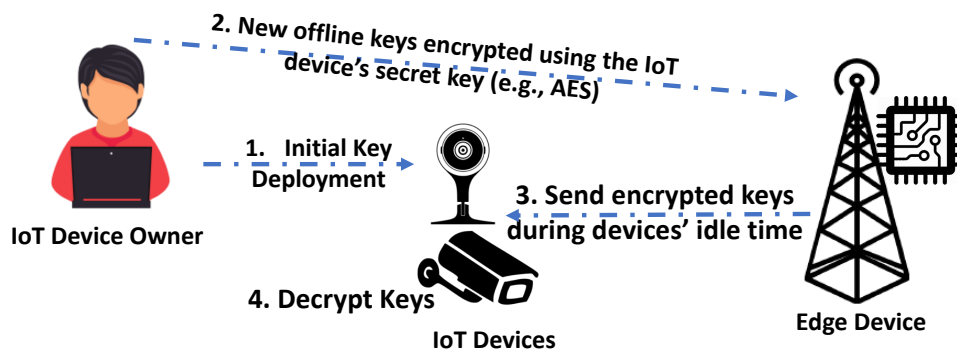


FIGURE 5.2: Key Update for Power Connected Devices

Our scheme proposes to ensure the timely processing of CNN requests when they are needed on IoT devices. Instead of performing real-time CNN requests on every piece of data collected, resource-constrained IoT devices usually require in-depth analytics using CNN when specific signals are detected. Taking real-time search and monitoring using drones as an example application for type-1 devices, fast local processing will be first performed for data collected to get estimated results [101]. Once suspicious signs are detected in estimated results, CNN based analytics are further conducted for a small set of data (e.g., video frames with the detected suspicious object). Given the high efficiency of my scheme, the performance of such CNN requests will be timely supported when enough pre-computed keys are available. For example, when the average frequency CNN requests is every one per ten seconds for a drone, only 360 sets of pre-computed keys

are needed for one-hour deployment, which is longer than most current drones' battery life [102]. Security camera is an example of type-2 devices, which requires CNN-based analytics to extract detailed information only when alarm is triggered by motion or audio sensors of the camera. Similar to the drone case, my scheme can timely support the peak CNN requests when suspicious signs are detected.

Assuming the average frequency of CNN-required alarm in a security camera is one per 10 minutes, and each alarm requires 5 CNN requests, 720 sets of pre-computed keys are needed for one-day usage. As evaluated in Section 6.1.2.3, an IoT device with a 32GB SD card is able to store enough keys to support 1600 requests for AlexNet. Such a result indicates 4.4 deployments and a 2.22-day support for type-1 and type-2 devices respectively when using AlexNet.

Note that, my scheme is designed for low-cost resource-constrained devices that require timely processing of moderate (or low) frequent CNN requests. For application scenarios that involve a large number of constant CNN requests, e.g., security critical surveillance systems, computational powerful devices are suggested to handle CNN requests directly at local.

## 5.7 Discussion - Offloading Pooling Layers

While pooling layers are usually compute-efficient in a CNN, they can also be securely offloaded to edge devices using my online/offline strategy to further promote the efficiency. Particularly, I focus on the offloading of average pooling layer as the example shown in Fig.5.3, which performs down-sampling by dividing the input into rectangular pooling regions and computing the average values of each region. Given a  $n \times n \times D$

matrix as input to an average pooling layer with  $q \times q$  pooling regions and  $s$  as stride, a  $(\frac{n-q}{s} + 1) \times (\frac{n-q}{s} + 1)$  matrix is generated as output.

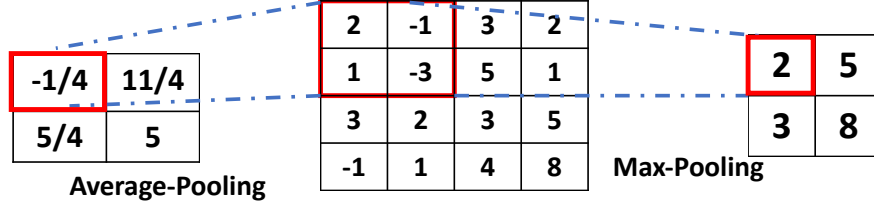


FIGURE 5.3: Example of Pooling Layers

To enable the offloading of an average pooling layer, the following changes will be made for the offline phase and online phase:

- **Offline Phase:** For each use of the average pooling layer, the IoT device owner generates  $D$  random matrices  $\{\mathcal{R}_{p,d}\}$  with each size of  $n \times n$ , and processes it with the average pooling layer to obtain  $(\frac{n-q}{s} + 1) \times (\frac{n-q}{s} + 1)$  matrices  $\{\mathcal{R}_{c,d}\}$ . The owner then runs Algorithm.2 using  $\{\mathcal{R}_{c,d}\}$  as encryption keys to generate  $\{\alpha_i\}$ .
- **Online Phase:** The online offloading of an average pooling layer can be combined with its follow up convolutional layer. Specifically, the IoT device encrypts each level of input  $\mathcal{I}_d$  for the average pooling layer as  $Enc(\mathcal{I}_d) = \mathcal{I}_d + \mathcal{R}_{p,d}$ , and sends  $Enc(\mathcal{I}_d)$  to the edge device. The edge processes the pooling layer with  $Enc(\mathcal{I}_d)$  as input and output  $Enc(\mathcal{O}_{p,d})$ . Then, the edge executes the convolutional layer with  $\{Enc(\mathcal{O}_{p,d})\}_{1 \leq d \leq D}$  as input and outputs  $Enc(\mathcal{O}_i), 1 \leq i \leq H$  for  $H$  kernels. Finally, the IoT device can decrypt  $\{Enc(\mathcal{O}_i)\}$  using  $\{\alpha_i\}$  as  $\mathcal{O}_i = Enc(\mathcal{O}_i) - \alpha_i, 1 \leq i \leq H$ .

It is worth to note that my design combines the offloading of the average pooling layer and its follow up convolutional layer. Thus, the IoT device only needs to encrypt the input of pooling layer and decrypt the outputs from the convolutional layers using efficient

matrix additions.

I now analyze the correctness for offloading average-pooling layer and convolutional layer together. By denoting the average-pooling operation as  $AvgP(\cdot)$ , I have the fact that

$$AvgP(Enc(\mathcal{I}_d)) = AvgP(\mathcal{I}_d) + AvgP(\mathcal{R}_{p,d})$$

In addition, according to the correctness of *PPCL* as discussed in Section 5.4.1, I have

$$\begin{aligned} & \sum_{d=1}^D (Conv_i(AvgP(\mathcal{I}_d) + AvgP(\mathcal{R}_{p,d}))) \\ &= \sum_{d=1}^D Conv_i(AvgP(\mathcal{I}_d)) + \sum_{d=1}^D Conv_i(AvgP(\mathcal{R}_{p,d})) \end{aligned}$$

where  $Conv_i(\cdot)$  is the processing of  $i_{th}$  kernel in the convolutional layer. Since  $\alpha_i = \sum_{d=1}^D Conv(AvgP(\mathcal{R}_{p,d}))$ , I get

$$\mathcal{O}_i = Enc(\mathcal{O}_i) - \alpha_i = \sum_{d=1}^D Conv_i(AvgP(\mathcal{I}_d)), 1 \leq i \leq H \quad (5.7)$$

Therefore, after the decryption, the IoT device can obtain the correct output for the average-pooling layer and convolutional layer.

## 5.8 Enhancement - Integrity Check

Due to the heavy computation and storage overhead brought by convolutional and fully-connected layers, untrusted edge devices may perform dishonest behaviors so that they can save their resource utilities. After receiving inference requests from the IoT devices, the edge devices may cheat the IoT devices by skipping the heavy convolutional operations and sending back random, apparently not correct, results to the IoT devices.

These incorrect results can badly effect or even completely ruin the final result of the entire CNN inference. In order to ensure the integrity of returned data from edge devices, my scheme also provides an optional integrity check functionality with only a minor efficiency cost. By enabling the integrity check, the IoT device can achieve an error detection rate of 99% while only losing 1.1% in offload percentage in the worst case compared with integrity check plugged in. The users can decide whether to turn on this functionality based on the actual deployment scenario and the trustworthiness of the edge devices.

The basic strategy of integrity check is to first sample a small portion of elements from returned data in each layer and then check whether there is an incorrect result occurring in the selected elements. To validate the correctness of a single element, the IoT device needs to go through the corresponded convolution operations locally. Although resource consuming, with high probability, this validation process can block IoT devices from taking incorrect results into next layer.

The error detection rate  $Pr(ED)$  can be calculated as below:

$$Pr(ED) = 1 - \frac{\binom{(1-\theta)N}{rN}}{\binom{N}{rN}} \quad (5.8)$$

where  $N$ ,  $r$  and  $\theta$  is the size, sample rate and error rate of returned data in a convolutional layer.  $\binom{(1-\theta)N}{rN}$  is the combination operation for selecting  $r \times N$  elements from  $(1 - \theta) \times N$  elements. In order to increase the error detection rate while lowering the additional validation computation on IoT device, I provide detailed evaluation by performing numerical analysis and practical experiments in Section 5.8 and Section 6.2 respectively. Based on my experiment results on AlexNet, my scheme with integrity

check turned on could achieve over 99% error detection rate while maintaining a similar computation offload rate compared with unplugging integrity check.

