

# Implementation Of Real-Time Learning On Homomorphically Encrypted Visual Inputs

by

Emtiaz MD Tafsir Bhuiyan

16301049

Mushfiqur Rahman

20241040

Sudipta Mondal

17301224

Sadman Warech

16301115

A thesis submitted to the Department of Computer Science and Engineering  
in partial fulfillment of the requirements for the degree of  
B.Sc. in Computer Science

Department of Computer Science and Engineering  
Brac University  
June 2021

© 2021. Brac University  
All rights reserved.

# Declaration

It is hereby declared that

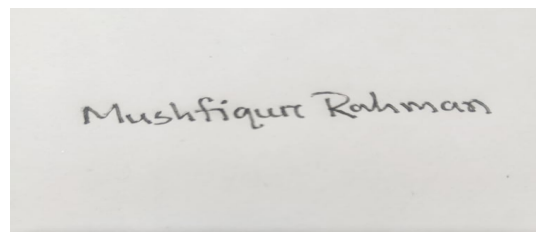
1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

**Student's Full Name & Signature:**



---

Emtiaz MD Tafsir Bhuiyan  
16301049



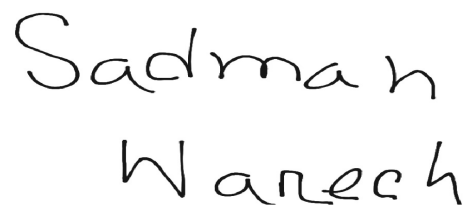
---

Mushfiquir Rahman  
20241040



---

Sudipta Mondal  
17301224



---

Sadman Warech  
16301115

# Approval

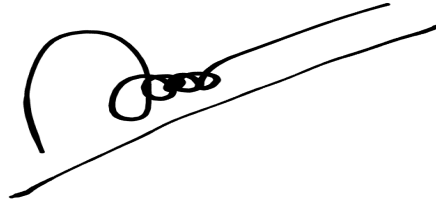
The thesis titled “Implementation Of Real-Time Learning On Homomorphically Encrypted Visual Inputs” submitted by

1. Emtiaz MD Tafsir Bhuiyan (16301049)
2. Mushfiqur Rahman (20241040)
3. Sudipta Mondal (17301224))
4. Sadman Warech (16301115)

Of Spring, 2021 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on June 06, 2021.

## Examining Committee:

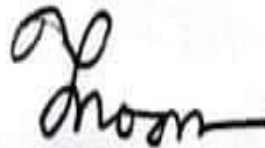
Supervisor:  
(Member)



---

Dr. Muhammad Iqbal Hossain  
Assistant Professor  
Department of Computer Science and Engineering  
BRAC University

Co-Supervisor:  
(Member)



---

Dr. Jannatun Noor Mukta  
Lecturer  
Department of Computer Science and Engineering  
BRAC University

Program Coordinator:  
(Member)

---

Dr. Md. Golam Rabiul Alam  
Associate Professor  
Department of Computer Science and Engineering  
BRAC University

Head of Department:  
(Chair)

---

Sadia Hamid Kazi  
Chairperson and Associate Professor  
Department of Computer Science and Engineering  
BRAC University

## **Ethics Statement**

The thesis is written in strict accordance with research ethics guidelines as well as BRAC University's standards and procedures. We have employed data from primary sources in our thesis. We are ensuring that we're using references and in-text citations correctly. We, the four authors, accept full responsibility for the infractions of the thesis code. We read many websites, YouTube tutorials, and research papers to solve problems. We also sought the assistance of some of our university's professors. Finally, we affirm that we are thankful to all of the people who have assisted us. Our work complies with the BRAC university's ethical standards.

# Abstract

It's challenging to provide security for cloud-based services, especially for cloud processing services, due to the fact that typical encryption techniques do not allow for calculation on encrypted data. The formation of Homomorphic Encryption techniques shows significant possibilities of incorporating encrypted computation on cloud infrastructures. This enables owners to outsource computation over confidential data to cloud vendors. Control and synthesis tasks of sensitive systems like traffic light control, article recommendation for online users and potentially, robot's action determination can be delegated to a cloud-based Reinforcement Learning agent. In this study, we designed two Deep Reinforcement Learning agents that work on ciphertexts using Homomorphic Encryption. Both agents take encrypted state images and produce encrypted actions. One learns on plain data but evaluates on encrypted inputs, while the other one operates fully on encrypted space. The performance of both agents is compared against plaintext RL agents with identical parameters. The paper also describes possible architectures for such systems.

**Keywords:** Homomorphic Encryption; Privacy preserving; Reinforcement Learning; Deep Q Learning

## **Dedication**

This research will be dedicated to our parents. We may not be able to complete our studies without their aid. We also want to dedicate the study to our friends who helped us improve. Throughout the year, our supervisor guided us. We'd like to dedicate it to him as well.

## Acknowledgement

All gratitude to the Almighty for allowing us to finish our thesis without any serious setbacks. First and foremost, we would want to express our gratitude to our cherished family members, to whom we will be eternally grateful. Furthermore, we want to express our gratitude to our supervisor, Dr. Muhammad Iqbal Hossain, for all of his cooperation and consistent guidance. Finally, we would like to extend our thanks to all of the faculty members and staff for providing us with such a beautiful learning atmosphere in which we were able to properly develop ourselves as well as this research. We are currently on the verge of graduating thanks to their kind assistance and prayers.



# Table of Contents

Declaration	i
Approval	ii
Ethics Statement	iv
Abstract	v
Dedication	vi
Acknowledgment	vii
Table of Contents	viii
List of Tables	x
List of Figures	xi
Nomenclature	xii
<b>1 Introduction</b>	<b>1</b>
1.1 Problem Statement . . . . .	1
1.2 Motivation . . . . .	2
1.3 Research Objective . . . . .	2
<b>2 Background</b>	<b>4</b>
2.1 Related Works . . . . .	4
2.2 Methodology Background . . . . .	10
2.2.1 Encryption . . . . .	10
2.2.2 Homomorphic encryption . . . . .	10
2.2.2.1 Partially Homomorphic Encryption (PHE) . . . . .	11
2.2.2.2 Somewhat Homomorphic Encryption (SWHE) . . . . .	11
2.2.2.3 Fully Homomorphic Encryption (FHE) . . . . .	11
2.2.3 Machine Learning . . . . .	14
2.2.3.1 Supervised Learning . . . . .	15
2.2.3.2 Unsupervised Learning . . . . .	15
2.2.4 Reinforcement Learning . . . . .	16
2.2.4.1 Q Learning . . . . .	18
2.2.4.2 State-Action-Reward-State-Action (SARSA) . . . . .	19
2.2.4.3 Deep Reinforcement Learning . . . . .	20

2.2.5	Neural Network . . . . .	20
2.2.6	Convolution Neural Network . . . . .	21
<b>3</b>	<b>Proposed Method</b>	<b>23</b>
3.1	System Architecture . . . . .	23
3.2	Used Algorithms . . . . .	25
3.2.1	Deep Q Learning . . . . .	26
3.2.2	CNN . . . . .	26
3.2.3	CKKS Scheme . . . . .	26
3.3	Used Libraries . . . . .	27
3.3.1	Microsoft SEAL . . . . .	27
3.3.2	TenSEAL . . . . .	27
3.3.3	PyTorch . . . . .	28
3.3.4	Torchvision . . . . .	28
3.3.5	Protocol Buffer . . . . .	28
3.3.6	Gym . . . . .	29
<b>4</b>	<b>Implementation and Experiment</b>	<b>30</b>
4.1	Environment Description . . . . .	30
4.2	Data Preprocessing . . . . .	31
4.3	Parameter Selection . . . . .	33
4.3.1	CNN Layers . . . . .	34
4.3.2	Encryption Parameters . . . . .	35
4.4	Data Encryption . . . . .	36
4.5	Agent structure . . . . .	37
4.5.1	Plain-Encrypted Agent . . . . .	39
4.5.2	Pure Encrypted Agent . . . . .	39
<b>5</b>	<b>Result Analysis</b>	<b>42</b>
5.1	Cost Analysis . . . . .	47
5.2	Performance Analysis . . . . .	48
<b>6</b>	<b>Conclusion and Future Works</b>	<b>51</b>
6.1	Conclusion . . . . .	51
6.2	Future Works . . . . .	51
	<b>Bibliography</b>	<b>53</b>

# List of Tables

2.1	Comparison Among Different Types of Neural Networks . . . . .	22
5.1	Description of Pure Encrypted Agents . . . . .	47
5.2	Description of Pure Encrypted Agents Equivalents . . . . .	47
5.3	Table of Unit Cost Analysis . . . . .	48
5.4	Table of Score achieved by the agents . . . . .	49

# List of Figures

2.1	High Level CKKS Procedure . . . . .	13
2.2	Simplified Reinforcement Learning Procedure . . . . .	16
2.3	RL Algorithm Tree . . . . .	17
2.4	Q learning Flowchart . . . . .	18
2.5	Simple Presentation of Policy Iteration . . . . .	19
3.1	Plain-Encrypted Architecture . . . . .	24
3.2	Pure Encrypted Architecture . . . . .	24
4.1	Sample Extracted Raw Image from GUI . . . . .	31
4.2	Sample Sliced Image After First Stage of Preprocessing . . . . .	31
4.3	Sample Zoomed Image After Second Stage of Preprocessing . . . . .	32
4.4	Sample Grayscaled Image After Third Stage of Preprocessing . . . . .	32
4.5	Sample Resized Image After Fourth Stage of Preprocessing . . . . .	33
4.6	Sample Transition Data as Image . . . . .	33
4.7	Configuration of CNN . . . . .	35
4.8	High Level Custom Network of Pure Encrypted Agent . . . . .	41
5.1	Reference Agent's Durations Over Episodes . . . . .	43
5.2	Reference Agent's Average Most Recent Durations Over Episodes . . . . .	43
5.3	Reference Agent . . . . .	43
5.4	<i>PlE</i> Learning Phase Durations Over Episodes . . . . .	44
5.5	<i>PlE</i> Learning Phase Average Most Recent Durations Over Episodes . . . . .	44
5.6	<i>PlE</i> Evaluation Phase Durations Over Episodes . . . . .	44
5.7	<i>PlE</i> Evaluation Phase Average Most Recent Durations Over Episodes . . . . .	44
5.8	Plain-Encrypted Agent . . . . .	44
5.9	<i>PuE</i> – 1 Durations Over Episodes . . . . .	45
5.10	<i>PuE</i> – 1 Average Most Recent Durations Over Episodes . . . . .	45
5.11	<i>PuE</i> – 2 Durations Over Episodes . . . . .	45
5.12	<i>PuE</i> – 2 Average Most Recent Durations Over Episodes . . . . .	45
5.13	Pure Encrypted Agents . . . . .	45
5.14	<i>PuE</i> – 1 Plain ref Durations Over Episodes . . . . .	46
5.15	<i>PuE</i> – 1 Plain ref Average Most Recent Durations Over Episode . . . . .	46
5.16	<i>PuE</i> – 2 Plain ref Durations Over Episodes . . . . .	46
5.17	<i>PuE</i> – 2 Plain ref Average Most Recent Durations Over Episodes . . . . .	46
5.18	<i>PuE</i> plaintext equivalent Agents . . . . .	46

# Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

*ANN* Artificial Neural Network

*CKKS* Cheon-Kim-Kim-Song

*CNN* Convolutional Neural Network

*DQN* Deep Q Network

*FHE* Fully Homomorphic Encryption

*GUI* Graphical User Interface

*HE* Homomorphic Encryption

*LWE* Learning With Errors

*ML* Machine Learning

*NN* Neural Network

*PHE* Partially Homomorphic Encryption

*PIE* Plain-Encrypted Agent

*PuE* Pure Encrypted Agent

*RL* Reinforcement Learning

*RLWE* Ring Learning With Errors

*RNN* Recurrent Neural Network

*SWHE* Somewhat Homomorphic Encryption

# Chapter 1

## Introduction

### 1.1 Problem Statement

Cloud computing infrastructures have paved a new way in modern IT and business sectors by lending storage and computation power to customers of different levels and backgrounds on an as-needed basis. Many of the electronic goods in today's market, such as Alexa, google lens, and countless IoT systems are using cloud back-end to improve the quality of customer service. Although it enables people to share computation and storage resources, it also allows resource owners/vendors to observe or collect incoming and outgoing data. It raised huge concerns in the world with regards to protecting user's privacy. Unlike cloud storage, cloud computing systems cannot utilize encryption to ensure confidentiality since standard encryption schemes like AES, DES, and Blowfish, do not support computation over ciphertexts. Hence, even if end-to-end encryption is adopted, nothing can stop vendors from taking a peek. Recent trends show that a large portion of the population of the world is dependent on ML (Machine Learning) based cloud services in one way or another. People are benefiting from being able to use ML features without having to own expensive computer hardware. Real-time learning agents are known for excelling at control and optimization problems. Cloud-based learning agents prove to be a cheap and feasible solution to many optimization problems as it eliminates resource constraints. Still, businesses refuse to use third-party cloud platforms to prevent exposure to sensitive control data. Since private information can be a valuable commodity in today's culture, a proper privacy-protecting system is a must-have feature for cloud-based control services.

Homomorphic encryption, envisioned by Rivest et al [1], which allows a third party to execute arbitrary functions on ciphertexts without decrypting them, can be considered as a promising solution for resolving data privacy issues in cloud-hosted ML services. Users with confidential data can upload only ciphertexts to cloud computing platforms using HE cryptography, obviating the need to provide any information regarding the decryption key. Noise growth and evaluation depth is a limiting factor for HE schemes. A few [2–4] workarounds have been suggested resulting in Fully Homomorphic Encryption (FHE). However, the computation cost was still beyond the scope of real-world application. More recent members of FHE schemes like CKKS [5] allow arithmetic of approximate numbers and it is more computation

friendly than most of the schemes. The multiplicative depth of CKKS ciphertexts depends heavily on encryption parameters where one has to systematically balance security, accuracy and depth to evaluate any given circuit. Arranging homomorphic operations to evaluate ML circuits is a challenge where the former requires evaluation circuits to be simple and linear and the latter works on complex functions and non-linearity. In the case of real-time learning agents, where time constraints carry significant importance, the window of application is narrower.

## 1.2 Motivation

Cloud computing is rapidly expanding its scalability in today's world. Despite its popularity going upwards, its privacy protection has been a big concern in this sector. As a result, the transitions towards Machine learning featured cloud computing are also seen in high numbers. Yet, there is a lack of robustness evident, and we were motivated to research a detailed review of potential vulnerabilities and the recent implementation of security measures. Our basic idea came out by the depth analysis of encryption lacking in the cloud computing architecture. Additionally, we were inspired by the past events of data breaching & attacks on cloud computing platforms. A big part of this sector is discouraging third-party cloud computing platforms because of personal & business end-to-end data protection agreements. This also motivated us to initiate this research which is a unique approach to create more trust for the users towards cloud computing platforms. The uniqueness of our study lies in the homomorphic encryption, which we believe will open up vast possibilities in the context of cloud-hosted ML services.