## 5.9 Enhancement - Fast Convolution

In order to improve the efficiency of the compute-intense convolutional layer, I investigate and integrate the fast convolution algorithm in [96]. Given matrix  $\mathcal{I}^{n \times n}$  as convolutional layer input in one channel and matrix  $\mathcal{K}^{k \times k}$  as one filter of that convolutional layer, an original convolution filtering  $\mathbf{Conv}(\mathcal{I}, \mathcal{K})$  requires  $(n - k + 1)^2 k^2$  element-wise multiplication (set stride = 1 and padding = 0 for simplification). Take  $n = 4, k = 3$  as an example, the convolutional layer output is a  $2 \times 2$  matrix. An original convolutional computation would cost  $(n - k + 1)^2 k^2 = 2^2 \times 3^2 = 36$  multiplications. The multiplication number can be minimized by applying the fast filtering algorithm in [96, 103] to transform  $\mathbf{Conv}(\cdot)$  into:

$$\mathbf{Conv}(\mathcal{I}, \mathcal{K}) = [A^T [[G\mathcal{K}G^T] \odot [B^T \mathcal{I}B]] A]^T \quad (5.9)$$

where  $\odot$  represents element-wise multiplication and:

$$\begin{aligned}
B^T &= \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} & G &= \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix} \\
A^T &= \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix} & & (5.10) \\
\mathcal{K} &= \begin{bmatrix} g_0 & g_1 & g_2 \\ g_3 & g_4 & g_5 \\ g_6 & g_7 & g_8 \end{bmatrix}^T & \mathcal{I} &= \begin{bmatrix} d_0 & d_1 & d_2 & d_3 \\ d_4 & d_5 & d_6 & d_7 \\ d_8 & d_9 & d_{10} & d_{11} \\ d_{12} & d_{13} & d_{14} & d_{15} \end{bmatrix}^T
\end{aligned}$$

in which  $B$ ,  $G$  and  $A$  are auxiliary coefficient matrices. Details of generating these matrices are stated in [103]. It can be easily proved that Equation 5.12 achieves equilibrium by applying values from Equation 5.10. Note that under my scenario, the trained CNN does not require frequent update and  $GKG^T$  can be precomputed once the CNN is deployed on the edge servers. Moreover, operations with  $A$  and  $B$  involved are considered as additions instead of multiplications because both  $A$  and  $B$  are composed of 0, 1 and -1. Thus, the total number of multiplications  $|(GKG^T) \odot [B^T \mathcal{I} B]| = 16$  is calculated by counting the participating elements in the element-wise multiplication.

To be more generic, according to the 1-D minimal filtering algorithm in [103, 104], computing  $m$ -dimension outputs with an  $r$ -dimension filter, denoted as  $F(m, r)$ , requires  $\mu(F(m, r)) = m + r - 1$  multiplications. By nesting this 1-D algorithm with itself, a 2-D version can be achieved as  $\mu(F(m \times m, r \times r)) = \mu(F(m, r)) \times \mu(F(m, r))$ . Thus, in my scenario, during a convolution filtering, the minimal number of multiplications required for computing  $(n - k + 1) \times (n - k + 1)$  outputs with  $k \times k$  filter can be derived as:

$$\begin{aligned}
\mu(F((n-k+1)^2, k^2)) &= \mu(F(n-k+1, k))^2 \\
&= (n-k+1+k-1)^2 \\
&= n^2
\end{aligned} \tag{5.11}$$

This is also consistent with the above example where  $n = 4, k = 3$  and the overall number of multiplications is 16. In my model, the plaintext input  $\mathcal{I}_\Gamma$ , key  $\mathcal{R}_{c,d}$  and encrypted input  $\mathcal{I}_\Gamma + \mathcal{R}_{c,d}$  share the same size and since all the operations in the fast filtering algorithm are linear operations, my scheme can perfectly enhance its efficiency by applying  $\mathcal{I}_\Gamma$  and  $\mathcal{R}_{c,d}$  in Equation 5.12 and obtain:

$$\begin{aligned}
&\mathbf{Conv}(\mathcal{I}_d + \mathcal{R}_{c,d}, \mathcal{K}) \\
&= [A^T [[G\mathcal{K}G^T] \odot [B^T(\mathcal{I}_d + \mathcal{R}_{c,d})B]] A]^T \\
&\mathbf{Conv}(\mathcal{R}_{c,d}, \mathcal{K}) = [A^T [[G\mathcal{K}G^T] \odot [B^T\mathcal{R}_{c,d}B]] A]^T
\end{aligned} \tag{5.12}$$

By integrating the fast filtering algorithm, my scheme can reduce the multiplication overhead by  $(\frac{(n-2p+k)k}{sn})^2$  times in each convolutional layer. And since the fast filtering algorithm can yield exactly the same result as the original convolution, there is no accuracy loss introduced.

## 5.10 Enhancement - Matrix Compression

In order to save communication overhead in my design, I investigate and integrate the well-known fast and efficient float data compression [97] before transmitting data between edge devices and IoTs. [97] proposed an efficient compressor for both 32-bit and 64-bit images. by placing emphasis on both 2D and 3D data for rendering. The compression algorithm leverages the Lorenzo predictor [105] to predict the data and meanwhile,



it also utilizes a Schindler's quasistatic probability model [106] based range coder to encode the residual. Details of the fast and efficient float data compression is stated in [97].

## 5.11 Conclusion

In this chapter, I proposed a novel online/offline scheme that enables resource-constrained IoT devices to efficiently execute CNN requests with privacy protection. My scheme uniquely designs a lightweight online/offline encryption scheme to provide private, efficient and accurate offloading of CNN inference tasks. By discovering the fact that linear operations in CNNs over input and random noise can be separated, my scheme pre-computes decryption keys to remove random noises and thus boosting the performance of real-time CNN requests. By integrating local edge devices, my scheme ameliorates the network latency and service availability issue. My scheme also makes the privacy-preserving operation over encrypted data on the edge device as efficient as that over unencrypted data. Moreover, the privacy protection in my scheme does not introduce any accuracy loss to the CNN inference. In addition, I design a few enhancement modules to bring more features and improve the performance of my scheme in terms of efficiency and data integrity protection. In the next chapter, I will analyze the security and performance of my design followed by experiment results on my prototype implementation.

## Chapter 6

# Evaluation of Privacy Protection Modules for Deep Learning Based Image Analysis

In this chapter, I first theoretically audit the security and performance of my design in my application scenario. Then I provide experimental results to evaluate the practical performance of my privacy protection modules as well as their enhancement for deep learning based image analysis stated in Chapter 5.

### 6.1 Security and Performance Analysis

#### 6.1.1 Security Analysis

In this section, I first prove the security of my online/offline encrypted used in *PPCL* and *PPFL*, and then show the security of the overall CNN inference in my setting as

described in Chapter 2 Section 2.

**Theorem 6.1.** *Given the ciphertext  $\mathcal{C}$  of a  $\gamma$ -bit random message  $\mathcal{M}$  generated using PPCL or PPFL, the probability for a probabilistic polynomial time (PPT) adversary  $\mathcal{A}$  to output a correct guess for  $\mathcal{M}$  shall have*

$$Pr[(\mathcal{M}^* = \mathcal{M})|\mathcal{C}] - Pr[\mathcal{M}^* = \mathcal{M}] \leq \epsilon \quad (6.1)$$

where  $\epsilon$  is a negligible value in terms of computational secrecy [100],  $\mathcal{M}^*$  is  $\mathcal{A}$ 's guess for  $\mathcal{M}$ , and  $Pr[\mathcal{M}^* = \mathcal{M}]$  is the probability  $\mathcal{A}$  makes a correct without ciphertext. Specifically, the corresponding ciphertext generated using PPCL or PPFL only introduces negligible additional advantages to  $\mathcal{A}$  for making a correct guess of  $\mathcal{M}$ .

*Proof.* Given an input matrix  $\mathcal{I}_d$  in PPCL (input vector  $\mathcal{V}$  in PPFL respectively), each  $\gamma$ -bit element ( $\mathcal{M}$ ) is encrypted by adding a  $\lambda$ -bit random number from uniform distribution (denoted as  $R$ ) as shown in Eq.5.1 and Eq.5.4.

To make a correct guess of  $\mathcal{M}$  without the ciphertext, the adversary  $\mathcal{A}$  has  $Pr[\mathcal{M}^* = \mathcal{M}] = \frac{1}{2^\gamma}$ , where  $\mathcal{M}^*$  is  $\mathcal{A}$ 's guess for  $\mathcal{M}$ . By given the ciphertext  $\mathcal{C} = \mathcal{M} + R$ ,  $\mathcal{C} \in [0, 2^\gamma + 2^\lambda]$  there are two cases according to the value of  $\mathcal{C}$

1.  $2^\gamma \leq \mathcal{C} \leq 2^\lambda$ . In this case, I have  $Pr[\mathcal{M}^* = \mathcal{M}] = \frac{1}{2^\gamma}$ , since  $\mathcal{C}$  has the uniform looking as  $R$  in the range of  $[2^\gamma, 2^\lambda]$ .
2.  $\mathcal{C} < 2^\gamma$  or  $\mathcal{C} > 2^\lambda$ . In this case, I have  $Pr[\mathcal{M}^* = \mathcal{M}] = \frac{1}{\mathcal{C}}$  or  $Pr[\mathcal{M}^* = \mathcal{M}] = \frac{1}{\mathcal{C} - 2^\lambda}$ , where  $Pr[\mathcal{M}^* = \mathcal{M}] > \frac{1}{2^\gamma}$ . This is because the distribution of  $\mathcal{C}$  is affected by  $\mathcal{M}$  and the total possible inputs are now reduced to  $\mathcal{C}$  or  $\mathcal{C} - 2^\lambda$ .

The second case can happen when  $R < 2^\gamma$  or  $R > 2^\lambda - 2^\gamma$ . As  $Pr[R < 2^\gamma] = Pr[R > 2^\lambda - 2^\gamma] = \frac{2^\gamma}{2^\lambda}$ , I have

$$Pr[R < 2^\gamma \text{ or } R > 2^\lambda - 2^\gamma] = Pr[R < 2^\gamma] + Pr[R > 2^\lambda - 2^\gamma] = \frac{1}{2^{\lambda-\gamma-1}}$$

Thus, to guarantee  $\frac{1}{2^{\lambda-\gamma-1}}$  is a negligible probability, such as  $\frac{1}{2^{128}}$ , my scheme can set the security parameter  $\lambda$  according to size of input message, i.e.,  $\lambda - \gamma - 1 > 128$ . I now use  $\epsilon = \frac{1}{2^{\lambda-\gamma-1}}$  to denote the negligible probability, and get the probability  $Pr[(\mathcal{M}^* = \mathcal{M})|\mathcal{C}]$  as

$$Pr[(\mathcal{M}^* = \mathcal{M})|\mathcal{C}] \leq \frac{1}{2^\gamma} * (1 - \epsilon) + 1 * \epsilon = \frac{1}{2^\gamma} + (1 - \frac{1}{2^\gamma})\epsilon$$

where  $\frac{1}{2^\gamma} * (1 - \epsilon)$  is the probability for a correct guess for  $2^\gamma \leq C \leq 2^\lambda$ , and the “1” in  $1 * \epsilon$  is best probability for a correct guess  $\mathcal{A}$  can have when  $C < 2^\gamma$  or  $C > 2^\lambda$ . As a result, I get

$$Pr[(\mathcal{M}^* = \mathcal{M})|\mathcal{C}] - Pr[\mathcal{M}^* = \mathcal{M}] \leq (1 - \frac{1}{2^\gamma})\epsilon < \epsilon$$

As  $\epsilon$  is negligible value, Theorem 6.1 is proved. □

I now discuss the security of the overall CNN inference. Without loss of generality, I use layer-x to denote a convolutional layer or a fully-connected layer needs to be offloaded,  $\mathcal{I}_x$  and  $\mathcal{O}_x$  are the input and output of layer-x. With regards to the offloading of layer-x,  $\mathcal{I}_x$  is encrypted using my online/offline encryption, which has been proved to be secure as shown in Theorem 6.1. When moving to the next layer, i.e., layer-(x+1),  $\mathcal{O}_x$  is processed through non-linear layers by the IoT device to generate the input of layer-(x+1) as  $\mathcal{I}_{(x+1)}$ . Before being offloaded, each element  $\mathcal{M}$  in  $\mathcal{I}_{(x+1)}$  is encrypted

by adding a random number  $R$  from uniform distribution. By selecting an appropriate security parameter  $\lambda$ , there will be only a negligible probability  $\frac{1}{2^{\lambda-\gamma-1}}$  that  $\mathcal{M}$  affects the uniform looking of ciphertext  $\mathcal{M}+R$  as proved in Theorem 6.1, where  $\gamma$  is the length of  $\mathcal{M}$  in bits. To be specific, by re-encrypting the input of each offloaded layer in my scheme, the negligible additional advantages introduced by each offloaded layer for the adversary to learn its input and output will not be accumulated for later layers in the CNN inference. Therefore, the security of the overall CNN inference is achieved in my scheme with proper selection of  $\lambda$ .

### 6.1.2 Numerical Analysis

The numerical analysis of my scheme is summarized in Table 6.1. For expression simplicity, I use one floating point operation (FLOP) to denote an addition or a multiplication. I use an AlexNet [46] model as the study case for analysis.

#### 6.1.2.1 Computational Cost

In the *Online* phase, the IoT device offloads compute-intense convolutional layers and fully-connected layers to the edge devices. Given a general convolutional layer, the IoT device only needs to perform  $D$  matrix addition with  $Dn^2$  FLOPs for encryption and  $H(\frac{n-k+2p}{s} + 1)^2$  FLOPs for decryption respectively. Compared with executing the same convolutional layer fully on the IoT device, which takes  $2DHk^2(\frac{n-k+2p}{s} + 1)^2$  FLOPs, my scheme significantly reduces real-time computation on the IoT device. It is worth to note that the stride  $s$  in a convolutional layer is typically a small value (e.g., 1 or 2). For a general fully-connected layer, the IoT device needs to perform  $m$  FLOPs for encryption and  $T$  FLOPs for decryption as shown in Eq.5.4 and Eq.5.6 respectively.

TABLE 6.1: Numerical Analysis Summary

Our Scheme						
	Input Size	IoT Computation (FLOPs)		Offloaded Cost to the Edge (FLOPs)	Communication Cost (Elements)	Storage Overhead (Elements)
		Input Encryption	Results Decryption			
<b>Convolutional</b>	$n \times n \times D$	$Dn^2$	$H(\frac{n-k+2p}{s} + 1)^2$	$2DHk^2 \times (\frac{n-k+2p}{s} + 1)^2$	$Dn^2 + H(\frac{n-k+2p}{s} + 1)^2$	$Dn^2 + H(\frac{n-k+2p}{s} + 1)^2$
<b>Fully-connected</b>	$m$	$m$	$T$	$2mT$	$m + T$	$m + T$
Offloading using Plaintext without Privacy Protection						
<b>Convolutional</b>	$n \times n \times D$	N/A	N/A	$2DHk^2 \times (\frac{n-k+2p}{s} + 1)^2$	$Dn^2 + H(\frac{n-k+2p}{s} + 1)^2$	N/A
<b>Fully-connected</b>	$m$	N/A	N/A	$2mT$	$m + T$	N/A

In this table:  $s$  is the stride,  $p$  is the size of padding,  $H$  is the number of kernels,  $k \times k$  is the size of kernels of a convolutional layer;  $T$  is the number of neurons of a fully-connected layer. Each element is 20 Bytes.

Differently, if the IoT device executes such a fully-connected layer at local,  $2mT$  FLOPs are needed.

Besides the offloading of convolutional layers and fully-connected layers, the IoT device also needs to process non-linear layers locally. These non-linear layers are highly compute-efficient. Taking the widely adopted activation layer - ReLU as an example, it only requires  $\frac{1}{2Dk^2}$  of its previous convolutional layer's cost or  $\frac{1}{2m}$  of its previous fully-connected layer's cost.

I now discuss the computational cost of my scheme using AlexNet and FaceNet. As shown in Table 6.2 and Table 6.3, my scheme can offload over 99.9% computational cost for convolutional layers and fully-connected layers in both models, and only leaves lightweight encryption and decryption on the IoT device. Compared with the offloaded convolutional layers and fully-connected layers, the local execution of all non-linear layers only requires 0.08% operations for AlexNet and 0.50% operations for FaceNet. This result further affirms my motivation to offload convolutional layers and fully-connected layers. Meanwhile, as FaceNet requires more computation (additional 0.9 billion FLOPs) in compute-intense layers compared with AlexNet, I observe that the offloaded computation ratio maintains over 99%. This observation supports that my scheme is able to be scaled up and reduce the local computation cost of IoT devices when dealing with more complex deep learning models.

With regards to the encrypted execution on the edge device, my scheme achieves the same computational cost as that directly using unencrypted data as shown in Table 6.1. This is because my encryption (Eq.5.1 and Eq.5.4) in *PPCL* and *PPFL* schemes make the ciphertexts  $Enc(\mathcal{I}_d)$  and  $Enc(\mathcal{V})$  remain the same dimension as their plaintexts  $\mathcal{I}_d$

TABLE 6.2: Example Numerical Analysis on AlexNet

	Parameters	Input Size	IoT Computation (FLOPs)	Offloaded Cost (FLOPs)	Offloaded Percentage	Communication Cost	Storage Overhead
<b>Conv-1</b>	$n=227, H=96$ $k=11, s=4$	$227 \times 227 \times 3$	444,987	210,830,400	99.79%	8691.15 KB	8691.15 KB
<b>Conv-2</b>	$n=27, H=256$ $k=5, s=1$	$27 \times 27 \times 96$	256,608	895,795,200	99.97%	5011.88 KB	5011.88 KB
<b>Conv-3</b>	$n=13, H=384$ $k=3, s=1$	$13 \times 13 \times 256$	108,160	299,040,768	99.96%	2112.50 KB	2112.50 KB
<b>Conv-4</b>	$n=13, H=384$ $k=3, s=1$	$13 \times 13 \times 384$	129,792	448,561,152	99.97%	2535.00 KB	2535.00 KB
<b>Conv-5</b>	$n=13, H=256$ $k=3, s=1$	$13 \times 13 \times 384$	108,160	299,040,768	99.96%	2112.50 KB	2112.50 KB
<b>FC-1</b>	$m=9216, T=4096$	9216	13,312	75,497,472	99.98%	260.00 KB	260.00 KB
<b>FC-2</b>	$m=4096, T=4096$	4096	8,192	33,554,432	99.98%	160.00 KB	160.00 KB
<b>FC-3</b>	$m=4096, T=1000$	4096	5,096	8,192,000	99.94%	99.53 KB	99.53 KB
<b>Total Cost</b>	N/A	N/A	1,074,307	2,270,512,192	99.95%	20.49 MB	20.49 MB
<b>Computation for All Activation and Pooling Layers on the IoT: 650,080 FLOPs and 1,102,176 FLOPs</b>							



TABLE 6.3: Example Numerical Analysis on FaceNet

	Parameters	Input Size	IoT Computation (FLOPs)	Offloaded Cost (FLOPs)	Offloaded Percentage	Communication Cost	Storage Overhead
<b>Conv-1</b>	n=220, H=64 k=7, s=2	220 × 220 ×3	1,306,800	341,510,400	99.62%	25523.44 KB	25523.44 KB
<b>Conv-2a</b>	n=55, H=64 k=1, s=1	55 × 55 ×64	387,200	24,780,800	98.44%	7562.50 KB	7562.50 KB
<b>Conv-2</b>	n=55, H=192 k=3, s=1	55 × 55 ×64	774,400	669,081,600	99.88%	15125.00 KB	15125.00 KB
<b>Conv-3a</b>	n=28, H=192 k=1, s=1	28 × 28 ×192	301,056	57,802,752	99.48%	5880.00 KB	5880.00 KB
<b>Conv-3</b>	n=28, H=384 k=3, s=1	28 × 28 ×192	451,584	1,040,449,536	99.96%	8820.00 KB	8820.00 KB
<b>Conv-4a</b>	n=14, H=384 k=1, s=1	14 × 14 ×384	150,528	57,802,752	99.74%	2940.00 KB	2940.00 KB
<b>Conv-4</b>	n=14, H=256 k=3, s=1	14 × 14 ×384	125,440	346,816,512	99.96%	2450.00 KB	2450.00 KB
<b>Conv-5a</b>	n=14, H=256 k=1, s=1	14 × 14 ×256	100,352	25,690,112	99.61%	1960.00 KB	1960.00 KB
<b>Conv-5</b>	n=14, H=256 k=3, s=1	14 × 14 ×256	100,352	231,211,008	99.96%	1960.00 KB	1960.00 KB
<b>Conv-6a</b>	n=14, H=256 k=1, s=1	14 × 14 ×256	100,352	25,690,112	99.61%	1960.00 KB	1960.00 KB
<b>Conv-6</b>	n=14, H=256 k=3, s=1	14 × 14 ×256	100,352	231,211,008	99.96%	1960.00 KB	1960.00 KB
<b>FC-1</b>	m=12544, T=4096	12544	16,640	102,760,448	99.98%	325.00 KB	325.00 KB
<b>FC-2</b>	m=4096, T=4096	4096	8,192	33,554,432	99.98%	160.00 KB	160.00 KB
<b>FC-7128</b>	m=4096, T=128	4096	4,224	1,048,576	99.60%	82.50 KB	82.50 KB
<b>Total Cost</b>	N/A	N/A	3,927,472	3,189,410,048	99.88%	74.91 MB	74.91 MB
<b>Computation for All Activation and Pooling Layers on the IoT: 2,334,848 FLOPs and 13,651,456 FLOPs</b>							

and  $\mathcal{V}$ . Such a decent property guarantees real-time computational performance on the edge device.