## 1.3 Research Objective

The objective of our research is to work towards a successful implementation of cloud-hosted real-time deep reinforcement learning agents, which will be injected by the robust system of homomorphic encryption procedure. To increase the security features as different encryption methods were not compatible with cloud computing architecture which was Machine learning driven, we were inspired to conduct this research. Subsequently, our primary goal is to design an architecture that supports secure real-time learning by utilizing state-of-the-art cryptographical elements through analyzing affiliated algorithms thoroughly. In the first step, our plan was to think about how to solve the stated problem. As a result, the approach we thought of was a privacy preserving real-time learning agent, which we will attempt to implement at a later stage of the research. Moving on, we will emphasize the perfect algorithm after a thorough analysis of reinforcement learning methods. Finally, we will work towards a solution of shaping the learning agent to fit into the scope of homomorphic cryptography. Additionally, we wish to demonstrate the efficiency of homomorphic cryptography and find out the most fitted scheme with our model. So, the core objective of this research is threefold:

1. Defining system architectures that support real-time learning.
2. Design learning agents that fit into the defined architectures.
3. In depth comparison of the designed agents against equivalent regular learning agents.



# Chapter 2

## Background

### 2.1 Related Works

Encryption has been a good way for keeping confidential data private. Among the most fundamental disadvantages of this method is that an information system using encrypted data can only store or receive the data for the user. Some of the really complicated processes appear to need decrypting the data first. The pioneer of the RSA encryption scheme Rivest et al. [1] have claimed that privacy homomorphism is an innovative approach in preserving the confidentiality of sensitive data. They are, however, restricted in their application since comparisons may not be incorporated in the list of functions to be employed. Decades have passed since the dawn of the idea without any significant progression. Until the first homomorphic encryption scheme that was vented by Gentry et al. in 2009 [2], and since then researchers have established a number of latest and more effective fully homomorphic schemes. Gentry proposed a method for assessing circuits on encrypted data that does not need decryption. Their approach was based on three phases. They mentioned a bootstrappable system based on ideal grids for public key encryption. Afterwards, Brakerski proposed a new homomorphic encryption scheme that's easy to define and evaluate, but its security is restricted to the worst-case complexity of issues on ideal lattices. They used Gentry's [2] standard "squashing" and "bootstrapping" methods to convert it into a completely homomorphic encryption scheme. Their scheme is based on another researcher's work, Lyubashevsky's ring learning with errors inference. The RLWE assumption reduces worst-case problems on perfect lattices and helps them to abstract out the lattice interpretation entirely. Brakerski-Gentry [4] proposed a completely unique procedure to fully homomorphic encryption (FHE) that greatly enhances efficiency and bases protection on infirm assumptions. A key mathematical contribution in their work may be a new method of designing leveled, totally homomorphic encryption schemes (capable of testing arbitrary polynomial-size circuits of a-priori bounded depth), without Gentry's bootstrapping technique. They specifically provided an alternative of FHE systems centered on learning with inaccuracy (LWE) or Ring LWE (RLWE) issues with  $2\lambda$  of security against known attacks. Such as, a leveled FHE scheme which will test depth- $L$  arithmetic circuits which uses  $O(\lambda \cdot L^3)$  per-gate computation, quasilinear within the protection

parameter. For an exponential estimation factor in  $L$ , security is determined by RLWE. The bootstrapping method would not be used in this architecture. In other words, a leveled FHE method that evaluates depth- $L$  arithmetic circuits (made up of fan-in 2 gates) utilizing  $O(\lambda^2)$  per-gate calculation that is irrespective of  $L$ .

Very recently, Cheon-Kim et al. [5] proposed a way to create a homomorphic encryption structure for estimated arithmetic. The essential idea is to introduce noise after large figures that convey a fundamental notion. This noise was previously introduced to the plaintext for security reasons, but it is now thought to be a component of error occurring during approximated calculations that is decreased including the plaintext by resizing. They also suggested a new batching methodology for an RLWE-based building. They demonstrated that their method could be expanded to evaluate complex functions like multiplicative inverse, exponential function, logistic function, and discrete Fourier transform effectively. They also discovered that accuracy failure while assessment is limited by the level of a circuit and can only reach one extra bit when compared to unencrypted estimation arithmetic such as floatingpoint processes. In another research [6], they extended their leveled homomorphic encryption structure. They proposed a new strategy to refresh low-level ciphertexts that supported Gentry's [4] bootstrapping procedure. They provided an efficient evaluation technique employing a scaled linear model similar to the modular reduction procedure. For every cycle, their approach only needs single homomorphic multiplication, hence the general processing cost rises linearly with the depth of the decryption circuit. They also demonstrated the way to encrypt packed ciphertexts using an open-source version of the RLWE construction. Meanwhile, Cheon's colleague Y. Song eventually joined in another research [7] where they improved Cheon's bootstrapping result. They reduced amortized bootstrapping time per plaintext slot from 1 second to 0.01 second. They used a sophisticated level-collapsing methodology for assessing DFT-like linear transformations on a ciphertext to obtain this outcome.

Regarding online privacy, Agarwal & Srikant [8] mentioned that the development of technologies that address privacy issues will be a fruitful route for future data analysis. Besides, they attempted to develop accurate models without access to precise information in individual data records. And they had considered the case of building a decision-tree classifier from training data that has had the values of individual records tampered with. Although it is not possible to accurately estimate original values in individual data records, they have proposed a novel reconstruction procedure to accurately estimate the distribution of original data values. Considering Machine Learning frameworks, when training a supervised Machine Learning classifier, many works concentrate on the data protection problem of the Internet of Things (IoT). The majority of current solutions presume that the classifier's training data can be accessed safely from various IoT data providers, says Haque-Hasan et al. [9] in their research. This paper proposes secure K-NN to guard data privacy when training a K-Nearest Neighbour (K-NN) classifier with IoT data from various entities. It employs a cryptosystem referred to as Paillier so as to guard all participants (i.e., IoT data analyst  $C$  and IoT data provider  $P$ ) Data privacy may be a concern when  $C$  analyzes the IoT data of  $P$ . Both participants' privacy issues arise and require a trusted third party. It shows that secure K-NN doesn't need any trusted third party at the time of interaction. The goal of k-Nearest Neighbor (k-NN)

extraction is to locate the  $k$  objects closest to the query items. Outlier detection, clustering, and  $k$ -NN categorization are just a few of the data mining methods that might benefit from it. The privacy-preserving shared  $k$ -NN is being developed to tackle the challenge while maintaining individual secrecy. On physically segmented data a variety of various confidentiality  $k$ -NN mining methods have been developed; however, they fail to handle the privacy problem when the number of participants surpasses two. As there are more than two parties, they proposed a set of principles for dealing with the issue of privacy. The protocols are built with the probability public-key cryptographic protocol and the interactive cryptosystem in mind as the essential confidentiality architecture. The security of the protocols is demonstrated using the Secure Multi-party Computation theory [10]. Author Barni [11] presented a unique approach with privacy preserving neural network training. He mentioned that even though there have been many studies related to protecting the privacy of supervised model learning, limited effort was done to broaden neural network learning through a privacy protection protocol. Neural networks are well-known for a number of implementations, including using voice recognition for a strong algorithm. They went into detail about privacy-preserving categorization as well as modeling with neural networks over horizontally segmented dataset. They considered a scenario involving two semi-honest but interested parties. They generalize with procedures for dependable value and stable matrix operating the algorithm of neural network categorization proposed by Rumelhart et al. [12]. The expanded algorithm does not guarantee confidentiality in the sense that Goldreich [13] defines, nevertheless they showed that the disclosed knowledge isn't tied to a certain entity and can even be obtained by contrasting data locally, resulting in the final version.

The topic of stable data analysis using a neural network (NN) is being debated. Secure processing denotes the probability that the NN owner has no knowledge of the processed data because it is delivered to him in encrypted format. At the same time, the NN is shielded because its owner will be unable to reveal the information contained inside it. The type of security considered ensures that the data given to the network, and also the network weights and activation functions are kept secret. By properly inputting fake data to any point of the proposed protocol, extra attention is taken to avoid any disclosure of sensitive information that could enable a malicious user to gain access to the NN secrets. Compared to earlier works in this area, the interaction between the user and the NN owner is held to a minimum, and no multiparty computation protocols are used [14]. As a result of the growth of distributed computing environments, numerous learning challenges also have to cope with dispersed data input. To improve learning collaborations, it is critical to resolve each data holder's privacy concerns by applying the privacy preservation concept to original learning algorithms. Chen and Jhong [15] focused on maintaining privacy in multilayered neural networks, a fundamental learning paradigm. They introduced a two-party disbursed backpropagation technique that allows a neural network to be trained without any side having to share her data with the other. They offered a thorough review of their algorithms' correctness and security. Experiments on different real-world data sets are used to validate the effectiveness of their algorithms.

If consumers encrypt the data they transfer to the cloud, privacy concerns will be alleviated. The cloud can still execute realistic calculations on the data if the en-

encryption strategy is homomorphic, says Lauter-Neherig et al. [16] in their paper “Can Homomorphic Encryption Be Practical?”. He claims that all fully homomorphic encryption algorithms currently known have a fair path to go before they can be utilized in reality. A variety of meaningful applications in the medical, financial, and commercial domains only need that the scheme be “somewhat” homomorphic, according to the author. He claims that the method is very effective and that the ciphertexts are relatively short. “With the same degree of security and homomorphic strength, our unoptimized magma implementation performs admirably even optimized pairing-based schemes”, he says. The thesis of the author has been published in the open-access journal *Cybernetics and Security*.

Demonstrating the effectiveness of pre-trained neural networks in practice, Dahl-Yu et al. [17] addressed a model for large-vocabulary speech recognition that leverages recent developments in utilizing deep belief networks for phone recognition. They defined a pre-trained deep neural network hidden Markov model (DNN-HMM) hybrid architecture that trains the DNN to get a distribution over senones (tied triphone states) as its performance. The deep belief network pre-training algorithm may be stable and sometimes beneficial thanks to initializing deep neural networks generatively, which will assist in optimization and reduce generalization errors. They described the majority of our model’s components, explained the protocol for applying CDDNN-HMMs to LVSR, and discussed the effects of various modeling choices on results. Experiments on a challenging business search dataset demonstrate that CDDNN-HMMs will significantly outperform the traditional context-dependent Gaussian mixture model (GMM)-HMMs, with an absolute sentence accuracy improvement of 5.8 percent and 9.2 percent (or relative error reduction of 16.0 percent and 23.2 percent) over the CD-GMM-HMMs trained using the minimum phone error rate (MPE) and maximum-likelihood (ML) criteria, respectively.

Graepel & Lauter [18] addressed the issue with incorporating Homomorphic encryption scheme with cloud hosted ML services by mentioned that, machine learning comprises two phases, the training stage and the classification stage, one or both of which may be exported to the cloud, and an intermediate deterministic verification stage to verify and validate the trained model. Besides, encrypting data before transferring it to the cloud is one way to protect the integrity of data. And they suggested a confidential protocol for machine learning functions, named ML-Confidential, based on Homomorphic Encryption. In a paper, Aslett-Holmes et al. [19] introduced two new predictive machine learning methods designed to train on completely homomorphic encrypted (FHE) files. The implementation of FHE schemes following Gentry [2] opens up the possibility of privacy protecting statistical machine learning research and modeling of encrypted data without breaching security constraints. They proposed customized algorithms for applying extremely random forests, involving a modern cryptographic stochastic fraction estimator, and naïve bayes, involving a semi-parametric model for the class decision boundary, and demonstrating how they can be used to learn and forecast from encrypted results. They showed that these approaches work competitively on a number of classification data sets and offer comprehensive knowledge regarding the statistical implications of these and other FHE strategies. In their article, Xie-Bilenko et al. [20] focused on a problem—‘how does a customer hire a predictive model owned by a third party without compromising private information?’ Their goal was to make an assessment

using the model without jeopardizing the forecast’s precision or the data’s privacy. They’ll need to employ neural networks to achieve great accuracy, as they’ve been demonstrated to beat other learning models in a multitude of activities. To satisfy the privacy conditions, they need to use homomorphic encryption within the following protocol: the info owner encrypts the info and sends the ciphertexts to the third party to extract a forecast from a professional model. The model runs on these ciphertexts and sends back the encrypted forecast. During this protocol, not only does the info stay confidential, even the values expected are accessible only to the info owner. Using homomorphic encryption and modifications to the activation functions and training algorithms of neural networks, they proved that protocol is feasible and should be feasible. Their approach enables us to create stable cloud-based neural network prediction services while respecting users’ privacy. Implementing machine learning is a challenge when it involves financial, medical or other forms of confidential information, not only needs correct estimates but also special attention to preserving data privacy and protection. Legal and ethical requirements can prohibit the use of cloud-based machine learning solutions for such tasks. In their work, Bachrach-Dowlin et al. [21] proposed a way to transform trained neural networks into CryptoNets, which can be applied to encrypted data. This helps a cloud provider to transfer their data in an encrypted manner to a cloud server that hosts the network. The scheme guarantees that the data stays confidential because the cloud would not have access to the keys used to decrypt it. Additionally, they have shown that the cloud provider is capable of applying the neural network to the encrypted data to render encrypted predictions and even return them in encrypted form. These encrypted forecasts may be submitted back to the owner of the hidden key who can decode them. Therefore, the cloud provider does not receive any knowledge about the raw data nor about the forecast it produced. They have illustrated CryptoNets on the MNIST optical character recognition tasks. CryptoNets reach 99 percent precision and can make about 59000 predictions every hour on a single PC. Therefore, they make high throughput, precise, and private predictions. Phong-Aono et al. [22] developed a privacy-preserving deep learning scheme in which multiple learning participants conduct neural network-based deep learning over a shared dataset of both without directly exposing the respondents’ local data to a central server. They re-evaluated the earlier work by Shokri and Shmatikov (ACM CCS 2015) and point out that local data details could be potentially leaked to an honest-but-curious server. They then went on to fix the problem by building an improved system which prevents information leaks to the server and accuracy is held intact. This system is a gateway between deep learning and cryptography, where they have employed asynchronous stochastic gradient descent implemented on neural networks, in collaboration with additively homomorphic encryption.

When implementing machine learning to sensitive data, one has to find a compromise between precision, information protection, and computational-complexity. Recent research merged Homomorphic Encryption and neural networks to render assumptions while defending against data exposure. However, these approaches are constrained by the breadth and depth of neural networks that can be used and show high latency also for comparatively simple networks. In a research, Brutzkus-Elisha et al. [23] presented two alternatives that overcome these drawbacks. In the first approach, they proposed more than 10 times improvement in latency and allowed

inference on wider networks relative to prior attempts with the same degree of protection. The increased efficiency is accomplished by innovative approaches to reflect the data during the computation. In the second approach, they implemented the strategy of transfer learning to provide private estimation services utilizing deep networks with a latency of  $\sim 0.16$  seconds. They also showed the effectiveness of their approach on many computer-vision's tasks. Minelli et al. [24] proposed new FHE architectures inspired by machine learning applications, with a particular focus on the issue of assessing previously trained cognitive models on encrypted data. Firstly, they presented a novel FHE scheme that is suited to testing neural networks on encrypted inputs. Their scheme achieves complexity that is virtually independent of the number of layers in the network, while the performance of previously proposed schemes greatly relies on the topology of the network. Then, they proposed a new technique for achieving circuit privacy for FHE. This helped them to hide the computation that is conducted on the encrypted data, as is required to protect proprietary machine learning algorithms. Their mechanism involves very limited computing overhead while retaining the same protection parameters. Together, these findings reinforce the pillars of effective FHE for machine learning, and pave the way towards functional privacy-preserving deep learning. Finally, they proposed and introduced a protocol focused on homomorphic encryption for the problem of private information retrieval.