In the *Offline* phase, the IoT device owner first prepares encryption keys by choosing random matrices for convolutional layers and fully-connected layers that will be offloaded. Meanwhile, the owner will take these encryption keys as inputs for their corresponding convolutional layers or fully-connected layers to obtain results as the decryption keys. In Section 6.2, I show that the offline phase can be efficiently executed using a regular laptop.

### 6.1.2.2 Communication Cost

The communication cost of my scheme comes from the transmission of encrypted inputs and outputs of convolutional layers and fully-connected layers. In my implementation, I use 160-bit random numbers (i.e.,  $\lambda = 160$ ) during all encryption processes in Eq.5.1 and Eq.5.4. Thus, each element in the ciphertext (a matrix or a vector) is 20-Byte long. To offload a convolutional layer with a  $n \times n \times D$  input, the IoT device first sends its corresponding ciphertext contains  $D$  encrypted matrices with  $Dn^2$  elements in total. Then,  $H$  encrypted result matrices are received from the edge device with each size of  $(\frac{n-k+2p}{s} + 1) \times (\frac{n-k+2p}{s} + 1)$ . With regards to the offloading of a fully-connected layer that takes a  $m$ -dimensional vector as input, the IoT device needs to send a  $m$ -dimensional vector as encrypted input and receive a  $T$ -dimensional vector as encrypted output from the edge device. As shown in Table 6.2, the communication cost for an offloading of the AlexNet is 20.49MB, which can be efficiently handled under the edge computing environment [107].

### 6.1.2.3 Storage Overhead

For the offloading of a convolutional layer with a  $n \times n \times D$  input, the IoT device needs to store  $D$  random matrices with  $n^2$  elements each as the encryption keys, and  $H$  matrices with size of  $(\frac{n-k+2p}{s} + 1) \times (\frac{n-k+2p}{s} + 1)$  as the decryption keys. To offload a fully-connected layer with a  $m$ -dimensional vector as input, a  $m$ -dimensional vector and a  $T$ -dimensional vector need to be pre-stored as the encryption key and decryption key respectively. Table 6.2 shows the offloading of an AlexNet request needs 20.49MB storage overhead. With the rise of IoT devices, low-power-consumption SD memory card has become an excellent fit to economically extend the storage of IoT devices [108], which usually have more than 32GB capacity. As discussed in Section 5.6, IoT devices deployed for one-time tasks can easily pre-load enough keys with a SD card. Meanwhile, remote key update can be adopted long-term deployment.

In this Table 6.4,  $s$  is the stride,  $p$  is the size of padding,  $H$  is the number of kernels,  $k \times k$  is the size of kernels of a convolutional layer;  $\theta$  is the error rate of the returned data;  $r$  is the sample rate of the returned data. Each element is 20 Bytes.

### 6.1.2.4 Resource Consumption of Integrity Check

Turning on the integrity check leads to additional resource consumption to local IoT device. As shown in Table 6.4, given a returned matrix of size  $H(\frac{n-k+2p}{s} + 1)^2$  and a sample rate of  $r$ , the validation process in Section 5.8 brings  $\lceil rH(\frac{n-k+2p}{s} + 1)^2 \rceil$  additional computation and makes the total computational cost of IoT devices rise to  $2Dk^2 \lceil rH(\frac{n-k+2p}{s} + 1)^2 \rceil$ . Since any convolutional result in the entire set of response map can be incorrect, IoT devices need to store all kernel parameters of each convolutional

TABLE 6.4: Numerical Analysis of Integrity Check

	Computation of the IoT (FLOPs)			Offloaded Cost to the Edge (FLOPs)	Communication Cost (Elements)	Storage Overhead (Elements)
	Input Encryption	Results Decryption	Results Validation			
<b>Integrity Check</b>	$Dn^2$	$H(\frac{n-k+2p}{s} + 1)^2$	$\lceil rH(\frac{n-k+2p}{s} + 1)^2 \rceil$ $2Dk^2 \times (\frac{n-k+2p}{s} + 1)^2$	$\frac{2DHk^2 \times (\frac{n-k+2p}{s} + 1)^2}{(\frac{n-k+2p}{s} + 1)^2}$	$Dn^2 + H(\frac{n-k+2p}{s} + 1)^2$	$Dn^2 + Hk^2 + H(\frac{n-k+2p}{s} + 1)^2$
<b>No Integrity Check</b>	$Dn^2$	$H(\frac{n-k+2p}{s} + 1)^2$	0	$\frac{2DHk^2 \times (\frac{n-k+2p}{s} + 1)^2}{(\frac{n-k+2p}{s} + 1)^2}$	$Dn^2 + H(\frac{n-k+2p}{s} + 1)^2$	$Dn^2 + H(\frac{n-k+2p}{s} + 1)^2$

TABLE 6.5: Example Comparison with/without Integrity Check

	Computation of IoT (FLOPs)		Offloaded Percentage	
	No Integrity Check	Integrity Check	No Integrity Check	Integrity Check
<b>Conv-1</b>	444,987	866,793	99.79%	99.59%
<b>Conv-2</b>	256,608	2,944,608	99.97%	99.67%
<b>Conv-3</b>	108,160	2,504,320	99.96%	99.16%
<b>Conv-4</b>	129,792	3,724,032	99.97%	99.17%
<b>Conv-5</b>	108,160	3,398,272	99.96%	98.86%

	Communication Cost (KB)		Storage Overhead (KB)	
	No Integrity Check	Integrity Check	No Integrity Check	Integrity Check
<b>Conv-1</b>	8691.15	8691.27	8691.15	8918.03
<b>Conv-2</b>	5011.88	5011.98	5011.88	5136.88
<b>Conv-3</b>	2112.50	2112.60	2112.50	2180.00
<b>Conv-4</b>	2535.00	2535.10	2535.00	2602.50
<b>Conv-5</b>	2112.50	2112.59	2112.50	2157.50

The results in this table is generated based on error rate  $\theta = 1\%$  and sample rate  $r = 0.2\%, 0.3\%, 0.8\%, 0.8\%, 1.1\%$  for Conv-1 - Conv-5 respectively.

layer locally, which adds on  $Hk^2$  storage overhead and makes the total IoT storage overhead to be  $Dn^2 + H(\frac{n-k+2p}{s} + 1)^2 + Hk^2$ .

Table 6.5 shows the resource consumption comparison between integrity check plugged in and plugged off. The results are calculated when error rate  $\theta = 1\%$  and sample rate  $r = 0.2\%, 0.3\%, 0.8\%, 0.8\%, 1.1\%$  for Conv-1 - Conv-5 respectively. Under this setting, IoT device can achieve 99%+ error detection rate in each convolutional layer. Since all the multiplication results of  $r\theta$  are less equal to  $1.1 \times 10^{-4}$ , the additional communication costs resulted from integrity check are tiny. As a result, the communication increments are less than  $4.74 \times 10^{-3}\%$  of the original communication costs. Compared with the heavy parameters in fully-connected layers, the parameters in convolutional layers only stand for a minor portion of the entire neural network model. Thus, even the highest additional storage overhead is only 227 KB while the lowest increment can be as low as 45 KB.

### 6.1.2.5 Analysis of Fast Convolution

In this section, I analyze the performance improvement introduced by the fast filtering algorithm. I scrutinize the multiplication number of each convolutional layer in both AlexNet and FaceNet to demonstrate that fast filtering algorithm could enhance convolution efficiency under different models.

From Table 6.6 I observe that the second convolutional layer in AlexNet achieves a highest multiplication speedup rate of  $26.89\times$ . That's because its filters are with a small size of  $5 \times 5$ , which is a perfect use case for fast filtering algorithm as indicated by [96]. Moreover, the outputs of that layer shrink a lot in size compared with its inputs. The first convolutional layer does not achieve a similar high speedup rate due to its bigger filters and so do the last two convolutional layers since they lack notable shrink in the output size.

Regarding FaceNet, I perform analysis for both categories where the first category includes all 11 convolutional layers while the second one does not apply layers whose filter sizes are  $1 \times 1$  (in other words, a-suffix layers). Table 6.7 indicates that convolution in these a-suffix layers with  $1 \times 1$  filters is indeed element-wise multiplication. Considering the nature of fast filtering algorithm stated in Section 5.9, speeding up such element-wise multiplication is unrealistic under my settings. Thus it is understandable that the speedup rate for the a-suffix layer is  $1\times$ , which also explains the reason that first category's speedup rate is about  $1.5\times$  lower than the second category's. The cases of AlexNet and FaceNet both demonstrate that the fast filtering algorithm can boost the performance of my design in terms of convolution efficiency.

TABLE 6.6: Efficiency Enhancement Analysis on AlexNet

	Parameters	Input Size	Original Convolution Multiplication Number	Fast Convolution Multiplication Number	Speedup Rate
<b>Conv-1</b>	$n=227, H=96$ $k=11, s=4$	$227 \times 227 \times 3$	123,370,632	14,840,352	$8.31 \times$
<b>Conv-2</b>	$n=27, H=256$ $k=5, s=1$	$27 \times 27 \times 96$	481,689,600	17,915,904	$26.89 \times$
<b>Conv-3</b>	$n=13, H=384$ $k=3, s=1$	$13 \times 13 \times 256$	353,894,400	16,613,376	$21.30 \times$
<b>Conv-4</b>	$n=13, H=384$ $k=3, s=1$	$13 \times 13 \times 384$	260,112,384	24,920,064	$10.44 \times$
<b>Conv-5</b>	$n=13, H=256$ $k=3, s=1$	$13 \times 13 \times 384$	173,408,256	16,613,376	$10.44 \times$
<b>Total</b>	N/A	N/A	1,392,475,272	90,903,072	$15.32 \times$

TABLE 6.7: Efficiency Enhancement Analysis on FaceNet

	Parameters	Input Size	Original Convolution Multiplication Number	Fast Convolution Multiplication Number	Speedup Rate
<b>Conv-1</b>	$n=220, H=64$ $k=7, s=2$	$220 \times 220 \times 3$	170,755,200	13,939,200	$12.25 \times$
<b>Conv-2a</b>	$n=55, H=64$ $k=1, s=1$	$55 \times 55 \times 64$	12,390,400	12,390,400	$1.00 \times$
<b>Conv-2</b>	$n=55, H=192$ $k=3, s=1$	$55 \times 55 \times 64$	334,540,800	37,171,200	$9.00 \times$
<b>Conv-3a</b>	$n=28, H=192$ $k=1, s=1$	$28 \times 28 \times 192$	28,901,376	28,901,376	$1.00 \times$
<b>Conv-3</b>	$n=28, H=384$ $k=3, s=1$	$28 \times 28 \times 192$	520,224,768	57,802,752	$9.00 \times$
<b>Conv-4a</b>	$n=14, H=384$ $k=1, s=1$	$14 \times 14 \times 384$	28,901,376	28,901,376	$1.00 \times$
<b>Conv-4</b>	$n=14, H=256$ $k=3, s=1$	$14 \times 14 \times 384$	173,408,256	19,267,584	$9.00 \times$
<b>Conv-5a</b>	$n=14, H=256$ $k=1, s=1$	$14 \times 14 \times 256$	12,845,056	12,845,056	$1.00 \times$
<b>Conv-5</b>	$n=14, H=256$ $k=3, s=1$	$14 \times 14 \times 256$	115,605,504	12,845,056	$9.00 \times$
<b>Conv-6a</b>	$n=14, H=256$ $k=1, s=1$	$14 \times 14 \times 256$	12,845,056	12,845,056	$1.00 \times$
<b>Conv-6</b>	$n=14, H=256$ $k=3, s=1$	$14 \times 14 \times 256$	115,605,504	12,845,056	$9.00 \times$
<b>Category 1 Total</b>	N/A	N/A	1,526,023,296	249,754,112	$6.11 \times$
<b>Category 2 Total</b>	N/A	N/A	1,430,140,032	153,870,848	$9.29 \times$



## 6.2 Evaluation

I implemented a prototype of my scheme using Python 2.7. In my implementation, TensorFlow and Keras libraries are adopted to support CNNs. The resource-constrained IoT device is a Raspberry Pi (Model A) with Raspbian Debian 7, which has 700 MHz single-core processor, 256MB memory, and 32GB SD card storage. The edge device and the IoT device owner is a Macbook Pro laptop with OS X 10.13.3, 3.1 GHz Intel Core i7 processor, 16GB memory, and 512GB SSD. The IoT device and the edge device are connected using WiFi in the same subnet. I use the well-known ImageNet [109] as the dataset for the evaluation of AlexNet. The security parameter  $\lambda$  is set as 160 in my implementation. I also implemented a privacy-preserving AlexNet using CryptoNets [36] for comparison.

### 6.2.1 Efficiency - Offline Phase

To generate the encryption and decryption keys for the execution of one AlexNet request, my scheme only requires 114ms for the IoT device owner. While each set of keys will only be used for one request, the owner can efficiently compute more than 2600 sets of keys for AlexNet using 5 minutes.

### 6.2.2 Efficiency - Online Phase

I summarize the evaluation results of a real-time AlexNet inference task in Table 6.8. Compared with executing the entire inference task on the IoT device, my scheme significantly reduces execution time from 124.99s to 3.508s, which indicates a  $35.63\times$  speedup rate. More importantly, with the increasing complexity of convolutional layers and fully-connected layers, my scheme retains or increases the high speedup rate as shown in the

TABLE 6.8: Experimental Evaluation Results on AlexNet

	IoT without Offloading (second)	Our Scheme				Speedup
		IoT Computation (second)	Edge Computation (second)	Communication (second)	Total (second)	
<b>Conv-1</b>	10.01	0.037	0.0103	0.849	0.896	11.17×
<b>Conv-2</b>	40.68	0.0405	0.0435	0.489	0.573	70.99×
<b>Conv-3</b>	19.93	0.0437	0.013	0.206	0.263	75.78×
<b>Conv-4</b>	29.78	0.0498	0.0184	0.248	0.316	94.24×
<b>Conv-5</b>	19.88	0.0420	0.0127	0.206	0.261	76.17×
<b>FC-1</b>	2.22	0.0013	0.0043	0.025	0.031	71.61×
<b>FC-2</b>	1.08	0.001	0.0025	0.016	0.019	56.84×
<b>FC-3</b>	0.27	0.0008	0.0009	0.01	0.012	22.5×
<b>Non-linear</b>	1.137	1.137	N/A	N/A	1.137	N/A
<b>Total Cost</b>	124.99	1.353	0.106	2.049	3.508	35.63×

TABLE 6.9: Comparison Between My Scheme and CryptoNets in First Convolutional Layer of AlexNet

	<b>Our Scheme (seconds)</b>	<b>A-CryptoNets (seconds)</b>
<b>Encryption</b>	0.012	459.93
<b>Convolution</b>	0.0103	625.86
<b>Decryption</b>	0.025	N/A

last column of Table 6.8. Taking AlexNet as an example, the highest speedup rates for them are all achieved with these more complex layers. Therefore, my scheme is promising to be scaled up to support more complex CNN architectures according to practical requirements.

It is noteworthy that the communication occupies 58.4% (2.049s/3.508s) cost of an offloaded inference task in my scheme. In my implementation, I use a wireless network with 10MB/s transmission speed between the IoT device and the edge device. In a real-world scenario, the devices are likely to be connected via wired or cellular connection, which allows a higher transmission speed than my experimental environment. Moreover, the upcoming 5G era for MEC environment will significantly improve the transmission speed [107] and further optimize the communication performance of my scheme.

To compare my scheme with homomorphic encryption-based solution [36] for CNN inference, I also implemented AlexNet using the CryptoNets scheme proposed in [36], denoted as A-CryptoNets. During my implementation, I use the same linear approximation and YASHE cryptosystem [110] as that in [36]. Table 6.9 shows the cost of processing the first convolutional layer in my scheme and A-CryptoNets. Due to the large input size required in AlexNet, A-CryptoNets requires 459.93s for encrypting the input data on the IoT device, and 625.86s for the convolutional operations on the edge device. This level of computational cost makes CryptoNets become hardly to satisfy time-sensitive tasks with complex CNN inference. As a comparison, my scheme can handle the first convolutional layer using 0.047s. Even considering the entire AlexNet inference task, my scheme only requires 3.508s to complete.

TABLE 6.10: Experimental Evaluation Results on Integrity Check

	IoT without Offloading (second)	Our Scheme			Speedup
		Integrity Check (second)	Speedup	No Integrity Check (second)	
<b>Conv-1</b>	10.01	0.916	10.93×	0.896	11.17×
<b>Conv-2</b>	40.68	0.655	62.11×	0.573	70.99×
<b>Conv-3</b>	19.93	0.402	49.58×	0.263	75.78×
<b>Conv-4</b>	29.78	0.524	56.83×	0.316	94.24×
<b>Conv-5</b>	19.88	0.470	42.30×	0.261	76.17×
<b>FC-1</b>	2.22	0.031	71.61×	0.031	71.61×
<b>FC-2</b>	1.08	0.019	56.84×	0.019	56.84×
<b>FC-3</b>	0.27	0.012	22.5 ×	0.012	22.5×
<b>Non-linear</b>	1.137	1.137	N/A	1.137	N/A
<b>Total Cost</b>	124.99	4.166	30.00×	3.508	35.63×

I also provide evaluation on my proposed integrity check module. In Table 6.10 I observe that even when the integrity check is plugged in, my scheme can still achieve a high speedup rate of 30.00× compared with AlexNet local execution. This demonstrates the practical usage of the utility of integrity check model. Users can opt whether to plug in the module and achieve the ability of detecting dishonest behavior of edge devices with a little efficiency trade-off.

TABLE 6.11: Power and Energy Consumption Evaluation

	IoT Local Executing AlexNet without Network Connection	Idle IoT with Network Connection	Our Scheme	
			IoT Computation	IoT Communication
Power (W)	0.81	0.78	1.17	1.42
Energy (J)	101.24	N/A	1.58	2.91

### 6.2.3 Energy Consumption

Compared with fully executing AlexNet inference tasks on the IoT device with high energy consumption, my scheme significantly saves the energy consumption for computation of the IoT device while introducing slight extra energy consumption for communication. In my evaluation, the IoT device (Raspberry Pi Model A) is powered by a 5V micro-USB adapter. The voltage and current is measured using a Powerjive USB multimeter [111] and the power is calculated by the multiplication of voltage and current. Table 6.11 shows the average IoT power consumption under different IoT device status. I observe that the network connection is a major power cost in IoT device. An idle IoT device with network connection can have a comparable power cost (0.78W) as executing AlexNet locally without network connection (0.81W). This local AlexNet execution power indicates an energy consumption of 101.24J when fully executing an inference task on the IoT device in 124.99 seconds. Differently, my scheme reduces the computation time on the IoT device to 1.353 seconds (1.59J energy consumption) with 2.049 seconds extra communication (2.90J energy consumption). Therefore, my scheme can save IoT energy consumption by  $\frac{101.24-(1.58+2.91)}{101.24} = 95.56\%$ .

### 6.2.4 Accuracy

To validate that there is no accuracy loss in my scheme, I also implemented original AlexNet without privacy protection as baseline. By using the same parameters, I achieve

exact the same accuracy (80.1%) as that obtained using original AlexNet [46] without privacy protection. On one hand, no approximation for non-linear layers is required in my scheme. On the other hand, the random noise introduced in my encryption can be perfectly eliminated during the decryption process.

### 6.2.5 Evaluation of Sample Rate in Integrity Check

In order to achieve a high error detection rate, different sample rate  $r$  needs to be calculated based on different settings in each convolutional layer. As shown in Figure 6.1, to make the error detection rate to surpass 99%, Conv-1 - Conv-5 need to set  $r$  to be 0.2%, 0.3%, 0.8%, 0.8%, 1.1% respectively. Figure 6.2 shows that as the size of the returned data rises, the sample rate  $r$  required to reach 99%+ error detection rate drops correspondingly. From this observation combined with Figure 6.3, the scalability of the integrity check feature is ensured and the additional resource consumption of a larger, more complex CNN is always minor compared with its original costs.