Reinforcement learning (RL) is a learning strategy that facilitates state-dependent learning by input from an environment and allows an action decision for optimizing a reward without previous awareness of the environment. If these RL techniques are used for data-centric services operating on cloud storage, major data protection problems arise because it is necessary to share privacy-related user data for RL-based services between the users and the cloud computing platform. Keeping this in mind, Suh et al. [25] designed a privacy preserving reinforcement learning model that utilizes the SARSA algorithm and FHE. They followed the SARSA algorithm since comparison is trivial in HE schemes and SARSA does not require any comparison operation unlike the Q learning algorithm. They proposed implementing the learning agent with blocking states, that restricts Q table to be updated during another ongoing update of the same state. This was done since encrypted computation consumes a great amount of time. Park-Kim et al. [26] considered utilizing homomorphic encryption (HE) scheme, which allows cloud storage systems to execute arithmetic operations without decrypting ciphertexts. Using the Homomorphic Encryption scheme, users are required to deliver only ciphertexts to the cloud storage network for using RL-based services. They suggested a privacy-preserving reinforcement learning (PPRL) system for the cloud computing network. The suggested architecture utilizes a cryptosystem focused on learning with errors (LWE) for fully homomorphic encryption (FHE). Performance measurement and assessment for the proposed PPRL platform was performed in a number of cloud computing-based intelligent service situations.

Due to massive progression on the topic of homomorphic schemes in application, Kim Laine and his team at Microsoft [27], introduced a simple encrypted arithmetic library in their paper that allows simple homomorphic operations and provides an interface for easy encryption/decryption over numerous well-established HE schemes like BFV and CKKS. The paper outlines the key functionality of SEAL 2.3.1, which

aims to include a high-level guide to utilizing homomorphic encryption for a large audience. The library is accessible from <http://sealcrypto.org>, which is approved under the MSRLicense Agreement. And in 2015, the first edition of the Simple Encrypted Arithmetic Library was published. SEAL was released with the specific purpose of providing a well-engineered and documented homomorphic encryption library, with no external dependencies, that would be simple to use both by experts and by non-experts with little or no cryptographic context.

## **2.2 Methodology Background**

### **2.2.1 Encryption**

Encryption is a strong type of security procedure that scrambles the output of any device, directory, or document in a way that it's impossible to decrypt without a decryption key. Businesses may ensure that only permitted users have access to private data by adopting encryption and exercising dependable encryption key protection. Even if misplaced, hacked, or obtained without permission, encrypted data is unreadable and effectively worthless without its key. In a research paper titled "Different Encryption Algorithms In Cloud" [28], explains the cryptography classification is mostly referred as: Symmetric and Asymmetric key encryption. In a symmetric key, the sender only sends one key which is called the secret key to the recipient. A single key is used for both encryption and decryption. This type of schemes requires both parties to exchange their secret key following some pre-defined protocol. Asymmetric schemes have two key components, a public key that is used to encrypt data and the other is a private key that is used during decryption. The asymmetric encryption schemes are also referred to as "public key encryption" techniques. For small mobile devices, the public key cryptography is not very efficient, as it is focused on mathematical functions and needs more computations [29]. Symmetric encryption algorithms are much quicker than asymmetric algorithms, as consumption needs less computing capacity. But it is robust in terms of security since the decryption key or the private key does not need to be sent to other party/parties.

### **2.2.2 Homomorphic encryption**

The Greek terms homos, which means "same," and morphe, which means "shape," are used to define homomorphism. Homomorphic encryption is a type of encryption that maintains data security while providing the necessary foundation for performing computation on it. It enables a third party to execute encrypted data processing without disclosing the data's contents. A homomorphic cryptography functions in the same way as other public encryption systems. It uses a public key to encrypt data. The unencrypted data is only accessible to the person who possesses the associated private key. It differs from other methods of encryption in that it employs an algebraic framework to allow you or others to do many calculations on the encrypted

data. In reality, when the given data is sampled as integer numbers with add and multiply as operational functions, most homomorphic encryption approaches perform better. This enables encrypted data to be edited and analyzed in the same way that plain data is without the need to decode it. To put it another way, HE allows to perform computation on encrypted data without knowing or gaining access to the decoded data's details. They have the ability to calculate and analyze encrypted data in order to generate an encrypted response. In contrast to other stable computation approaches, homomorphic encryption uses arithmetic circuits rather than booleans. Homomorphic encryption can be divided into three categories. The main distinction between them is the number and types of mathematical operations performed on their ciphertext. The following are the three types of homomorphic encryption:

- Partially Homomorphic Encryption
- Somewhat Homomorphic Encryption
- Fully Homomorphic Encryption

#### **2.2.2.1 Partially Homomorphic Encryption (PHE)**

Any circuit composed of a single form of gate, addition, or multiplication, but never both, can be evaluated using this method. It does not impose any limitations on the circuit's size or depth. This style is ideal for applications that only require the addition or multiplication of encrypted data. A PHE that allows an unbounded number of modular multiplications is the RSA cryptosystem.

#### **2.2.2.2 Somewhat Homomorphic Encryption (SWHE)**

This type of scheme can test circuits that contain both addition and multiplication gates, but the depth of arithmetic operation is limited. Leveled Homomorphic Encryption is a subset of SWHE that can evaluate circuits with variable depth. It requires the parameters to be set before encryption, so the parameters of one's selection must be tailored to the circuits they want to evaluate. SWHE can help with low-degree polynomial evaluations up to a point, but we occasionally need to evaluate operational sequences in greater depth.

#### **2.2.2.3 Fully Homomorphic Encryption (FHE)**

Will et al. mentioned in their article [30] that fully homomorphic encryption (FHE) is frequently referred to as the "holy grail" of cloud security. While many are aware of its potential, few are aware of how FHE operates and why, despite its promises, it is not yet a realistic solution. Fully homomorphic encryption (FHE), however still in its early phases of research, holds a lot of potential for balancing efficiency and



security by helping to keep data safe while permitting access. Fully homomorphic encryption seeks to allow anybody with exposure to the public encryption keys to execute functionalities on encrypted data. This approach has significant implications for improving cloud computing security. The disadvantage of using this form of encryption is that it sacrifices pace for versatility. Unfortunately, homomorphic encryption is currently too slow to be effective. It is currently in last place in the encryption competition. This is partly due to the higher computing overhead of homomorphic encryption compared to plaintext operations.

Fully Homomorphic Encryption (FHE) schemes can test circuits that include addition and multiplication gates. Still, unlike Somewhat Homomorphic Encryption (SWHE), they have an infinite circuit depth, making them ideal for deep learning applications. Even though many FHE systems have been proposed over the last decade, putting them into practice has proven to be difficult. FHE is, in reality, now built on top of SWHE. Thanks to Craig Gentry, who demonstrated in his paper how to make FHE from SWHE using a technique he called bootstrapping. The mentioned system is a lattice-based cryptosystem which was proposed and developed by Craig Gentry [2] in 2009. FHE is considered as far more powerful and a great way to secure the outsourced data in an efficient manner [31]. The proposed scheme given by Gentry has three important components:

1. A somewhat homomorphic encryption scheme (SWHES)
2. A bootstrappable encryption scheme (BES)
3. A combination of above two components

This scheme has the ability to conduct the homomorphic computation on low grade polynomials.

An encryption scheme includes the following 3 algorithms:

---

- **KeyGen (...security\_parameters) → keys:**

---

The Key Generation algorithm takes as input one or more security parameters. The scheme's key(s) are then output.

---

- **Encrypt (plaintext, key, randomness) → ciphertext:**

---

A plaintext, a key, and some randomness (encryption must be probabilistic to be “safe”) are all inputs to the Encryption method. It then outputs the ciphertext that corresponds (i.e., encrypted data).

---

- **Decrypt (ciphertext, key) → plaintext:**

---

The Decryption algorithm has two inputs: a ciphertext and a key. It returns the plaintext equivalent.

A simple overview of a few well-established FHE schemes is given below:

1. **BGV**: The Brakerski-Gentry-Vaikuntanathan (BGV) [4] technique is one of the most efficient homomorphic encryption systems because of allowing execution of the same operations on many ciphertexts at the same time. In Spite of it being efficient it is very difficult to use. This strategy is a completely homomorphic encryption strategy that may be used both with a LWE and an RLWE instance, however the RLWE instance has outperformed its counterpart. In their work they presented “leveled FHE without bootstrapping” construction in modular steps. First, they described a plain GLWE-based encryption scheme with no homomorphic operations. Next, they augmented the plain scheme with variants of the “relinearization” and “dimension reduction” techniques [32]. Then they laid out full-fledged construction of FHE without bootstrapping. The name of this scheme usually comes from the last names of the authors who created it. As in, “BGV” comes from the authors Brakerski, Gentry, and Vaikuntanathan.
2. **BFV**: Fan and Vercauteren [33] adapted Brakerski’s [34] scheme from the LWE setting to the RLWE setting in 2012. They employ relinearization in the same way that does, but their version is more efficient. They have utilized modulus switching to make the bootstrapping method easier.
3. **CKKS**: The Cheon-Kim-Kim-Song (CKKS) [5] scheme is a homomorphic encryption (HE) scheme which supports approximate computations of real (complex) numbers. It includes a new rescaling procedure for managing the magnitude of plaintext, as well as approximate addition and multiplication of encrypted messages. This procedure reduces the modulus of a ciphertext, resulting in rounding of the plaintext.

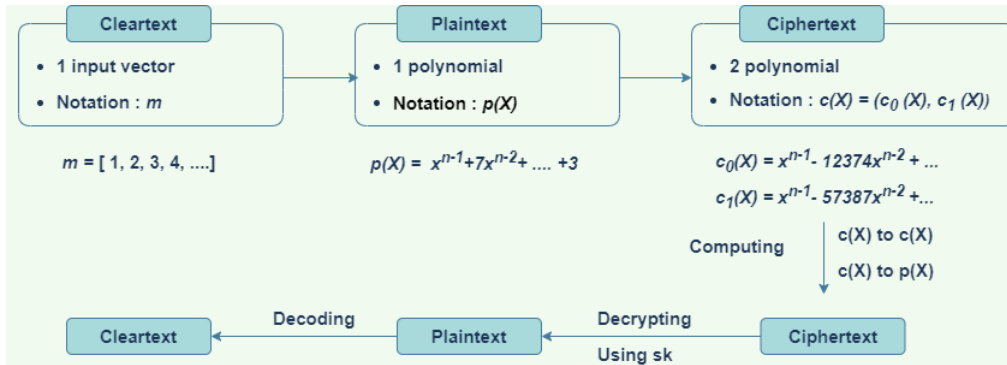


Figure 2.1: High Level CKKS Procedure

The high level CKKS procedure is depicted in fig: 2.1. We can see that a message  $m$ , which is a vector of values on which we want to compute, is first encoded into a plaintext polynomial  $p(X)$  and then encrypted with a public key. When compared to typical vector computations, CKKS uses polynomials because they offer a good balance of security and efficiency. After the message has been encrypted into  $c$ , which is a set of polynomials, CKKS offers a number of operations on it, including addition, multiplication, and rotation. While addition is relatively simple, multiplication has the distinct property of greatly

increasing the amount of noise contained in the ciphertext, as a result, only a limited number of multiplications are permitted. Permutations on the slots of a particular ciphertext are known as rotations. If we represent a function by  $f$  as a collection of homomorphic operations, then decrypting  $c' = f(c)$  with the secret key yields  $p' = f(p)$ . As a result, once we've decoded it, we'll receive  $m = f(m)$ . This offers us a high-level understanding of how CKKS works. Moreover, authors tried to present a method for efficient approximate computation on HE. The main idea was to treat an encryption noise as part of error occurring during approximate computations. They introduced a new batching technique for packing much information in a single ciphertext, so they could achieve practical performance advantage by parallelism. Another benefit of their scheme is the rescaling procedure, which enables us to preserve the precision of the message after approximate computation. Furthermore, it reduces the size of ciphertext significantly so the scheme can be a reasonable solution for computation over large integers.

While working with untrustworthy environments, such as public clouds or external parties, data remains protected and private if HE technique is employed. The data is kept encrypted at all times, reducing the chances of important information being compromised. The balance between data usefulness and data privacy is no longer an issue. To protect the privacy of data, there is no need to disguise or remove any elements. Without jeopardizing privacy, any feature can be used in an analysis. Quantum attacks are resistant to fully homomorphic encryption systems. Despite fully homomorphic encryption being useful in many scenarios there are some limitations of this encryption scheme. Fully homomorphic encryption is still commercially unfeasible for computationally intensive applications due to slow calculation performance or accuracy issues. The research community agrees that completely homomorphic encryption research has a long way to go, but it is already helpful in conjunction with other privacyenhancing technologies such as secure multiparty computation. However, in its current state, completely homomorphic encryption can handle use cases that aren't computationally costly, such as prediction using a pre-trained model. FHE has two recognized drawbacks at the moment. Firstly, the lack of multi-user support. Assume that a lot of people on the same system decide to put their personal data secure from the operator. An option is for the operator to have a unique database for every user, which would be encrypted with the person's public key. If the database was large and there were a large number of people, this would eventually become impossible. The second drawback is, large computational overhead. Currently, all fully homomorphic encryption systems have a significant computational expense. While this cost is exponential in size, it looks substantially constant, which considerably increases runtime and renders homomorphic calculation of complicated systems impossible.

### 2.2.3 Machine Learning

Machine learning is indeed a crucial part of the rapidly expanding subject of data science. Models are designed to generate classifications or forecasts using inferential

statistics, revealing critical insight in data analysis applications. Following that, these findings drive verdicts within systems and enterprises, with the goal of influencing key economic benchmarks. The machine learning capabilities depend on the dataset in studies. Machine learning doesn't hamper productivity or bothers the users. It just acts as a data centric approach which learns from more and more scenarios and evolves. There are mainly 3 types of machine learning:

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

#### **2.2.3.1 Supervised Learning**

Data on both input and output are provided in supervised learning. The algorithm uses the information to predict and compares the forecast to the expected result when using supervised learning. If wrong, the algorithm will somehow change to make better future predictions. One needs to train the machine in supervised learning with well-labeled data. It means that the correct answer to specific data has already been tagged. The learning in the presence of a supervisor or professor can be compared. An algorithm for supervised learning is derived from labeled training data and helps to predict results for unpredictable data. A learning model requires time and the technical expertise of a highly qualified team of data scientists to develop, scale, deploy and also accurately monitor the performance. In addition, data scientists need to reconstruct models to ensure that the data is valid until their data changes.

#### **2.2.3.2 Unsupervised Learning**

Unsupervised learning is a machine learning methodology in which the model is not monitored. Rather, we need to provide the model the ability to find information by itself. It primarily encompasses information that has not been labeled. Unsupervised algorithms automatically learn patterns of data or groupings. Unsupervised learning is excellent when we need to find patterns in a set of uncategorized data. Unsupervised models can also be classified as clustering or association tools.

The two subsets of ML, Supervised and Unsupervised, are heavily data-centric. One requires the labeling of training inputs while the other performs well on large chunks of data. Unlike these, the third subset, Reinforcement Learning, relies on an environment rather than a dataset. RL agents are able to optimize themselves by taking feedbacks from the environment assigned to them. Thus, real-time learning can be implemented using RL. So, we have discussed about Reinforcement Learning thoroughly in the next section.

## 2.2.4 Reinforcement Learning

Reinforcement learning is an approach for machine learning, which focuses on how artificial entities might conduct themselves in a particular situation. Reinforcement Learning is a deep learning methodology which enables a percentage of total reward to be optimized. This learning approach teaches us how to achieve a complex goal or optimize a particular dimension over a long period. In his blog, Huang [35] describes that to assess their previous conduct, an RL agent earns a late compensation in the future stage. A typical RL configuration comprises only two components: the agent as well as the environment. The environment refers to the item on which the agent is functioning, whereas the agent symbolizes the RL algorithm. The environment communicates a state to the agent, which the agent responds to depending on the knowledge it has. The world then sends the agent a pair of following state and incentive messages. The agent will update its data using the prize given by the environment to assess its previous activity. The loop will continue until the environment communicates a terminal status which will finish the episode. Figure 2.2 depicts the pattern that most RL algorithms follow.

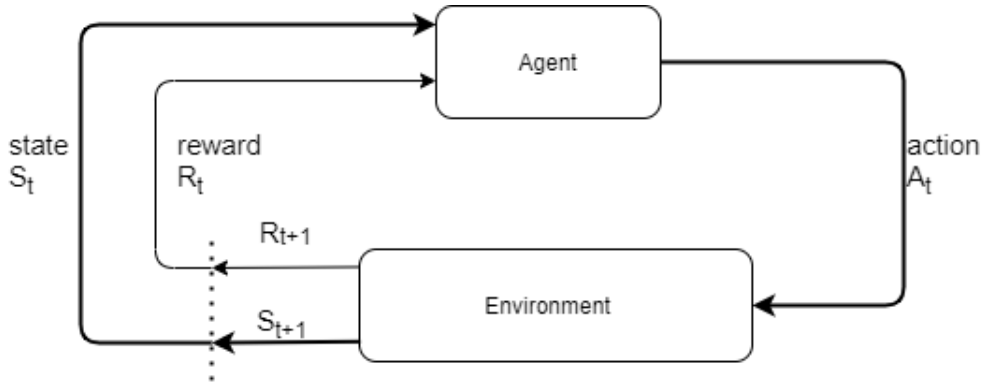


Figure 2.2: Simplified Reinforcement Learning Procedure

Let us discuss some of the terminology used in the field of RL.

1. **Agent:** The learner and decision-maker are known as the agent.
2. **Environment (e):** The Agent's environment is where it discovers and determines what steps to take.
3. **Action (A):** A selection of acts that the agent will carry out.
4. **State (S):** The Agent's current state in the environment is its state.
5. **Reward (R):** The prize is an immediate response from the environment that evaluates the previous behavior.
6. **Policy( $\pi$ ):** The Agent's policy is how it decides what to do next based on the current situation.
7. **Value Function (V):** The value function is used to convert states to numbers. The longterm benefit earned by starting from a certain state and implementing a given policy is represented by a state's value.

8. **Q-value (Q):** Q-value is close to value, except that it needs an additional parameter, the current operation,  $a$ . The phrase "Q policy" refers to both the long-term recovery of the current condition and policy execution.

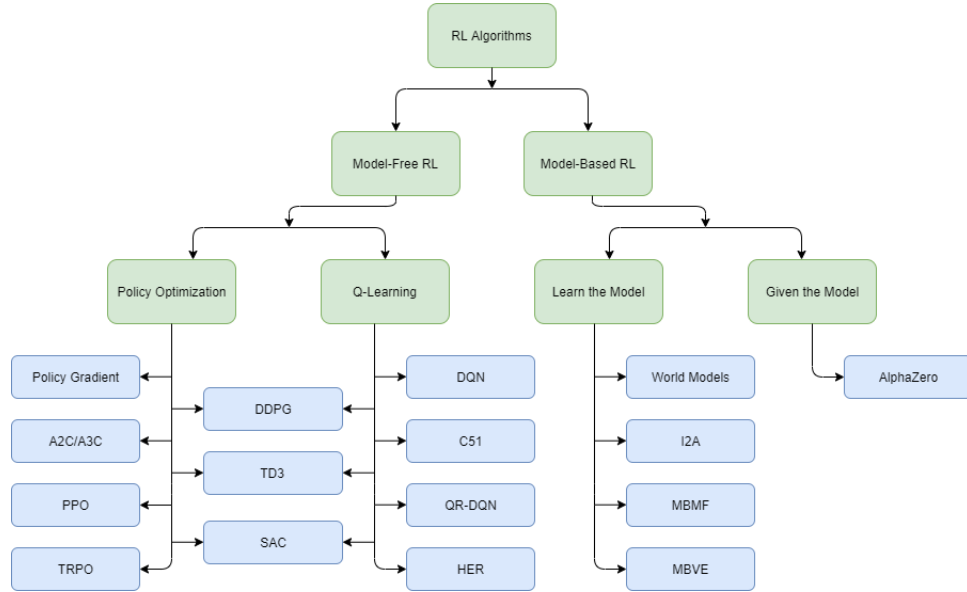


Figure 2.3: RL Algorithm Tree

A Reinforcement Learning algorithm is implemented in three ways:

1. **Model-Based:** In this training process, we need to build a virtual model for every environment. In this particular environment, the agent learns how to work. ModelBased RL draws on previous experience in developing an inner transition model as well as current environmental conditions. Suitable action in this world-model is then selected through searching or planning. On the other hand, model-free RL utilizes the experience to master one or two simple quantities (state/action values or policies) directly but without evaluating or using a world-model to achieve the same good conduct. A State's worth, given a policy, should be measured by its future utility, starting with the state. Model-free approaches are statistically less successful than model-based approaches because information from the environment is mixed with past, and presumably wrong, estimations or ideas about state values instead of being utilized openly.
2. **Policy-Based:** Agent tries to develop a policy in a policy-based RL system that guides it to achieve the most reward in the future by performing actions in each state. The contrast between Off-policy and On-policy techniques is that with the former, you don't have to adhere to any precise policy; in fact, your agent might act arbitrarily, and off-policy techniques can still identify the best policy. On-policy procedures, in contrast, are reliant on the policy in question. As in the context of off-policy QLearning, it will determine the best strategy regardless of the policy utilized throughout exploration.

3. **Value-Based:** In a value-based reinforcement learning strategy, users should maximize the value function  $V$ . In this way, the agent expects the current states to return maximum value for adopting a certain policy for a long time.

The following are some of the most critical features of reinforcement learning.

- There is no supervisor, merely a number or an incentive signal.
- Making decisions in a certain order.
- In reinforcement complications, time is crucial.
- Feedback is never given immediately and is always delayed.
- The actions of the agent determine the following data they collect.

## Reinforcement learning algorithms

In reinforcement learning, there are three well known algorithms

- Q Learning
- SARSA
- Deep Q Learning

### 2.2.4.1 Q Learning

Q learning is a value-based reinforcement learning agent for guiding an agent's decision. The action-value function Q-learning  $Q(s, a)$ : how well in a particular state to pull an action. An action a given state assigns a scalar value. In figure: 2.4, the algorithm is well represented. The optimal Q-value, denoted as  $Q^*$ , is as follows:

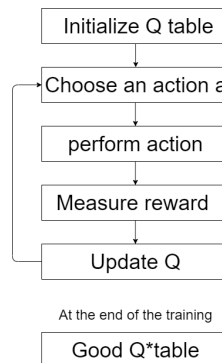


Figure 2.4: Q learning Flowchart

$$Q^*(s, a) = E_{s'} [r + \lambda a_{\max} Q^*(s', a') \mid s, a]$$

The target is to increase the Q-value as much as possible. Two value update approaches are nearly related to Q-learning before digging into the approach to maximize Q-value.

## Policy Iteration

The feedback circle between policy evaluation and policy improvement is known as policy iteration. The greedy policy produced from one of the most recent policy

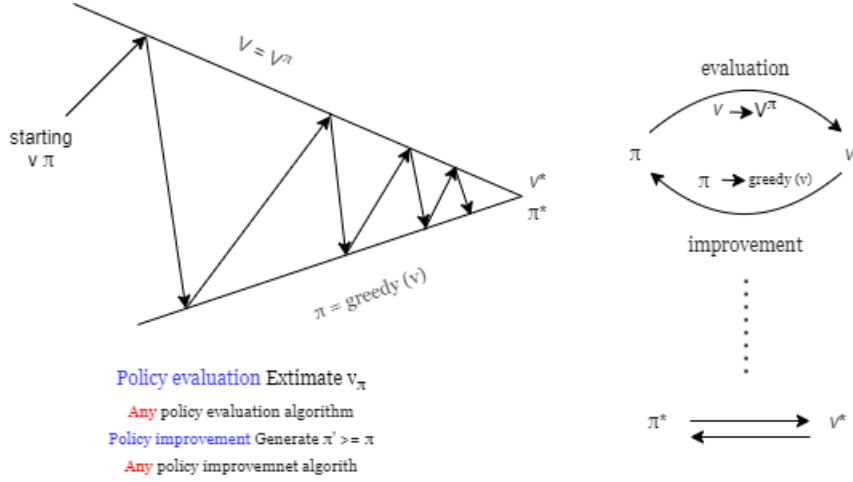


Figure 2.5: Simple Presentation of Policy Iteration

upgrades is used to compute the value function  $V$  for policy assessment. Policy enhancement, on the other hand, updates the policy to include the best route  $V$  for each state. The Bellman Equation serves as the foundation for the updated equations. It iterates until convergence is reached.

## Value Iteration

There is only one component in Value Iteration. It modifies the value parameter  $V$  using the Ideal Bellman Formula.

$$\begin{aligned} v_*(s) &= a_{\max} E [R_{t+1} + v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= a_{\max} \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_*(s')] \end{aligned}$$

### 2.2.4.2 State-Action-Reward-State-Action (SARSA)

SARSA is quite similar to Q-learning. Q-learning is a value-based algorithm, whereas SARSA is a policy-based algorithm. It means that SARSA, instead of the greedy policy, learns the Qvalue on the basis of current policy's operation. Zhao et al., [36] in their research, have mentioned that SARSA, a type of on-policy reinforcement learning approach, is combined with deep learning to tackle the video game control difficulties. Moreover, they calculate the stateaction value with a deep convolutional neural network and modify it through SARSA learning. In addition, experience replay is used to adapt the learning method to scalable machine learning challenges.



### 2.2.4.3 Deep Reinforcement Learning

While Q-learning seems to be a suitable technique, it has one major flaw: it isn't particularly generic. Q-learning is similar to dynamic programming in that it is a double array of updated integers. This shows that it has no idea what actions to undertake in states which the Q-learning Agent has not seen before. An optimal policy or policy function can be simulated with a neural network which allows neural networks to connect values and status pairs to Q values. We can train a neural network on samples from the state or action domain to learn to predict how important those are compared to our goal in reinforcement learning, instead of using a lookup table to record, map and update all possible states and their values, which is impractical for rather significant problems. An example of deep Reinforcement learning in practice is the Deep Q Network (DQN), first developed by DeepMind [37]. For estimating the Q-value function, DQN uses one or multiple neural networks. The Network input is current, and the Q-value for each action is the corresponding output.

There are also two other techniques for DQN training:

1. **Experience Replay:** In a typical RL scenario, training samples are strongly interrelated and lower data-efficient, resulting in more substantial network convergence. An experience replay is a way of solving the sample distribution problem. In principle, sample transitions are saved, and the knowledge is then randomly chosen from the “transition pool.”
2. **Separate Target Network:** The architecture of the Q network target is identical to that of the value calculator. Each C step is reset to another network. The fluctuation is, therefore, less severe and, therefore, more stable.

### 2.2.5 Neural Network

Neural networks are a set of algorithms that recognize patterns and are based on the human brain. They use a kind of machine perception to perceive sensory data, marking or clustering raw data. All real-world data, including images, sound, text, and time series, must be transformed into numerical patterns stored in vectors. Neural networks assist us in clustering and classifying data. “Stacked neural networks” or networks of many layers, are referred to as “deep learning”. Nodes are the unit elements of any given layer. A node is simply a location where computation occurs, loosely modeled after a neuron in the human brain, which fires when it receives enough stimuli. A node combines data input with a set of coefficients, or weights, that either amplify or dampen the information, thus assigning importance to inputs concerning the task the algorithm is attempting to learn. The sum of these input-weight products is then passed through a node's so-called activation function, determining whether and how far the signal can advance through the network to influence the ultimate result, such as a classification act. The neuron has been “activated” if the signals pass through. A node layer collects neuron-like switches that turn on and off as data passes through the network. Starting with an initial input layer that receives the data, each layer's output is also the subsequent layer's input.

Considering the structure of NN, deep learning networks differ from one hidden level neural networks in the depth or number of node layers through which data should be transferred in a multi-stage pattern matching technique. External neural networks with only a single input and output layer or at most one hidden unit in between were the earliest perceptrons. Learning with three or more layers is referred to as deep learning. It's a well-defined word that refers to several hidden layers. Deep-learning networks can process heavy, high-dimensional data sets with billions of variables going through nonlinear functions, thereby allowing them to take highdimensional sets of data essentially. Digesting and grouping the world's unfiltered, unlabeled media, discovering patterns and abnormalities in data that no human has ever arranged in a relational database or given a name to, is among the issues that deep learning thrives at. Unlike other conventional machine-learning algorithms, deep-learning networks extract features automatically without the need for human interaction. When learning features from unmarked data every node level in a deep system tries to decrease the gap between the network's estimations and the probability dispersion of the input data by continually recreating the input from which it pulls its observations. In a logistical or SoftMax classifier, the output layer of deep-learning networks provides a probability to a certain result or mark. It is referred to as prediction, but only in the widest perspective.

The way humans communicate with the environment is changing with Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and Artificial Neural Networks (ANN) as well as other forms of deep learning neural networks. Such neural networks are indeed the heart of the deep learning revolution, driving applications including crewless aircraft, selfdriving automobiles and language recognition. One of the prevalent types of deep neural, CNN is briefly described in the next section since it plays a significant role in our work.

## 2.2.6 Convolution Neural Network

The name CNN comes from the inclusion of one or several convolution layers in a deep learning architecture. CNN building blocks are the filters, known also as the kernels. Each convolution layer consists of several kernels. The kernel amount usually depends on the diversity and complexity of input data and output-space. Each kernel builds a certain feature map from the input data during optimization. These feature maps only extract the data that is relevant for its successor nodes. The convolution procedure is used to eliminate irrelevant features utilizing seed from the input. Although convolutional neural networks have been designed to handle problems with visual data, sequence inputs also function well. The advantages of CNN architecture is given below:

- CNN captures the spatial features of an image. The layout and relationships of pixels in a picture are called spatial characteristics. They help us to recognize an object precisely, and its location and connection with other items.
- Without identifying them, CNN recognizes the filters instantly. Such filters help to identify from input data the main essential and suitable attributes.

Comparison among different types of Neural Networks			
	ANN	RNN	CNN
Data	Tabular data	sequence data (Time series, Text, Audio)	Image Data
Recurrent connections	No	Yes	No
Parameter sharing	No	Yes	Yes
Spatial relationship	No	No	Yes
Vanishing & Exploding Gradients	Yes	Yes	Yes

Table 2.1: Comparison Among Different Types of Neural Networks

- CNN also uses the definition of parameter sharing. A feature map is created by applying a single filter to various sections of an input.

The basic difference between CNN and other types of NN architectures are presented in table 2.1.

# Chapter 3

## Proposed Method

We aim to implement a model that will allow users to employ cloud-hosted real-time deep reinforcement learning agents without jeopardizing sensitive information using homomorphic encryption techniques as its backbone. This chapter focuses on the groundwork on which we built our model on. Section 3.1 describes the overall system architecture and illuminates the flow of data. The architecture is designed keeping preservation of privacy as its core focus while also keeping network overhead in mind. Then we further elaborate on the algorithms chosen with which the model is implemented along with the reasons behind our choice in Section 3.2. The algorithms are used to implement distinct components of the model while the system architecture represents the model as a whole. In Section 3.3, the details about the libraries used are provided, also mentioning what purpose each of them accomplishes and how they work in our model.