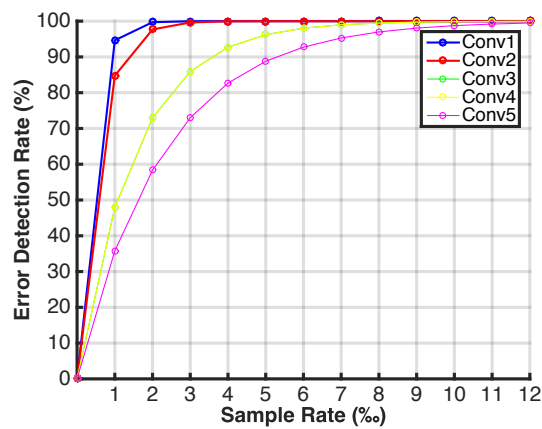


FIGURE 6.1: Evaluation of Sample Rate  $r$  and Error Detection Rate

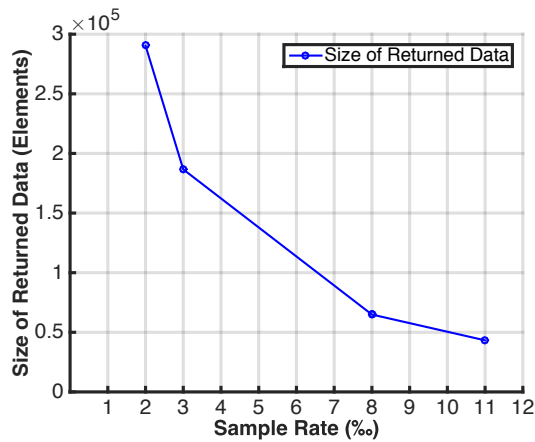


FIGURE 6.2: Evaluation of Sample Rate  $r$  and Returned Data Size

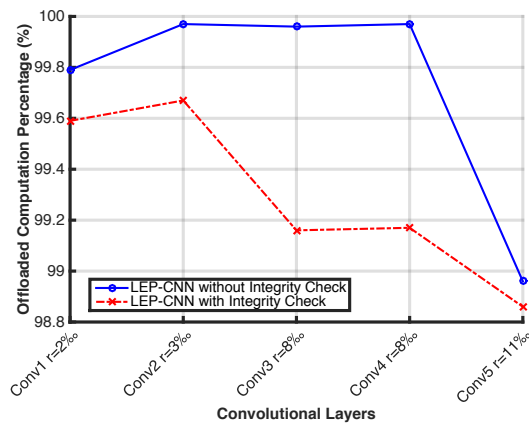


FIGURE 6.3: Evaluation of Convolutional Layers and Offloaded Computation Percentage

### 6.2.5.1 Evaluation of Matrix Compression

In order to evaluate the performance of the matrix compression module, I perform experiments on both AlexNet and FaceNet to illustrate the integration of ZFP compression can alleviate the communication cost in different models. From Table 6.12 and Table 6.13, it is clear that the communication cost of my scheme is reduced by 72%+ in both models.

TABLE 6.12: Communication Enhancement on AlexNet

	Parameters	Input Size	Original Communication Overhead	Compressed Communication Overhead	Reduced Ratio
<b>Conv-1</b>	$n=227, H=96$ $k=11, s=4$	$227 \times 227 \times 3$	8691.15 KB	2411.85 KB	72.25%
<b>Conv-2</b>	$n=27, H=256$ $k=5, s=1$	$27 \times 27 \times 96$	5011.88 KB	1378.27 KB	72.50%
<b>Conv-3</b>	$n=13, H=384$ $k=3, s=1$	$13 \times 13 \times 256$	2112.50 KB	580.02 KB	72.54%
<b>Conv-4</b>	$n=13, H=384$ $k=3, s=1$	$13 \times 13 \times 384$	2535.00 KB	698.47 KB	72.45%
<b>Conv-5</b>	$n=13, H=256$ $k=3, s=1$	$13 \times 13 \times 384$	2112.50 KB	582.06 KB	72.45%
<b>FC-1</b>	$m=9216, T=4096$	9216	260.00 KB	70.27 KB	72.97%
<b>FC-2</b>	$m=4096, T=4096$	4096	160.00 KB	43.64 KB	72.73%
<b>FC-3</b>	$m=4096, T=1000$	4096	99.53 KB	27.14 KB	72.73%
<b>Total</b>	N/A	N/A	20.49 MB	5.66 MB	72.38%

### 6.3 Conclusion

In this chapter, I give out detailed security analysis to show that my scheme is secure with formal proof. By performing extensive numerical analysis as well as prototype implementation over the well-known CNN architectures and datasets, I present the practical performance of my privacy-preserving compute-intense CNN layers along with a



TABLE 6.13: Communication Enhancement on FaceNet

	Parameters	Input Size	Original Communication Overhead	Compressed Communication Overhead	Reduced Ratio
<b>Conv-1</b>	n=220, H=64, k=7, s=2	220 × 220 × 3	25523.44 KB	6646.73 KB	73.96%
<b>Conv-2a</b>	n=55, H=64, k=1, s=1	55 × 55 × 64	7562.50 KB	2102.34 KB	72.20%
<b>Conv-2</b>	n=55, H=192, k=3, s=1	55 × 55 × 64	15125.00 KB	4204.67 KB	72.20%
<b>Conv-3a</b>	n=28, H=192, k=1, s=1	28 × 28 × 192	5880.00 KB	1588.67 KB	72.98%
<b>Conv-3</b>	n=28, H=384, k=3, s=1	28 × 28 × 192	8820.00 KB	2383.00 KB	72.98%
<b>Conv-4a</b>	n=14, H=384, k=1, s=1	14 × 14 × 384	2940.00 KB	810.70 KB	72.43%
<b>Conv-4</b>	n=14, H=256, k=3, s=1	14 × 14 × 384	2450.00 KB	675.58 KB	72.43%
<b>Conv-5a</b>	n=14, H=256, k=1, s=1	14 × 14 × 256	1960.00 KB	542.54 KB	72.31%
<b>Conv-5</b>	n=14, H=256, k=3, s=1	14 × 14 × 256	1960.00 KB	542.54 KB	72.31%
<b>Conv-6a</b>	n=14, H=256, k=1, s=1	14 × 14 × 256	1960.00 KB	542.54 KB	72.31%
<b>Conv-6</b>	n=14, H=256, k=3, s=1	14 × 14 × 256	1960.00 KB	542.54 KB	72.31%
<b>FC-1</b>	m=12544, T=4096	12544	325.00 KB	91.00 KB	72.00%
<b>FC-2</b>	m=4096, T=4096	4096	160.00 KB	43.64 KB	72.73%
<b>FC-7128</b>	m=4096, T=128	4096	82.50 KB	22.50 KB	72.73%
<b>Category 1 Total</b>	N/A	N/A	74.91 MB	20.25 MB	72.97%
<b>Category 2 Total</b>	N/A	N/A	55.08 MB	14.80 MB	73.13%

set of pluggable modules. Experiment results prove that my designed modules are able to efficiently, accurately protect the privacy of deep learning based image analysis by greatly tranfering the heavy computation burden from IoT devices to edge devices with no accuracy loss in a privacy-preserving manner.

## Chapter 7

# Future Works and Conclusion

In this chapter, I discuss several future research tasks of this dissertation and then conclude this dissertation.

### 7.1 Extension of Descriptor Based Image Analysis

As presented in Chapter 3, I design the privacy-preserving randomized k-d forest index to improve the search efficiency my initial design CAPIA. However, the k-d tree as well its derivatives have large construction cost in the setup stage, which makes them not suitable for frequently updating datasets. To overcome this limitation, one possible direction is to investigate data structures more fit to frequent dataset insertion (e.g., graphs, R-Tree [112], etc.). Then the challenge to be solved is how to incorporate the potential data structure into image annotation task in a privacy-preserving manner. Furthermore, in my privacy-preserving distance comparison design, the major type of operations is matrix multiplication. I plan to convert matrix multiplications into simple element multiplications, and then make them compatible with MapReduce [113], which

is an extremely efficient model for the processing of a large number of simple operations over big datasets. In addition, it's also worth to migrate my design to other popular imagery data analysis tasks such as image classification, object detection, information retrieval etc.

## 7.2 Extension of Deep Learning Based Image Analysis

In Chapter 5, I present privacy-preserving convolution and fully connected layer along with the novel online/offline strategy to enable efficient, accurate and private image analysis using CNN. In my design, I offload the convolutional layer and fully connected layer due to that their representation of heavy computation and storage overhead. A possible next move can be targeted at securely offloading all the non-linear layer in the CNN model. This can bring huge boost to the efficiency of CNN inference due to that once both the linear and non-linear layers are offloaded, the communication between IoT and edge device will be reduced to just two rounds. Instead of uploading encrypted results after local execution of non-linear layer, the IoT device only needs to submit the encrypted image matrix before the first layer and then just waits for the final inference results. In addition, as the flourishing of different deep learning models, I expect to investigate other deep structures such as Long Short-Term Memory (LSTM) and other recurrent neural networks, Regional Convolution Neural Network (RCNN) and its follow-up regional CNNs, Generative Adversarial Network (GAN) and other adversarial networks etc. Different privacy-preserving schemes should be designed to adapt unique characteristics for these models. Moreover, recently, federated learning [114] got increasing attraction in the field since it provides an efficient methodology to train deep learning models among multiple registered ad-hoc imagery data contributors. Unlike traditional

collaborative learning frameworks which need to upload imagery data to perform the multi-party training, federated learning enables the participants to upload the weight update after each local training epoch. This greatly reduces the communication cost and alleviate the potential risk for information leakage from uploading the entire image. However, even the weight update can still be compromise the privacy of user image if attacked by a well trained GAN [115]. How to prevent image reconstruction or membership attack launched by GAN is an open challenge to traditional defense strategy in traditional threat model definition.