### 3.1 System Architecture

We have come up with not only one but two unique architectures. While both fulfills our goal, each of them provides their own functionalities to be incorporated in different scenarios. The first one follows the principle of learning on plaintext and evaluating on ciphertext in real-time (described in fig: 3.1) and the second one follows encrypted real-time learning (described in fig: 3.2) which operates purely on encrypted data, image objects in our case. From here on we will refer to the first architecture using Plain-Encrypted architecture and the second one using the term Pure Encrypted. Learning agents suitable to perform in both architectures will be discussed in Chapter 4.

The flow of data for each of these systems along with necessary details is provided below

**Plain-Encrypted:** According to this model, the cloud platform runs a simulated version of users' environment or an identical environment on their own. A plaintext agent will utilize the hosted environment for learning while a single or multiple en-

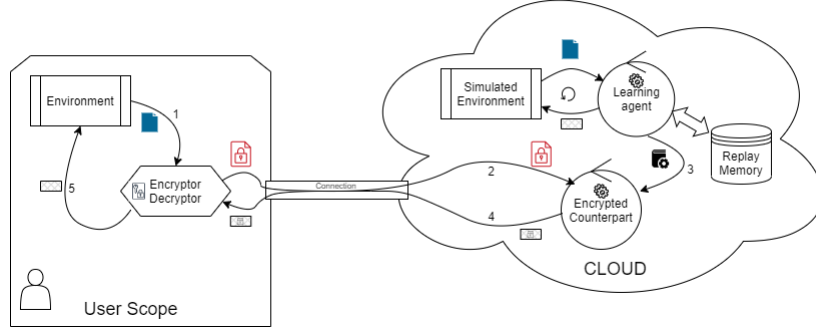


Figure 3.1: Plain-Encrypted Architecture

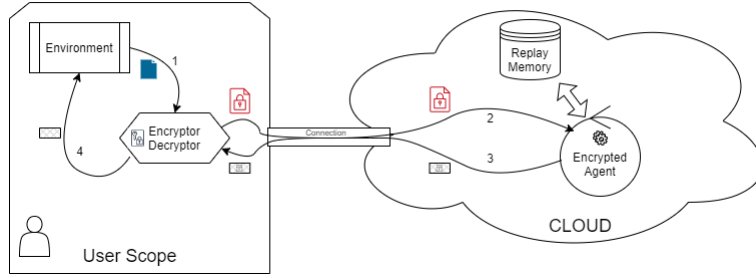


Figure 3.2: Pure Encrypted Architecture

encrypted counterparts of that agent will perform evaluation on ciphertexts sent from the users' domain. The encrypted counterpart performs the same set of arithmetic as the plain one but it is tailored for encrypted operations. Before an evaluation, the encrypted counterpart will fetch the most updated parameters of the plain learning agent to use as its own and deliver encrypted outputs. If the cloud platform is able to run an identical environment, no feedback from the user's system will be necessary. The dataflow for this architecture is as follows

- The user sends plain state data to their own encryption-decryption module to convert into ciphertext using their public key.
- The encrypted state is sent to the cloud platform and the cloud queues the data to be used in the encrypted evaluation.
- The encrypted counterpart fetches the most recent parameters of the plain learning agent and updates its own.
- The generated encrypted output using updated parameters is sent back to users' domain where it is fed back to the encryption decryption module.
- The module decrypts the output using users secret key, extracts the most suitable action then feeds it to the environment.

This architecture is befitting for environments where the goal is similar across all users. For example- traffic light automation, control for industrial robots, commercial chat-bots etc. This feature allows this architecture to be a multi-user system

since only one instance of learning environment is necessary to satisfy the needs of any number of customers. It is also efficient in a sense that the plain learning agent can keep on optimizing even when none of the users are requiring the cloud platform's services.

**Pure Encrypted:** This model is pretty simple and straightforward in terms of components in comparison to the previous one. It consists of only one agent that is responsible for both learning and evaluation. However, the network overhead is a bit larger since it relies on the reward given from the user's environment to calculate expected value in order to optimize itself. This models' dataflow is given-

1. The user sends plain state data to their own encryption-decryption module to convert into ciphertext using their public key.
2. The encrypted state is sent to the cloud platform and the cloud queues the data to be used in the encrypted learning and evaluation.
3. Encrypted output is generated according to the agent's policy and is sent back to users' domain where it is fed back to the encryption-decryption module.
4. The module decrypts the output using users secret key, extracts the most suitable action then feeds it to the environment.

Additionally, the necessary values for optimization are also sent to the cloud platform but it can be done while sending the inputs for the next evaluation, hence this step is absent in the figure. The Pure-Encrypted architecture is appropriate for environments where the user's preference plays a significant role in determining preferred actions. For example recommending news articles, targeted ads, personal electronic assistant, personalized chatbots. These environments require learning from scratch as simulated training or training in advance will not bring equal value across all users.

## 3.2 Used Algorithms

There are a handful of algorithms from which we had to choose from that would work as the base of our models when it comes to implementing them. Our general goal is to design a model that would combine real-time learning abilities that come with reinforcement learning and privacy preserving computation using homomorphic encryption. Both of these concepts come with a few well-established algorithms. A methodical analysis of these algorithms was imperative to determine the specific algorithms which best suit the problem at hand. This section describes our chosen algorithms and also illustrates the criteria on which these choices were made.

### 3.2.1 Deep Q Learning

As we discussed earlier in Section 2.2.5, SARSA and Q-learning both are table or function-oriented algorithms that work well with a finite set of states and actions. But when it comes to continuous state space, these algorithms struggle in terms of memory or function complexity. Deep reinforcement learning eliminates this problem by using neural networks to work as a value function. Since we wish to work with visual inputs i.e., images as states, our state space is also a continuous one, therefore, SARSA or Q-learning would not be an optimal choice. Deep Q learning on the other hand, promises to deliver what we ask for. Thus, we have decided to employ a deep Q network to act as our learning agent. The concern for inherent error was eliminated with a plaintext version of the agent as the results were satisfactory.

### 3.2.2 CNN

Since deep Q network utilizes neural networks to compute action values, the appropriate type of neural network had to be determined. Different types of neural networks excel at different types of inputs. Among all the types, ANN and CNN are known and widely used for processing images and fits our objective. However, since we wish to work with homomorphic computation which is heavy and time consuming, limiting the number of computations reduces the time taken for each step by a great margin and is one of our top priorities to make the model more practical. The convolution property of CNN helps to reduce the number of network nodes without compromising effectiveness. Therefore, Convolutional Neural Network was the go-to network to work as the engine of our deep Q learning agent. ANN on the other hand, would be too heavy computationally in this context, threatening the practicality of our designed models by making them too time consuming.

### 3.2.3 CKKS Scheme

The CKKS scheme, formerly known as HEAAN, designed by Cheon et al. [5] is one of the most recent members of Homomorphic Cryptography. Unlike previous encryption schemes like BGV and BFV, CKKS is an approximate homomorphic encryption scheme, meaning it performs approximate arithmetic on encrypted vectors making it much faster than its predecessors yet less precise. The precision of decrypted vectors depends heavily on the encryption parameters (poly-modulus degree and coefficient modulus). With appropriate parameters, the loss of precision becomes negligible. This scheme also allows homomorphic operations on floating point numbers using a rescaling factor unlike many other schemes which only work on integers. The functionality for vector rotation enables this scheme to be incorporated in many applications including CNN. Since we opted to use CNN as our learning agent, CKKS proves to be the best candidate for this scenario. It also provides flexibility in input pre-processing for supporting floating point numbers, along with faster operations which aids our cause. The version of CKKS that we

adopted for our model is levelled-fullyhomomorphic as we did not need the bootstrapping mechanism. Simple relinearization and rescaling operations proved to be enough for our implementation. However, a bootstrapping function is available for CKKS [6] and can be included if required but time constraints should be taken into consideration as CKKS bootstrapping also requires heavy computation.

### 3.3 Used Libraries

Here we will describe the tools and libraries that we utilized for implementing the overall system. The function of each tool in our system is also discussed in their relative sections.

#### 3.3.1 Microsoft SEAL

Several schemes have been developed for homomorphic encryption, but those were hard for normal software developers to use because their usage needed understanding of critical mathematics. Then there is Microsoft seal coming up with a convenient and very simple API which has state-of-the-art execution. Cloud operators need decrypted access to customer data to do direct computation in any kind of traditional cloud storage. Whereas, in Microsoft SEAL, the cloud operators don't need any unencrypted access to customer data which maintains the integrity of customer data and it stays secure and not exposed to anyone but the customer because all the computations are done in encrypted data.

In our research, all the cryptographical procedures are handled by Microsoft SEAL in the backend. The library itself is built on C++ and has a built-in wrapper for C# representations. SEAL takes encryption parameters defined by the user and generates necessary keys to conduct homomorphic operations based on the chosen HE scheme. Currently the library supports BFV and CKKS schemes.

#### 3.3.2 TenSEAL

TenSEAL is an open-source library that can be readily incorporated into common machine learning frameworks for privacy-preserving machine learning utilizing homomorphic encryption. It takes care of all the difficulties that come with implementing tensor operations on encrypted files. TenSEAL is based on Microsoft SEAL's implementation of numerous HE schemes. Clients can use one of the supported frontend languages (C++ or Python) to work with plain or encrypted tensors. The encryption context, simple tensors, and encrypted tensors are the three main components of the core API.

In our work, TenSEAL serves as an interface between the python compiler and C++ compiler on which the SEAL library runs on. It provides the compatibility



we need to work with HE schemes that are mostly implemented in cpp with robust ML frameworks present in python. The TenSEAL library also comes with a simple convolution procedure on HE ciphertexts that proved to be helpful during this research.

### 3.3.3 PyTorch

PyTorch is one of the most popular machine learning modules which is based on the Torch library. PyTorch has a python interface with a C++ backend. It is able to utilize a system's GPU along with CPU to accelerate the heavy computations needed for ML operations. PyTorch has some major features like having an easy interface, auto gradients, and computational graphs. It offers an API which is easy to use and runs on python. This library is pythonic in nature, and it works well with the Python data science stack. As a result, it will take advantage of all of the python environment's resources and features. PyTorch also offers a few dynamic computational graphs. The three levels of abstraction in PyTorch are well-known, which are Tensor, variable and module. The advantages of PyTorch makes it a very reliable and useful platform for deep learning. Debugging and understanding the code is very easy and it also includes many loss functions. Many well-defined and widely used NN layer functions are present in PyTorch.

We chose to use PyTorch because it provides sufficient room for customization unlike many deep learning frameworks. As we are working with irregular data types i.e. ciphertexts, custom layer design was one of the core requirements.

### 3.3.4 Torchvision

Torchvision is a PyTorch library that helps you easily use pre-configured models in computer vision applications. This is especially useful when inferencing with a simple pre-trained model and finding the similar target objects, which were present in the pre-training dataset. However, when using a dataset which is customized, the model frequently needs to be retrained and modified for such more advanced use cases.

While it is true that what we are working with subtly falls within the scope of computer vision, our objective is much simpler in that context. Hence, we did not require the advanced functionalities of Torchvision. It is used only to perform some preprocessing steps on the standard input images.

### 3.3.5 Protocol Buffer

Protocol Buffers is a cross-platform library for serializing structured data that is free and open source. It can be used to create programs that interact with one another over a network or to store data. A description interface language describes

the structure of certain data, and the source code to generate or parse a byte stream that represents the structured data of that description. Protocol Buffers were created with effectiveness and convenience in mind. It was created with the goal of being smaller and faster than XML. Protocol Buffers are widely utilized at Google for preserving and sharing various kinds of organized data.

Protocol Buffer is able to serialize the rather large and complicated structure of the HE ciphertexts and encryption contexts (many different keys altogether) produced by the SEAL library. In our model, the library is used as a gateway by successfully transforming encrypted objects into raw bytes that are transferable over a network.

### **3.3.6 Gym**

Gym is simply a set of tools for testing and evaluating reinforcement learning algorithms. It makes no preconceptions about the architecture of any agent and works with a number computing framework. An example is TensorFlow or Theano. The gym collection contains test scenarios that may be used to put reinforcement learning methods to the test. Because these contexts share an interface, generic algorithms may be written.

We have employed the Gym module to test the designed agents. Gym also provides a GUI for user observation that we use to extract visual inputs for our agents. The comparisons are done by running the equivalent regular learning agents on the same testing environment as our encrypted agents.

# Chapter 4

## Implementation and Experiment

Previously, we have described the groundwork for our model. This section illustrates implementing the overall model and the agents working on it. To demonstrate our model, we opted to use a simple environment which fits our system criteria. The cart-pole environment provided by the gym library is both simple and provides a visual window for human observation which we can use to extract visual states through screenshots. The screenshot images are modified for efficiency's sake and then encrypted with the CKKS scheme using proper encryption parameters. These encrypted images act as states for our learning agents. As we have described two architectures in the previous section, Plain-Encrypted and Pure Encrypted, two distinct agents were designed, each suitable for its own architecture. Although the Plain-Encrypted architecture defines two seemingly separate agents, plain-learning agent and its encrypted counterpart, both are in fact, components of a single one which utilizes both components to achieve a single goal.

### 4.1 Environment Description

The classic control problem of pole-balancing represented by the name Cart-Pole environment by the gym library was chosen because of its simplicity as we wish to only observe the learning capabilities of our agent. Experiments with different parameters had to be done in order to find the optimal configuration. A complex environment with a complex action sequence would only make the agent too time consuming to experiment with. Here are the details about the environment-

- **Description:** An unactuated joint connects a pole to a cart that runs along a frictionless track. A force of +1 or -1 is applied to the cart to regulate the machine. The idea is to keep the pendulum from falling over. The pole appears upright and the cart appears in the middle of the screen at the start of each episode. When the pole is more than 15 degrees from vertical or the cart goes more than 2.4 units away from the center, the episode terminates.
- **State:** The library provides the cart's position and angle of the pole as numer-

ical state values. But in our implementation, we are going to use the screenshots taken from the GUI. A sample screenshot is shown in fig: 4.1.

- **Action:** A force of +1 or -1 on the cart meaning the cart can only go leftward or rightward. Hence, the length of our actionspace is 2.
- **Reward:** At each time step the virtual environment provides a reward of 1 if the pole remains upright, otherwise 0.
- **Goal:** Maximize the total reward collected meaning keep the pole upright for as long as possible while staying inside the defined range.

## 4.2 Data Preprocessing

We perform some necessary preprocessing on these images to make them manageable for our model without trading off most of the information needed to understand the current environment state. The images are modified to reduce feature size leading to a minimum number of computations over a single state. The standard unadulterated image size is  $3 \times 400 \times 600$ , meaning the feature size is 720000, which is a lot.

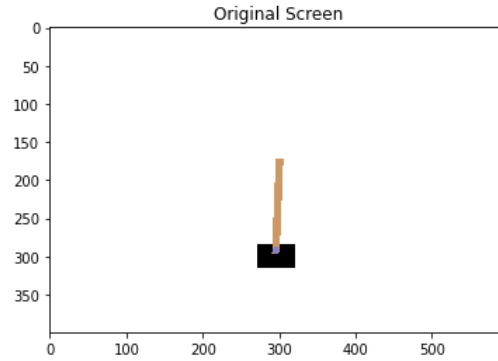


Figure 4.1: Sample Extracted Raw Image from GUI

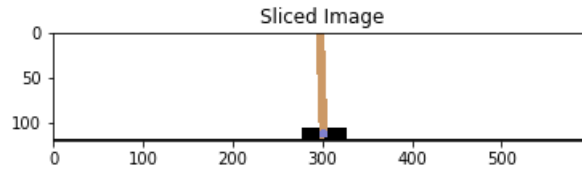


Figure 4.2: Sample Sliced Image After First Stage of Preprocessing

From fig: 4.1, we can see the actual cart and pole objects consist of a very small portion of the entire image state. Since the cart can only move left or right, a lot of the upper and lower area of the image is redundant as these regions remain unchanged regardless of the environment's state. So, we omit these regions out of the image through slicing (fig: 4.2), reducing the feature size to  $3 \times 120 \times 600 = 216000$ . After doing so, we are still left with a relatively large feature size, so we zoom in on our object of interest by cutting off excess regions from the carts left

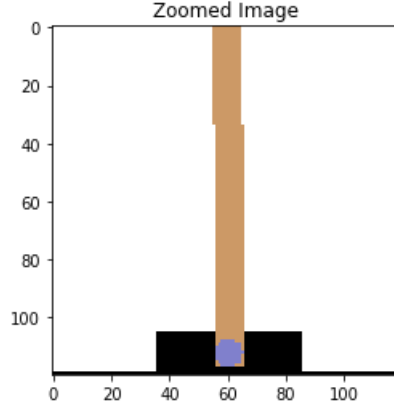


Figure 4.3: Sample Zoomed Image After Second Stage of Preprocessing

and right. At first, we get the carts location from the environment and then use that information to zoom on the objects (fig: 4.3). This operation brings the feature size to  $3 \times 120 \times 120 = 43200$ . Although this operation omits some information about the object's displacement, this is a necessary trade off we had to take. At this stage the image contains 3 channels, red, green and blue. We perform channel reduction on the image by applying torchvision's grayscale filter on it (fig: 4.4), turning the image into  $1 \times 120 \times 120$ . Furthermore, we resize the image down to  $1 \times 30 \times 30$  by reducing the image resolution (fig: 4.5). Following all of these operations, we managed to significantly reduce the feature size from 720000 to 900. We could not perform further reduction operations without losing any more valuable information. The resized image at hand represents the object's current state but we wished our learning agent to capitalize on the transitions rather than the current state itself. Let  $S_t$  be the graphic condition of the environment at any time step  $t$ , then the transitions are represented by  $\Delta S_t = S_t - S_{t-1}$ , These transitions  $(\Delta S_1, \Delta S_2, \Delta S_3, \dots, \Delta S_t)$  will function as the input for our agent. A sample state transition is shown in fig: 4.6.

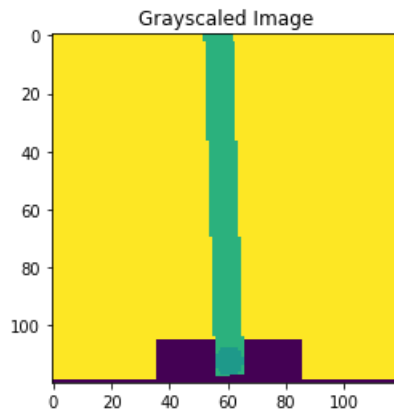


Figure 4.4: Sample Grayscaled Image After Third Stage of Preprocessing

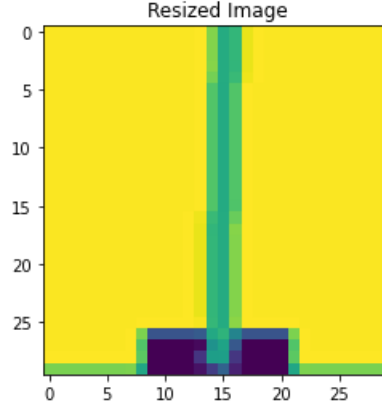


Figure 4.5: Sample Resized Image After Fourth Stage of Preprocessing

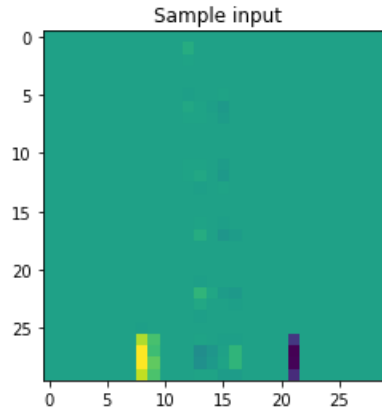


Figure 4.6: Sample Transition Data as Image

### 4.3 Parameter Selection

Selecting proper parameter was not an easy task since our model has to show decent learning rate, be efficient enough to be employed in a real-time learning scenario and ensures data security. Most challenging aspect was to find a proper balance that satisfies all of those requirements. The encryption parameter directly depends on the level of operation performed on the ciphertext. And the operation level is defined by the Neural Network parameters. The deeper the level, the costlier it is. Higher number of operations on an encrypted vector threatens the vector to be unusable by increasing the noise threshold beyond tolerance. In case that happens, the data will need to be bootstrapped, making it even slower. On the other hand, insufficient number of operations will not allow the model to learn and optimize itself efficiently, taking longer time to reach convergence. We have tried out several combinations and observed their performance. We have tested learning rate by using chosen parameters on plaintext learning model and measured time taken by using those on encrypted model.

### 4.3.1 CNN Layers

The depth of layers in a CNN model tends to increase with the number of features i.e. pixels in a standard input image. Layer's depth helps feature extraction and classifying the input images. In contrast, a high number of layers makes the model sluggish because of the high amount of computation. In our case, this is even more severe as we are working with encrypted computation which is hundred times slower than regular plaintext computation. So we opted to use a simple layer structure consisting of a convolution layer and two fully connected linear layers with an unorthodox activation function in between each of the layers. The activation functions in a neural network is what makes it non-linear which is needed by the learning algorithm. Homomorphic functions however, work better with linearity. The CKKS scheme in particular, achieves homomorphism over polynomial rings and can evaluate only polynomial functions with relatively small degrees. The popular activation functions like sigmoid and rectified linear are not polynomial functions, thus computing these functions over ciphertext is not possible as of yet.

$$\begin{aligned}\text{Sigmoid} &: z \rightarrow 1/(1 + \exp(-z)) \\ \text{Rectified Linear} &: z \rightarrow \max(0, z)\end{aligned}$$

There are several comparison functions proposed in recent papers [5] [6] for homomorphic schemes, but these are too expensive to employ on a large scale, for example evaluating the rectified linear function or the max-pooling operation on a network with hundreds of values. Assuming more efficient comparison schemes emerge in the future, integrating these operations on a network may become feasible.

According to the solution of Xie et al. [20] we opted to use a low degree polynomial to approximate an activation function, or rather a variation of it adopted by Dowlin et al. [21] in their work, that is the square activation function.

$$\text{Square Activation: } z \rightarrow z \times z$$

The square activation is the lowest degree polynomial which does not increase our multiplicative depth by too much and also achieves some level of non-linearity.

The description of the network is provided below

- **Convolution Layer:** Takes  $30 \times 30$  image as input. Has 4 kernels of size  $7 \times 7$ , with strides of 3. The output size is therefore,  $4 \times 8 \times 8$ .
- **Square Activation:** Performs  $Z \rightarrow Z \times Z$  operation on the convolution layer's output. The output size remains unchanged.
- **Reshape:** Flattens the square activation layer's output by reshaping it. Input size is  $4 \times 8 \times 8$  and the output size is 256.
- **Fully Connected Layer:** Fully connects 256 input nodes with 64 output nodes. The output size of this layer is therefore 64.

- **Square Activation:** Performs  $Z \rightarrow Z \times Z$  operation on the previous fully connected linear layer's output. The output size remains unchanged.
- **Fully Connected Layer:** Fully connects 64 input nodes with 2 output nodes. The output size of this layer is the size of our actionspace which is 2.

Fig: 4.7 represents the configuration of the derived neural network

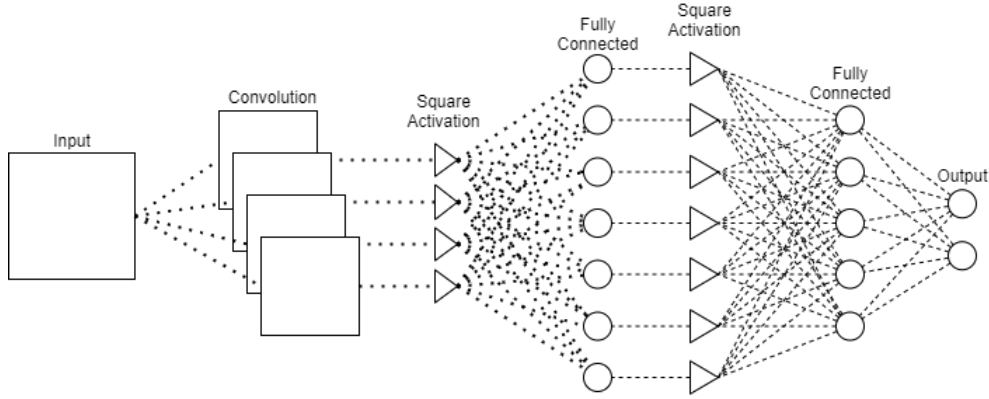


Figure 4.7: Configuration of CNN

### 4.3.2 Encryption Parameters

Every homomorphic encryption scheme has its own set of parameters, and the CKKS scheme is no exception to that. The encryption parameters for the CKKS scheme are  $N$ ,  $Q_{L+1}$  and  $k$ , where  $N$  represents the polynomial modulus degree,  $Q_{L+1}$  represents a series of coefficient modulus and  $k$  stands for the scaling factor.

**Polynomial modulus degree:** This parameter directly affects

- Number of coefficients present in plaintext polynomials.
- Size of sample ciphertext - Computational efficiency (bigger is worse).
- Level of security (bigger is better).

**Scaling factor:** Encoding a vector of real numbers into a plaintext polynomial is the first stage in the CKKS technique. The encoding precision for the binary sequence of the value is defined by the scaling factor. Conceptually, we're addressing binary precision as shown below:

Floating-point representation

$$1.01011 = \underbrace{101011}_{\text{significand}} * \underbrace{2^{-5}}_{\text{scaling factor (base exponent)}}$$

**Coefficient modulus:** This parameter is not a single number; it is in fact a series of prime numbers ( $[q_0, q_1, q_2, q_3, \dots, q_L, q_{L+1}]$  where  $L$  is the multiplicative depth).



The coefficient modulus influence

- Size of sample. ciphertext
- Number of encrypted multiplications supported.
- Level of security (bigger is worse).

Fortunately, we do not have to define the prime numbers ourselves. The SEAL library generates the prime numbers itself from given bit sizes. So, to set this parameter, we just have to provide a list of bit sizes. Each of the coefficient modulus can have a length of 60 bits at most and be congruent to 1 modulo  $2^{\text{poly modulus degree}}$ .

The convolution function for encrypted images provided by the TenSeal library performs 2 multiplication operations on ciphertexts (more discussion in the next section). Additional 4 multiplication operations are done by the other network layers. Each square activation and linear layer performs exactly 1 multiplication. The multiplicative depth in this context is therefore, 6 meaning we need a total of  $L + 2$  or 8 coefficient moduli. The other parameter components have to be chosen carefully as they are dependent on one another given the security level. For our application we chose to achieve 128-bit security. According to CKKS performance test [38], 128-bit security can be achieved with a polynomial modulus degree of  $2^y : y \geq 12$  with an upper bound for  $\sum_{i=0}^{L+1} (\log Q_i)$  corresponding to each value of  $y$ . Here  $\sum_{i=0}^{L+1} (\log Q_i)$  simply represents the total sum of coefficient modulus bit sizes. We need to use a higher polynomial modulus degree if the upper bound is surpassed in order to keep the same level of security. As we discussed earlier, polynomial degree directly affects computational efficiency, we generally want to choose the lowest possible value of  $y$ . However, choosing a relatively low value of  $y$  means we cannot use relatively higher coefficient modulus bit sizes. And for the CKKS scheme in TenSeal, all the bit sizes except the first and the last one has to be equal with our scale factor  $k$ . This means we are trading off precision with computation time. Lower precision results in inaccurate computations which is a severe issue since most of the values we are working with are between 0 and 1. It also threatens the learning rate of our agent. Since the precision carries significant importance, we wanted to ensure at least 20-bit precision. This gives us the minimum polynomial modulus degree of  $2^{13} = 8192$  with a maximum bit scale of 26 bits, keeping 5 bits for integers. Hence, the first and last value of coefficient modulus is  $26 + 5 = 31$ bits. The associated keys and ciphertexts (using our sample input) generated using these parameters occupy sizes of 810.06 KB (for all of the keys) and 465.83 KB respectively.

## 4.4 Data Encryption

Before we move on to encrypting images, the convolution function for ciphertext needs to be illustrated first. The convolution operation on ciphertext does not follow the same procedure as regular convolution. TenSeals encrypted convolution

operation adopts the technique of Johnson et al. [39], representing the convolution function as a single matrix multiplication operation. This methodology necessitates encrypted matrix-plain vector multiplication, which they accomplished by conducting element-wise matrix transpose multiplication with a duplicate plain vector. Finally, the result is rotated and accumulated into a single vector. The entire operation requires the input image to be represented in a one-dimensional vector rather than a four-dimensional tensor used in regular convolution. This transformation cannot be done easily in encrypted state, it has to be done as a pre-processing step before the data is encrypted. Unfortunately, this also limits the number of convolution operations supported on a ciphertext. Multiple convolutions would require the ciphertext to be deciphered, reshaped and transformed again with parameters of the next convolution. The transformation function initially maps the pixel values that overlap the kernel matrices with each stride and stacks them together resulting in a bigger matrix with repeated pixels. Then that matrix is translated into a column using vertical scan. This requires the kernel shape and stride to be known by the encryption module. The function generates a window size that tells the convolution function how to map each input value to its corresponding kernel value. After transforming the image into a single column, it is treated as a regular vector and is encrypted using the context generated from the parameters discussed in the previous section.

## 4.5 Agent structure

As we have discussed earlier, we have two distinct system architectures both fulfilling our objective yet specializing in different scenarios. Intuitively, it is obvious that a single agent design will not be sufficient in serving both systems equally. Therefore, we needed to design two separate agents to satisfy system constraints. Despite being separate, both agents have a lot of similarities in terms of general structure and a few components, they also work using similar principles. The general structure of both agents follows the framework of Minh et al. [37] as they contain a TargetQ network and an experience replay. They also consist of one or several policy networks that serve as the policy function. The experience replay is implemented with a transition memory of 10000 units. Each transition  $T_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$  where  $s, a, r$  represents state, action and reward respectively and  $t$  stands for timestep. At each timestep current state  $s_t$ , current action  $a_t$ , reward obtained from performing said action  $r_t$ , observed effect of the action performed or next state  $s_{t+1}$  are packed in a transition unit  $T_t$  and pushed into the memory.

$$\text{Consider the function } R_t = \sum_{t=t_0}^{\infty} (\gamma^{t-t_0} \times r_t)$$

Where  $R_t$  represents the discounted cumulative reward,  $\gamma$  is the discount factor. The purpose of training the policy net is to maximize  $R_t$  for any given input state  $s_t$ . We'll use the fact that, for some policy, every Q function obeys the Bellman equation as our training update law.

$$Q(s, a) = r + \gamma Q(s', \pi(s')) \text{ [} s' \text{ \& } \pi \text{ represents the next state \& policy function ]}$$

To calculate temporal difference error  $\delta$ , we can use

$$\delta = Q(s, a) - \left( r + \gamma \times \max_a Q(s', a) \right)$$

Using  $\delta$ , we can calculate the Huber Loss function  $L$ , where

$$L(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \text{if } |\delta| \leq 1, \\ |\delta| - \frac{1}{2} & \text{else } |\delta| > 1 \end{cases}$$

We can then use  $L(\delta)$  to optimize our policy network's weights and biases. We are utilizing the Root Mean Squared Propagation algorithm for our optimizer, which is an unreleased iterative optimization approach introduced by Geoff Hinton in Module 6 of his Coursera Lesson. The RMSProp optimization works especially well with mini-batch learning of neural networks.