### **7.3 A Privacy-preserving Hybrid Cloud-Edge Framework for Image Analysis**

Another promising research direction is regarding performing image analysis tasks on mobile edge computing (MEC), and utilize MEC to overcome the limitation of high network latency in cloud-assisted architectures. I first plan to utilize MEC to resolve the “data drowning” issue in cloud-assisted architectures in the sense that not all the imagery data collected by devices worth an in-depth analysis. For example, in video data captured by a surveillance camera, only frames that contain suspect objects need additional analytics on them. Therefore, I will utilize MEC to detect imagery data that has a high potential to contain critical information, and only offload these critical data to cloud for further analysis. Considering the third-party deployment nature of MEC, I will design and develop privacy-preserving solutions for popular imagery data filtering techniques, such as fast object detection, content-based sampling of video sequences, etc. In addition, while cloud computing is a prevalent choice for the offloading of image analysis, I argue that many image analysis tasks can be completed directly using resources

on MEC. For example, many CNN architectures can be handled efficiently by moderate GPUs, which can be provided by MEC infrastructures. In my design, I will investigate how to utilize computation resources of MEC to provide light-weight privacy-preserving image analysis services. This design will be extremely suitable for time-sensitive imagery analysis tasks (e.g., hazard detection, intelligence, and reconnaissance), since MEC is typically within one-hop communication range with devices and is able to provide the fastest response to them. Finally, I will integrate my designs for MEC and public cloud as a coherent framework, which will provide privacy-preserving image analysis services in a hybrid manner to fulfill the performance requirement of different application scenarios.

## 7.4 Conclusion

In this dissertation, I address the challenge of how to leverage the power of cloud/edge server to perform efficient and accurate image analysis in a privacy-preserving manner. I develop a generic methodology for imagery data analysis under different scenarios and take *efficiency*, *accuracy*, as well as *security* into consideration at the same time. I also evaluate other rubrics such as storage cost and energy consumption for some specific deployment environment (e.g. resource-constrained IoT devices). To be specific, I design a few novel modules to be plugged in to meet different requirements in various situations. *PL1C*, *PKLC* along with their enhancement module *PL1C – RF* and *PKLC – RF* is designed to protect the privacy of image per se and corresponding feature vectors while enabling accurate and efficient image annotation. *PPCL* and *PPKL* is proposed to efficiently support privacy protection of user image in CNN inference stage with no accuracy loss. Pluggable integrity check, fast convolution and matrix compression module are also introduced to further bring fancier features to the scheme and enhance its

---

performance in terms of efficiency. With the knowledge that imagery data analysis tasks could be deployed in cloud/edge environments using either descriptor/deep learning based approaches, I integrate these modules in cloud-assisted descriptor based image annotation task and edge-assisted deep learning based image analysis task. Thorough security analysis is provided to show that my modules are secure in their application scenarios. Extensive numerical analysis as well as prototype implementation over the well-known dataset and CNN architectures demonstrate the practical performance of my design.

# Bibliography

- [1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [2] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.
- [3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [4] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
- [5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [6] Ameesh Makadia, Vladimir Pavlovic, and Sanjiv Kumar. A new baseline for image annotation. *Computer Vision–ECCV 2008*, pages 316–329, 2008.



- 
- [7] Changbo Yang, Ming Dong, and Jing Hua. Region-based image annotation using asymmetrical support vector machine-based multiple-instance learning. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2057–2063. IEEE, 2006.
- [8] Gustavo Carneiro, Antoni B Chan, Pedro J Moreno, and Nuno Vasconcelos. Supervised learning of semantic classes for image annotation and retrieval. *IEEE transactions on pattern analysis and machine intelligence*, 29(3):394–410, 2007.
- [9] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.
- [10] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004.
- [11] Robert M Haralick and Linda G Shapiro. Image segmentation techniques. *Computer vision, graphics, and image processing*, 29(1):100–132, 1985.
- [12] Nikhil R Pal and Sankar K Pal. A review on image segmentation techniques. *Pattern recognition*, 26(9):1277–1294, 1993.
- [13] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.
- [14] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

- [15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.
- [16] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. *arXiv preprint arXiv:1703.06870*, 2017.
- [17] Richard Oneslager. How Many Photos Were Taken Last Year? <https://blog.forever.com/forever-blog/2018/1/22/how-many-photos-were-taken-last-year>, 2018.
- [18] Jianchao Yang, Kai Yu, Yihong Gong, and Thomas Huang. Linear spatial pyramid matching using sparse coding for image classification. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1794–1801. IEEE, 2009.
- [19] Peter Gehler and Sebastian Nowozin. On feature combination for multiclass object classification. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 221–228. IEEE, 2009.
- [20] Jiri Matas, Ondrej Chum, Martin Urban, and Tomáš Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and vision computing*, 22(10):761–767, 2004.
- [21] Anuj Mohan, Constantine Papageorgiou, and Tomaso Poggio. Example-based object detection in images by components. *IEEE transactions on pattern analysis and machine intelligence*, 23(4):349–361, 2001.

- 
- [22] Ameesh Makadia, Vladimir Pavlovic, and Sanjiv Kumar. Baselines for image annotation. *Int. J. Comput. Vision*, 90(1):88–105, October 2010. ISSN 0920-5691. doi: 10.1007/s11263-010-0338-6. URL <http://dx.doi.org/10.1007/s11263-010-0338-6>.
- [23] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [24] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.
- [25] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. Deep face recognition. In *BMVC*, volume 1, page 6, 2015.
- [26] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.
- [27] M Shamim Hossain and Ghulam Muhammad. Cloud-assisted industrial internet of things (iiot)-enabled framework for health monitoring. *Computer Networks*, 101:192–202, 2016.
- [28] Rajesh Bahadur Thapa and Yuji Murayama. Urban mapping, accuracy, & image classification: A comparison of multiple approaches in tsukuba city, japan. *Applied geography*, 29(1):135–144, 2009.
- [29] Privacyrights.org. The Privacy Implications of Cloud Computing. <https://www.privacyrights.org/blog/privacy-implications-cloud-computing>, 2017.

- [30] Centers for Medicare & Medicaid Services et al. The health insurance portability and accountability act of 1996 (hipaa). *Online at <http://www.cms.hhs.gov/hipaa>*, 1996.
- [31] Boxcryptor. Encrypt your files in your dropbox. <https://www.boxcryptor.com/en/dropbox>. [Online; accessed Aug. 2016].
- [32] Amazon Simple Storage Service. Protecting data using encryption. <http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingEncryption.html>. [Online; accessed Aug. 2016].
- [33] Chao-Yung Hsu, Chun-Shien Lu, and Soo-Chang Pei. Image feature extraction in encrypted domain with privacy-preserving sift. *IEEE Transactions on Image Processing*, 21(11):4593–4607, 2012.
- [34] Qian Wang, Jingjun Wang, Shengshan Hu, Qin Zou, and Kui Ren. Sechog: Privacy-preserving outsourcing computation of histogram of oriented gradients in the cloud. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 257–268. ACM, 2016.
- [35] Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Inald Lagendijk, and Tomas Toft. Privacy-preserving face recognition. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 235–253. Springer, 2009.
- [36] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33rd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24*, pages 201–210, 2016.

- [37] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. Privacy-preserving classification on deep neural network. *IACR Cryptology ePrint Archive*, 2017:35, 2017.
- [38] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Cryptodl: Deep neural networks over encrypted data. *CoRR*, abs/1711.05189, 2017. URL <http://arxiv.org/abs/1711.05189>.
- [39] P. Mohassel and Y. Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38, May 2017.
- [40] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: Efficient and private neural network training. (PETS 2019), February 2019.
- [41] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1651–1669, 2018.
- [42] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18*, pages 35–52, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5693-0. doi: 10.1145/3243734.3243760. URL <http://doi.acm.org/10.1145/3243734.3243760>.
- [43] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, ASIACCS '18*, pages

- 707–721, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5576-6. doi: 10.1145/3196494.3196522. URL <http://doi.acm.org/10.1145/3196494.3196522>.
- [44] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17*, pages 619–631, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4946-8. doi: 10.1145/3133956.3134056. URL <http://doi.acm.org/10.1145/3133956.3134056>.
- [45] Cornejo Mario and Poumeyrol Mathieu. Benchmarking Paillier Encryption. <https://medium.com/snips-ai/benchmarking-paillier-encryption-15631a0b5ad8>, 2018. [Online; accessed July-2018].
- [46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS'12*, pages 1097–1105, USA, 2012. Curran Associates Inc. URL <http://dl.acm.org/citation.cfm?id=2999134.2999257>.
- [47] Abdul Ghafoor and Sead Muftic. Cryptonet: Softwareprotection and secure execution environment. *Int. J. Computer Science and Network Security*, 10(2):19–26, 2010.
- [48] Arif Ahmed and Ejaz Ahmed. A survey on mobile edge computing. In *Intelligent Systems and Control (ISCO), 2016 10th International Conference on*, pages 1–8. IEEE, 2016.

- 
- [49] L. Yang, H. Zhang, M. Li, J. Guo, and H. Ji. Mobile edge computing empowered energy efficient task offloading in 5g. *IEEE Transactions on Vehicular Technology*, PP(99):1–1, 2018. ISSN 0018-9545. doi: 10.1109/TVT.2018.2799620.
- [50] Cong Wang, Ning Cao, Jin Li, Kui Ren, and Wenjing Lou. Secure ranked keyword search over encrypted cloud data. In *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems, ICDCS '10*, pages 253–262, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4059-7. doi: 10.1109/ICDCS.2010.34.
- [51] Wenhai Sun, Bing Wang, Ning Cao, Ming Li, Wenjing Lou, Y. Thomas Hou, and Hui Li. Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, ASIA CCS '13*, pages 71–82, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1767-2. doi: 10.1145/2484313.2484322.
- [52] Bing Wang, Shucheng Yu, Wenjing Lou, and Y Thomas Hou. Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. In *INFOCOM, 2014 Proceedings IEEE*, April 2014.
- [53] Phan NhatHai, Wu Xintao, Hu Han, and Dou Dejing. Adaptive laplace mechanism: Differential privacy preservation in deep learning. In *Proceedings of the 2017 IEEE International Conference on Data Mining, ICDM '17*. IEEE, 2017.
- [54] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications*

- Security*, CCS '16, pages 308–318, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4139-4.
- [55] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, SP '00, pages 44–55, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0665-8.
- [56] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *Advances in Cryptology - EURO-CRYPT 2004*, pages 506–522. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-21935-4. doi: 10.1007/978-3-540-24676-3\_30.
- [57] Google. Cloud Vision API. <https://cloud.google.com/vision/>, 2016.
- [58] Scale. Scale Image Annotation API. <https://www.scaleapi.com/image-annotation>.
- [59] Marius Muja and David G Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.
- [60] Xin-Jing Wang, Lei Zhang, Feng Jing, and Wei-Ying Ma. Annosearch: Image auto-annotation by search. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1483–1490. IEEE, 2006.
- [61] Bryan C Russell, Antonio Torralba, Kevin P Murphy, and William T Freeman. Labelme: a database and web-based tool for image annotation. *International journal of computer vision*, 77(1-3):157–173, 2008.