The target network estimates the value of  $\max_a Q(s', a)$  by taking a mini-batch  $B$  of states sampled uniformly random from the transition memory such a way that for any  $s \in S_B, s_i \neq \text{termination state}$  is worth noting that the target network's weights and biases are equal to the weights and biases of an old version of the constantly optimizing policy network. Basically, we're replacing the target network's parameter on a fixed interval  $C$ . This breaks the correlation of the target function and policy function.

### Learning Parameters:

Batch size  $|B|$  = experiment variable  $[32 \leq |B| \leq 128]$

Target update interval  $C = 20$

Learning Rate  $\alpha = 0.01$

Discount factor  $\gamma = 0.99$

Random action probability  $\varepsilon$  :

Starting value  $\varepsilon_0 = 0.99$

Lower Limit  $\varepsilon_f = 0.05$

Decay rate  $\Delta\varepsilon = 200$

At any step  $t$ , the current threshold of  $\varepsilon$ ,

$$\varepsilon_t = \varepsilon_f + (\varepsilon_0 - \varepsilon_f) \times e^{-t/\Delta\varepsilon}$$

The discussion above is true for both of the agents. However, each has its own unique internal structure to adapt with encrypted data according to the requirements of the architecture they were designed for.

### 4.5.1 Plain-Encrypted Agent

This agent contains two policy networks, a plaintext policy network and a ciphertext policy network. Both networks represent the same policy function since before determining the policy for a ciphertext input, the parameters of the plaintext policy network are copied into the ciphertext policy network. The optimizer uses the loss calculated from the plaintext policy network’s output and updates its weights and biases. Because both policy networks are being synchronized in seemingly real-time there is no need to optimize the weights and biases of the ciphertext policy network. Hence, the transition memory only holds the plaintext transition units generated from the replicated environment, so there is no need for a second target network. The plaintext policy network is defined as the learning agent in our Plain-Encrypted model and the ciphertext policy network is defined as the encrypted counterpart.

---

#### Algorithm 1 Learning module

---

```

1:  $S_t \leftarrow$  get transition from clone env
2:  $s_t \leftarrow$  preprocess  $s_t$ 
3:  $\omega \leftarrow$  sample random number in range 0-1
4: if  $\omega$  less than  $\mathcal{E}_t$  then
5:    $a_t \leftarrow$  random action from action space
6: else:
7:    $a_t \leftarrow Q_{pp}(s_t)$ ,  $Q_{pp}$  is the plaintext policy function
8:  $r_t, s_{t+1} \leftarrow$  feed  $a_t$  to clone env then record reward and next transition
9:  $s_{t+1} \leftarrow$  preprocess  $s_{t+1}$ 
10: store transition unit  $T_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$  into memory  $D$ 
11:  $B \leftarrow$  sample  $|B|$  transitions from  $D$  randomly
12:  $y_B \leftarrow r_B + \gamma \times \max_a Q_t(s'_B, a_B)$ ,  $Q_t$  is the target function
13: perform optimizer step on  $L(Q_{pp}(s_B, a_B) - y_B) = 0$ 

```

---



---

#### Algorithm 2 Evaluation module

---

```

1:  $\hat{s}_t \leftarrow$  receive and deserialize user’s encrypted transition
2: set param ( $Q_{cp}$ )  $\leftarrow$  param( $Q_{pp}$ ),  $Q_{cp}$  is the ciphertext policy function
3:  $\hat{a}_t \leftarrow Q_{cp}(\hat{s}_t)$ 
4: serialize and return  $\hat{a}_t$ 

```

---

For our experiment, we have incorporated a phase-switching mechanism that runs the agent in plaintext phase and ciphertext phase with a predefined interval. In our case, for every 50 episodes of the plaintext optimization phase, there are 10 episodes of the encrypted evaluation phase.

### 4.5.2 Pure Encrypted Agent

The Pure-Encrypted agent has only one policy network for representing its policy function as this agent works only on ciphertext, there is no need for a separate

plaintext policy network. One of the most challenging tasks of encrypted learning is generating the gradients needed for optimization. Not only does it add a load of computation time but also some of the functions needed for generating gradients are not supported by current homomorphic schemes. Moreover, the generated gradients would also be ciphertexts as doing any plaintext-ciphertext operation results in a ciphertext.

Consider  $E$  as the encryption function

$$\begin{aligned} m \oplus E(m) &\equiv E(2m) [\oplus \text{ represents encrypted add function}] \\ m \otimes E(m) &\equiv E(m^2) [\otimes \text{ represents encrypted mult function}] \end{aligned}$$

This means either the optimizer function needs to be designed for encrypted gradients, meaning the weights and biases would also be encrypted, or the generated gradients needs to be sent to the user for decryption and returned to the agent so that the original optimizer can work on plaintext weights and biases. Of course, the model could work with encrypted weights and biases, except plaintext-ciphertext operations are much faster than ciphertext-ciphertext operations and add very little noise compared to any ciphertext-ciphertext operations. This means, the weights and biases must be bootstrapped every now and then to limit noise level to make it not exceed the tolerable threshold. All of this would make the model too slow to work with. The second option is not viable either as decrypting hundreds of gradients at each time step puts a burden on the user's machine.

Instead, we opted to follow an alternative route. Using pytorch, we designed the layers to generate the plaintext gradient function by generating a dummy plaintext output as well as our actual encrypted output. The dummy plaintext output is generated by forward passing a closecorrelated plaintext input through an identical plaintext function using the same sets of parameters. The strength of correlation between the plaintext input and actual encrypted input is proportional with the model's learning efficiency and inversely proportional with data security. In our case, we are using a rather loose correlation, the computed mean of actual input transition's pixel values. A plaintext input with the same dimension of actual input is created using only the mean value. The close-correlated input and the actual encrypted input pass through their respective functions. The dummy output generated by plaintext functions as well as the actual encrypted output are sent to the user's machine. The user decrypts the encrypted output values then replaces the dummy output values with the actual ones. The user computes overall loss and then sends it to the agent to perform the backward operation based on the overall loss. Fig: 4.8 illustrates the network with our designed custom layer. Both the policy and target network hold this internal structure and performs the same kind of operation. The mean values, encrypted states, actions, rewards and encrypted next states are pushed into the replay memory for optimization's sake. The learning function of this agent is exactly the same as the learning function of Plain-Encrypted agent provided in Algorithm: 1 without input collection and preprocessing since that part is handled in the user's domain. Another exception is, before computing individual loss, the data needs to be sent to the user then the overall loss needs to be retrieved from the user's domain. This procedure is not the most efficient as the user still needs to decrypt twice the batch size of unit outputs, one for the batch output of  $Q_p(\hat{s}_B, a_B)$ , another for the batch output of  $Q_t(\hat{s}'_B, a_B)$ . Also, there is added network overhead

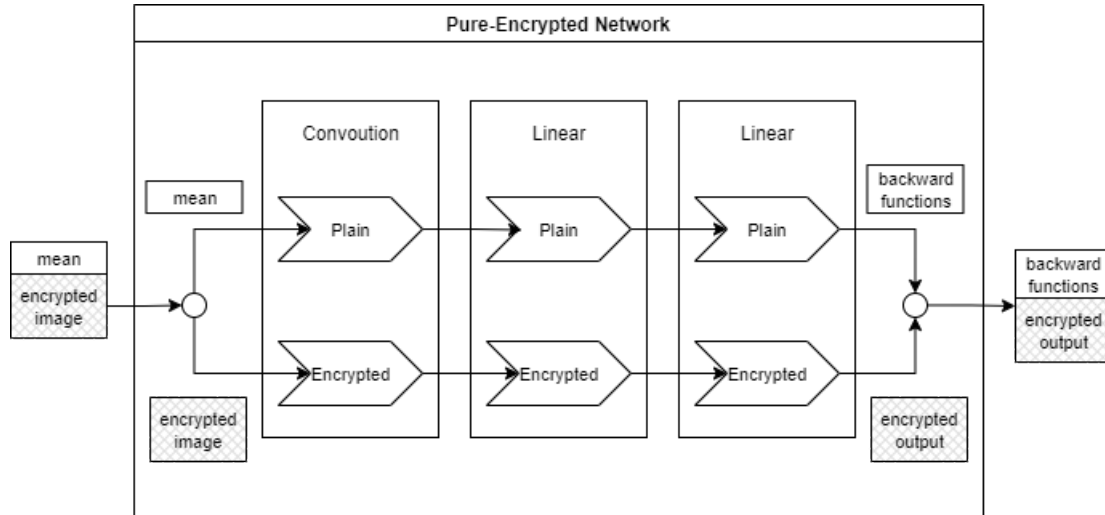


Figure 4.8: High Level Custom Network of Pure Encrypted Agent

to consider. But considering the other options, this approach was much simpler and faster compared to them. We will observe how effective our alternate approach is in Chapter 5. This agent does not require a separate evaluation function.

# Chapter 5

## Result Analysis

In this section we are going to discuss and analyze our findings. We have trained several agents using our methods and details about a few of them along with their learning graph will be presented to compare with equivalent plaintext versions of those agents. But first, we need to define a few terms and attributes that are present in the figure titles.

*PlE* → short for Plain-Encrypted agent

*PuE* → short for Pure Encrypted agent

–*d* → means the figure shows the individual episode durations in terms of time steps

–*a* → means the figure shows the average time steps in the last 20 episodes

–*l* → means the figure is of the learning module (only for *PlE*)

–*e* → means the figure is of the evaluation module (only for *PlE*)

For reference, we have trained a demo plaintext agent with the exact same general structure and learning functions as our designed agents. The agent is trained for 500 episodes following below parameters.

$$|B| = 128$$

$$C = 20$$

$$\alpha = 0.01$$

$$\gamma = 0.99$$

$$\varepsilon_0 = 0.99$$

$$\varepsilon_f = 0.05$$

$$\Delta\varepsilon = 200$$

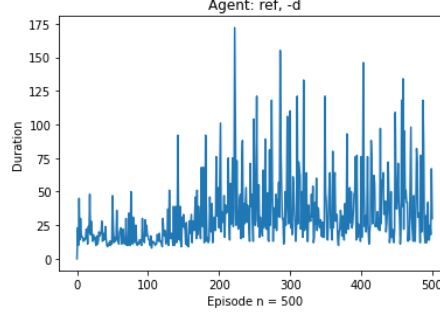


Figure 5.1: Reference Agent's Durations Over Episodes

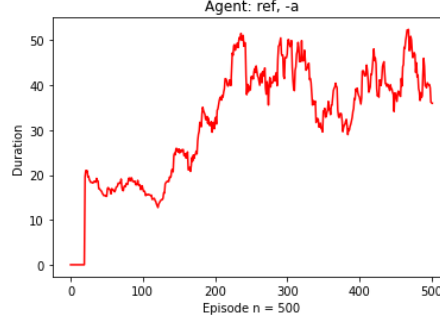


Figure 5.2: Reference Agent's Average Most Recent Durations Over Episodes

Figure 5.3: Reference Agent

Figure: 5.1 & 5.2 represents the learning graph of the reference agent. Within 500 episodes, the agent managed to get a maximum episode duration of 172-time steps with a maximum episode duration averaged across the most recent 20 episodes of 52.45-time steps. It is worth noting that our selected environment has a lower bound of 10-time steps for any given action sequence in an episode, which means an untrained model can achieve an average episode duration of 10 to 13-time steps. The reference agent takes about 8.5 minutes to complete 500 episodes.

In total, we are going to present observations of 3 trained agents, one of them being a PlainEncrypted *PIE* agent and the other two are Pure-Encrypted *PuE* agents. The two *PuE* agents are trained using different batch sizes. The first one used a batch size of 128, and the second one's batch size was 32. This was done to observe the effect of batch size on learning rate and computation cost in terms of time taken. Because for training, our agents have to evaluate  $|B|$  number of states on both policy and target functions to calculate the loss and optimize, so at each step the agent has to evaluate  $2 \times |B|$  states. This is a big concern since unit evaluation time of encrypted data is around 1000 times more than unit evaluation time of plaintext data. Therefore, the batch size  $|B|$  influences the training time heavily when it comes to learning on encrypted data. The Plain-Encrypted agent did not require to be trained on different batch sizes, since the added computation time for plaintext learning due to  $|B|$  is negligible compared to encrypted learning. All the other parameters of all agents are the same as the reference agent.

Figure: 5.4 - 5.7 demonstrates the performance of our Plain-Encrypted agent. The agent elapsed a total of 666 episodes, from which 536 episodes were spent on plain-