- [62] Yashaswi Verma and C. V. Jawahar. Image annotation using metric learning in semantic neighbourhoods. In *Proceedings of the 12th European Conference on Computer Vision - Volume Part III, ECCV'12*, pages 836–849, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-33711-6. doi: 10.1007/978-3-642-33712-3\_60. URL [http://dx.doi.org/10.1007/978-3-642-33712-3\\_60](http://dx.doi.org/10.1007/978-3-642-33712-3_60).
- [63] Olivier Chapelle, Patrick Haffner, and Vladimir N Vapnik. Support vector machines for histogram-based image classification. *IEEE transactions on Neural Networks*, 10(5):1055–1064, 1999.
- [64] Claudio Cusano, Gianluigi Ciocca, and Raimondo Schettini. Image annotation using svm. In *Electronic Imaging 2004*, pages 330–338. International Society for Optics and Photonics, 2003.
- [65] Rui Shi, Huamin Feng, Tat-Seng Chua, and Chin-Hui Lee. An adaptive image content representation and segmentation approach to automatic image annotation. In *International conference on image and video retrieval*, pages 545–554. Springer, 2004.
- [66] Aditya Vailaya, Mário AT Figueiredo, Anil K Jain, and Hong-Jiang Zhang. Image classification for content-based indexing. *IEEE transactions on image processing*, 10(1):117–130, 2001.
- [67] Shi Rui, Wanjun Jin, and Tat-Seng Chua. A novel approach to auto image annotation based on pairwise constrained clustering and semi-naive bayesian model. In *Multimedia Modelling Conference, 2005. MMM 2005. Proceedings of the 11th International*, pages 322–327. IEEE, 2005.

- [68] Hongchao Zhou and Gregory Wornell. Efficient homomorphic encryption on integer vectors and its applications. In *Information Theory and Applications Workshop (ITA), 2014*, pages 1–9. IEEE, 2014.
- [69] S. Rane, W. Sun, and A. Vetro. Privacy-preserving approximation of l1 distance for multimedia applications. In *2010 IEEE International Conference on Multimedia and Expo*, pages 492–497, July 2010. doi: 10.1109/ICME.2010.5583030.
- [70] B Johnson William and Lindenstrauss Joram. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26:189–206, 1984.
- [71] Hugo Jair Escalante, Carlos A. Hernández, Jesus A. Gonzalez, A. López-López, Manuel Montes, Eduardo F. Morales, L. Enrique Sucar, Luis Villaseñor, and Michael Grubinger. The segmented and annotated iapr tc-12 benchmark. *Comput. Vis. Image Underst.*, 114(4):419–428, April 2010. ISSN 1077-3142. doi: 10.1016/j.cviu.2009.03.008. URL <http://dx.doi.org/10.1016/j.cviu.2009.03.008>.
- [72] MIT Computer Science and Artificial Intelligence Laboratory. Labelme dataset. <http://labelme.csail.mit.edu/Release3.0/browserTools/php/dataset.php>, 2017.
- [73] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.
- [74] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
- [75] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O’Neill. Order-preserving symmetric encryption. In *Proceedings of the 28th Annual International*

- Conference on Advances in Cryptology: The Theory and Applications of Cryptographic Techniques*, EUROCRYPT '09, pages 224–241, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-01000-2. doi: 10.1007/978-3-642-01001-9\_13. URL [http://dx.doi.org/10.1007/978-3-642-01001-9\\_13](http://dx.doi.org/10.1007/978-3-642-01001-9_13).
- [76] Daniel S. Roche, Daniel Apon, Seung Geol Choi, and Arkady Yerukhimovich. Pope: Partial order preserving encoding. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1131–1142, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4139-4. doi: 10.1145/2976749.2978345. URL <http://doi.acm.org/10.1145/2976749.2978345>.
- [77] Zvika Brakerski, Craig Gentry, and Shai Halevi. Packed ciphertexts in lwe-based homomorphic encryption. In *16th International Conference on Practice and Theory in Public-Key Cryptography (PKC)*, pages 1–13, February 2013.
- [78] Jonathan Katz and Yehuda Lindell. *Chapter 11, Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2007.
- [79] B. Yao, F. Li, and X. Xiao. Secure nearest neighbor revisited. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 733–744, April 2013. doi: 10.1109/ICDE.2013.6544870.
- [80] NumPy Developers. Numpy. *NumPy Numpy. Scipy Developers*, 2013.
- [81] Gary Bradski et al. The opencv library. *Doctor Dobbs Journal*, 25(11):120–126, 2000.
- [82] Filip Wasilewski. PyWavelets - Wavelet Transforms in Python. <https://pywavelets.readthedocs.io/en/latest/>, 2006.

- [83] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [84] Guang-Hai Liu and Jing-Yu Yang. Content-based image retrieval using color difference histogram. *Pattern Recognition*, 46(1):188–198, 2013.
- [85] Yifan Tian, Yantian Hou, and Jiawei Yuan. Capia: Cloud assisted privacy-preserving image annotation. In *Communications and Network Security (CNS), 2017 IEEE Conference on*, pages 1–9. IEEE, 2017.
- [86] M. Verhelst and B. Moons. Embedded deep neural network processing: Algorithmic and processor techniques bring deep learning to iot and edge devices. *IEEE Solid-State Circuits Magazine*, 9(4):55–65, Fall 2017. ISSN 1943-0582. doi: 10.1109/MSSC.2017.2745818.
- [87] S. Kodali, P. Hansen, N. Mulholland, P. Whatmough, D. Brooks, and G. Y. Wei. Applications of deep neural networks for ultra low power iot. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 589–592, Nov 2017. doi: 10.1109/ICCD.2017.102.
- [88] Matt Burns. Arm chips with Nvidia AI could change the Internet of Things. <https://techcrunch.com/2018/03/27/arm-chips-will-with-nvidia-ai-could-change-the-internet-of-things/>, 2018. [Online; accessed July-2018].
- [89] Mohammadi Mehdi, Al-Fuqaha Ala, Sorour Sameh, and Guizani Mohsen. Deep learning for iot big data and streaming analytics: A survey. *arXiv:1712.04301*, 2017.

- [90] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823, June 2015. doi: 10.1109/CVPR.2015.7298682.
- [91] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016. doi: 10.1109/CVPR.2016.90.
- [92] J. Zhou, Z. Cao, X. Dong, and A. V. Vasilakos. Security and privacy for cloud-based iot: Challenges. *IEEE Communications Magazine*, 55(1):26–33, January 2017. ISSN 0163-6804. doi: 10.1109/MCOM.2017.1600363CM.
- [93] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, Oct 2016. ISSN 2327-4662. doi: 10.1109/JIOT.2016.2579198.
- [94] Balachandra Reddy Kandukuri, Atanu Rakshit, et al. Cloud security issues. In *Services Computing, 2009. SCC'09. IEEE International Conference on Services Computing*, pages 517–520. IEEE, 2009.
- [95] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.
- [96] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4013–4021, 2016.

- [97] Peter Lindstrom and Martin Isenburg. Fast and efficient compression of floating-point data. *IEEE transactions on visualization and computer graphics*, 12(5): 1245–1250, 2006.
- [98] Wikipedia. Convolutional neural network . [https://en.wikipedia.org/wiki/Convolutional\\_neural\\_network](https://en.wikipedia.org/wiki/Convolutional_neural_network). [Online; accessed July-2018].
- [99] Jason Cong and Bingjun Xiao. *Minimizing Computation in Convolutional Neural Networks*, pages 281–290. Springer International Publishing, Cham, 2014. ISBN 978-3-319-11179-7. doi: 10.1007/978-3-319-11179-7\_36.
- [100] Jonathan Katz and Yehuda Lindell. *Chapter 3.3, Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2007.
- [101] J. Lee, J. Wang, D. Crandall, S. Šabanović, and G. Fox. Real-time, cloud-based object detection for unmanned aerial vehicles. In *2017 First IEEE International Conference on Robotic Computing (IRC)*, pages 36–43, April 2017. doi: 10.1109/IRC.2017.77.
- [102] Airdata UAV. Drone Flight Stats. <https://airdata.com/blog/2017/drone-flight-stats-part-1>, 2018. [Online; accessed July-2018].
- [103] Shmuel Winograd. *Arithmetic complexity of computations*, volume 33. Siam, 1980.
- [104] Shmuel Winograd. On multiplication of polynomials modulo a polynomial. *SIAM Journal on Computing*, 9(2):225–229, 1980.
- [105] Lawrence Ibarria, Peter Lindstrom, Jarek Rossignac, and Andrzej Szymczak. Out-of-core compression and decompression of large n-dimensional scalar fields. In *Computer Graphics Forum*, volume 22, pages 343–348. Wiley Online Library, 2003.

- 
- [106] Michael Schindler. A fast renormalisation for arithmetic coding. In *Proceedings DCC'98 Data Compression Conference (Cat. No. 98TB100225)*, page 572. IEEE, 1998.
- [107] B. P. Rimal, D. P. Van, and M. Maier. Mobile edge computing empowered fiber-wireless access networks in the 5g era. *IEEE Communications Magazine*, 55(2): 192–200, February 2017. ISSN 0163-6804. doi: 10.1109/MCOM.2017.1600156CM.
- [108] Paul, Norbury. Now Trending: SD Memory Cards. [https://www.sdcard.org/press/thoughtleadership/180118\\_Now\\_Trending\\_SD\\_Memory\\_Cards.html](https://www.sdcard.org/press/thoughtleadership/180118_Now_Trending_SD_Memory_Cards.html), 2018. [Online; accessed July-2018].
- [109] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.
- [110] Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In Martijn Stam, editor, *Cryptography and Coding*, pages 45–64, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-45239-0.
- [111] Raspberry Pi Dramble. Power Consumption Benchmarks. <http://www.pidramble.com/wiki/benchmarks/power-consumption>, 2018. [Online; accessed April-2019].
- [112] Antonin Guttman. *R-trees: A dynamic index structure for spatial searching*, volume 14. ACM, 1984.
- [113] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

- 
- [114] Jakub Konečný, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.
- [115] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 2512–2520. IEEE, 2019.