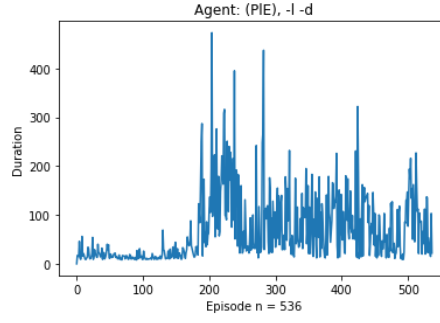


Figure 5.4: *PIE* Learning Phase Durations Over Episodes

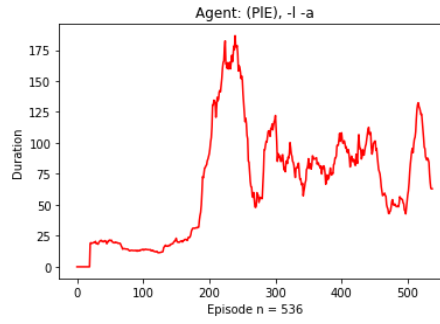


Figure 5.5: *PIE* Learning Phase Average Most Recent Durations Over Episodes

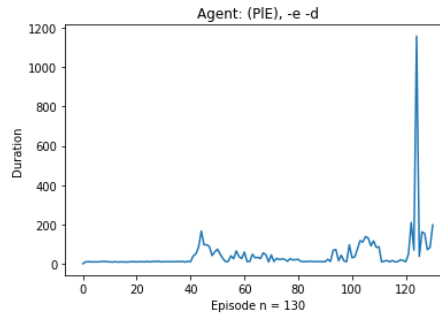


Figure 5.6: *PIE* Evaluation Phase Durations Over Episodes

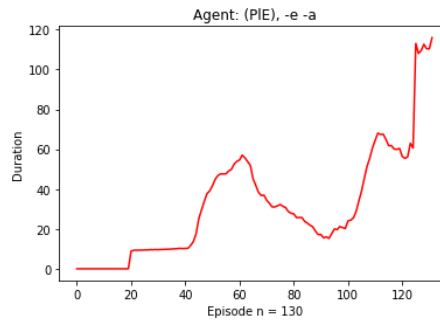


Figure 5.7: *PIE* Evaluation Phase Average Most Recent Durations Over Episodes

Figure 5.8: Plain-Encrypted Agent

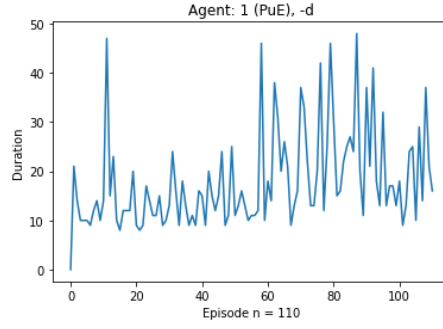


Figure 5.9:  $PuE - 1$  Durations Over Episodes

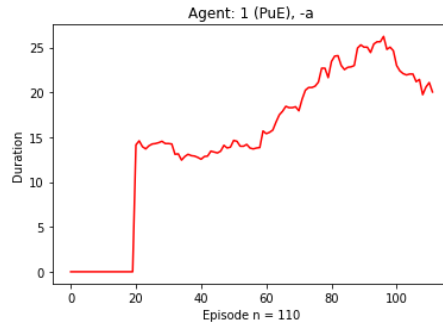


Figure 5.10:  $PuE - 1$  Average Most Recent Durations Over Episodes

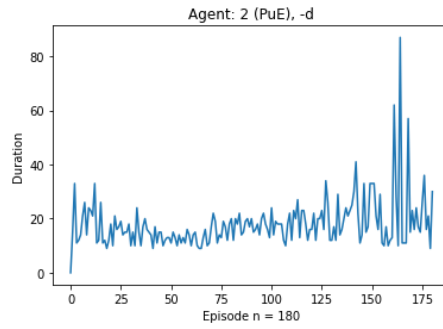


Figure 5.11:  $PuE - 2$  Durations Over Episodes

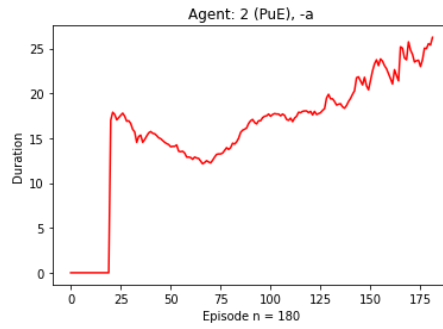


Figure 5.12:  $PuE - 2$  Average Most Recent Durations Over Episodes

Figure 5.13: Pure Encrypted Agents

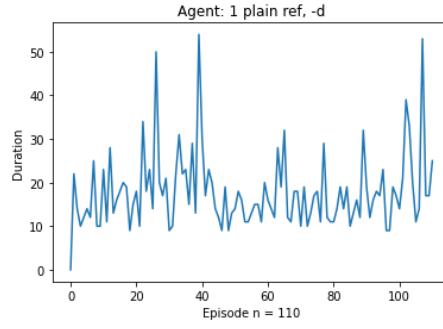


Figure 5.14:  $PuE$  – 1 Plain ref Durations Over Episodes



Figure 5.15:  $PuE$  – 1 Plain ref Average Most Recent Durations Over Episode

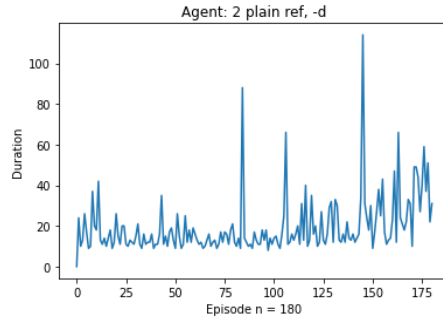


Figure 5.16:  $PuE$  – 2 Plain ref Durations Over Episodes

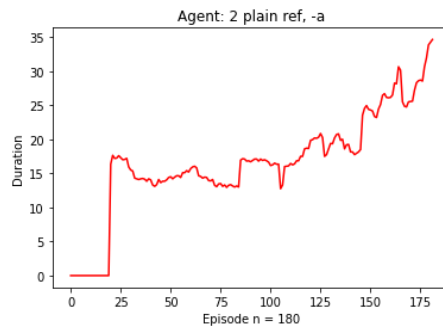


Figure 5.17:  $PuE$  – 2 Plain ref Average Most Recent Durations Over Episodes

Figure 5.18:  $PuE$  plaintext equivalent Agents

Agent: 1 <i>PuE</i>	Agent: 2 <i>PuE</i>
Operates on ciphertexts	Operates on ciphertexts
With a batch size of 128	With a batch size of 32
Trained for 110 episodes	Trained for 180 episodes

Table 5.1: Description of Pure Encrypted Agents

Agent: 1 Plain ref	Agent: 2 Plain ref
Operates on plaintexts	Operates on plaintexts
With a batch size of 128	With a batch size of 32
Trained for 110 episodes	Trained for 180 episodes

Table 5.2: Description of Pure Encrypted Agents Equivalents

text learning and 130 episodes on ciphertext evaluation. Figure: 5.4 describes the distribution of individual episode durations across the learning episodes and figure: 5.5 represents the graph of average episode durations. Figure: 5.6 & 5.7 describes the evaluation episodes in the same way. Due to our phase switching mechanism, the evaluation graphs are plotted for 10 episodes after every 50 learning episodes. So, in the evaluation's graphs, the weights and biases are updated in every 10 episodes. We are using the term evaluation because the random action probability is set to 0 whenever the agent is in this phase.

For our *PuE* agents, we have trained 2 different plaintext agents to properly visualize the effectiveness of encrypted learning. The plaintext agents are trained for the same number of episodes as the *PuE* agents with the same sets of parameters. Let us define the Pure-Encrypted agents in table 5.1. And let's also define their plaintext equivalents (termed as plain ref) in table 5.2. The learning graphs of both *PuE* agents are given in figure: 5.9 - 5.12. The learning graphs of the plaintext equivalents of both *PuE* agents are given in figure: 5.14 - 5.17.

## 5.1 Cost Analysis

Before we proceed to analyze the time cost of each agent, we need to measure the unit time costs relevant to our network structure and selected encryption parameters discussed in section 4.3.2. The unit time cost is necessary to properly understand and estimate computation cost of any agent with known learning parameters. Our findings are presented in table: 5.3.

The Plain-Encrypted agent that we trained for 536 episodes and evaluated on ciphertexts for 130 episodes took in total of approximately 9 minutes to complete its learning phase and around 4 hours for completing all the encrypted evaluations. Each 10-episode evaluation phase takes an average of 18 minutes to complete. The amount of time taken depends directly on the duration achieved on an episode. Hence, it can be said that implicitly, the time cost depends on the agent's performance.

Unit cost analysis (Averaged across 100 iterations)	
Sample image encryption time	0.012158 seconds
Sample output decryption time	0.000553 seconds
Sample encrypted image evaluation time	2.513748 seconds
Unit time for encrypted learning step	2.536583 seconds
Sample plaintext image evaluation time	0.003124 seconds
Encrypted-Plaintext evaluation time ratio	804.51633
Encryption context serialization time	0.016804 seconds
Encryption context serialization time without sk	2.134047 seconds
Sample encrypted image serialization time	0.015135 seconds

Table 5.3: Table of Unit Cost Analysis

In the case of the Pure Encrypted agents, Agent: 1 with a batch size of 128, took 171 hours to complete 110 episodes and Agent: 2 having a batch size of 32, took 50 hours to complete 180 episodes. This is where the difference of batch sizes becomes apparent. The first agent had to evaluate  $1+128+128$  (1 current state on policy, 128 batch states on policy, 128 batch next state on target) encrypted images on each optimization step where the second agent had to evaluate  $1+32+32$  ciphertexts. The agents completed a total of 1987 and 3280 steps respectively. If all of those steps were optimization steps, according to our unit cost test, the agents would've taken-

$$1987 \times (1 + 128 + 128) \times 2.536583 = 1295328.94\text{sec} \equiv 460 \text{ hours [ for Agent 1 ]}$$

$$3280 \times (1 + 32 + 32) \times 2.536583 = 540799.5\text{sec} \equiv 150 \text{ hours [ for Agent 2 ]}$$

But not all steps are optimization steps. The agents do not start optimization until their transition memory holds  $|B|$  or more transition units. Additionally, there is the random action factor to consider. If an action is chosen randomly, no optimization takes place within that time step. This is why our agent took 171 hours and 50 hours respectively instead of 460 hours and 150 hours.

## 5.2 Performance Analysis

The biggest factor regarding performance when it comes to encrypted arithmetic is precision. Precision defines the upper bound of prediction accuracy. We have conducted precision tests on encrypted data by forward passing numerous plaintext sample images and their ciphertexts produced by our encryption parameter through the policy and target function, then measured the absolute error.

Using Algorithm 1 with  $n=100$ , we computed the absolute error of ciphertext computation on our circuit  $e_a = 0.006$ . This interprets as, we are only going get false output if  $|a_i - a_j| \leq 10^{-2}$  for any  $a_i \in A, a_j \in A$  where  $A$  is the predicted action value set. In our case,  $|A| = 2$  which makes the predicted action values less likely to be within that margin of error. This proves one of our agents, the Plain-Encrypted agent, to function as effectively as any plaintext version of the model. In fact, from figure: 5.4 & 5.6 we can see in an encrypted evaluation phase, the agent managed

---

**Algorithm 3** Precision Test

---

```
1:  $S \leftarrow$  sample n states from state-space  $\langle s_1, s_2, s_3 \dots s_n \rangle$   
2:  $\hat{S} \leftarrow$  encrypt each element of S into new vector  $\hat{s}_i = \text{Encrypt}(s_i)$   
3:  $A \leftarrow$  compute plaintext values on function  $Q(S)$   
4:  $\hat{A} \leftarrow$  compute encrypted values on homomorphic function  $\hat{Q}(\hat{s})$   
5:  $A_d \leftarrow$  Decrypt  $\hat{A}$   
6:  $e_a \leftarrow$  compute average absolute error  $|\overline{A - A_d}|$ 
```

---

Agents	Max score	Max Average score
Agent: 1 (PuE)	48	26.25
Agent: 1 plain ref	54	23.6
Agent: 2 (PuE)	87	26.25
Agent: 2 plain ref	114	34.65

Table 5.4: Table of Score achieved by the agents

to achieve a score of 1159 which even the plaintext learning module could not get. The absolute maximum the plaintext module could get was a score of 482. Then encrypted module managed to get a maximum of 115.95 average score in the most recent 20 episodes, according to figure: 5.7. It can also be seen that the graph was still climbing when our iterations stopped.

Because of being computationally expensive, we could not conduct experiments with total learning episodes higher than 180, when it came to the Pure Encrypted agents. As discussed in the cost analysis section, completing 180 episodes with a batch size of 32, the agent took about 2 days. The first agent that we defined earlier, with batch size of 128, could not even complete more than 110 episodes over a span of 1 week. This gravely affects the performance of the agents as we cannot compare it with the first reference plaintext agent. From figure: 5.2, we can see that the plaintext reference agent only starts to show noticeable improvement around the 200-episode mark. That is why we needed to train equivalent plaintext agents for comparison. Table 5.4 presents necessary information extracted from figures: 5.9 - 5.17.

From the start we can see the overall performance of encrypted learning agents is not up to the level of plaintext learning agents. This was expected since we are generating gradients using close-correlated inputs. But judging from the graphs in figure 5.10 & 5.12, we can surely say that these agents were improving from their initial condition i.e., learning. Therefore, our alternative approach did prove to be fruitful. The first PuE agent even managed to get a higher avg score than its plaintext equivalent. The difference in batch size impacts the learning rate of these agents. As we can see in figures 5.10 & 5.12, both agents got the same average maximum score of 26.25, but Agent: 2 got to that mark around the end of its iterations at exactly 180<sup>th</sup> episode where Agent: 1 passed that mark around 95<sup>th</sup> episode.

Comparing the figures: 10 with 15, and 12 with 17, we can see the encrypted learn-

ing agents show similar learning patterns with their respective equivalent plaintext agents. The encrypted agents are getting the patterns with a few delayed episodes, for example- the Agent: 1 plain ref achieves a local maxima between episodes 20 to 60 (figure: 5.15) whereas Agent: 1 achieves the same feat between episodes 60 to 110. This means Agent: 1 may achieve convergence with a greater number of episode iterations compared to Agent: 1 plain ref. The same can be said for Agent: 2 and its equivalent Agent: 2 plain ref.

All of the analysis was conducted using a machine having a Core i7-7700 x64 CPU with a speed of 3.60 GHz and an Nvidia GeForce GT 710 GPU.

# Chapter 6

## Conclusion and Future Works

### 6.1 Conclusion

In this thesis, we have developed two deep reinforcement learning agents that work on encrypted data using CKKS encryption scheme to generate ciphertexts. Both agents acquire states as encrypted images from the user’s environment and generate encrypted action values. At first, we developed system architectures to facilitate learning in real-time where encryption standards are set by the user. Furthermore, we have designed learning agents to match prescribed architectures. In addition to that, we have conducted a thorough comparison of the designed agents with conventional regular learners. Considering the unit evaluation time according to our analysis, the first agent of our design can cope with a constant exchange of state and action given the proper hardware configuration, and can be shaped into certain real life scenarios. The second agent which learns on ciphertexts, can be used where heavy interaction is not required, for example- optimizing a song recommendation system in a streaming service. In addition to that, an inclusion of both agents can be visualized where the ultimate goal is defined by the first agent and user’s preferences are handled by the second agent. If we take the trend of computational power into consideration, adopting FHE schemes in a feasible ML driven cloud service is not far into the future.

### 6.2 Future Works

Of course, the system that we described in our paper including the agents themselves are not the most efficient, and there is still room for improvement. The homomorphic cryptography itself has not seen much progression until recently and it is still being explored. We can be positive that in the near future we will see newer and more efficient schemes or a rework of the existing ones that will allow more arithmetic operations, not being limited to only addition and multiplication. Given the advancement, encrypted backpropagation can be both possible and feasible. This will guarantee the increasing performance of our encrypted learning agent.



Nevertheless, this research was conducted to implement and observe the overall performance of a basic secure real-time learning agent. In the future, we would like to use the framework from this research to build a learning agent that attempts to solve a real-life problem.

# Bibliography

- [1] R. L. Rivest, L. Adleman, M. L. Dertouzos, *et al.*, “On data banks and privacy homomorphisms,” *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.
- [2] C. Gentry, “Fully homomorphic encryption using ideal lattices,” in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pp. 169–178, 2009.
- [3] J.-S. Coron, D. Naccache, and M. Tibouchi, “Public key compression and modulus switching for fully homomorphic encryption over the integers,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 446–464, Springer, 2012.
- [4] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, “(leveled) fully homomorphic encryption without bootstrapping,” *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.
- [5] J. H. Cheon, A. Kim, M. Kim, and Y. Song, “Homomorphic encryption for arithmetic of approximate numbers,” in *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 409–437, Springer, 2017.
- [6] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, “Bootstrapping for approximate homomorphic encryption,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 360–384, Springer, 2018.
- [7] H. Chen, I. Chillotti, and Y. Song, “Improved bootstrapping for approximate homomorphic encryption,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 34–54, Springer, 2019.
- [8] R. Agrawal and R. Srikant, “Privacy-preserving data mining,” in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pp. 439–450, 2000.
- [9] R. U. Haque, A. Hasan, Q. Jiang, and Q. Qu, “Privacy-preserving k-nearest neighbors training over blockchain-based encrypted health data,” *Electronics*, vol. 9, no. 12, p. 2096, 2020.

- [10] Y. Qi and M. J. Atallah, “Efficient privacy-preserving k-nearest neighbor search,” in *2008 The 28th International Conference on Distributed Computing Systems*, pp. 311–319, IEEE, 2008.
- [11] M. Barni, C. Orlandi, and A. Piva, “A privacy-preserving protocol for neural-network-based computation,” in *Proceedings of the 8th workshop on Multimedia and security*, pp. 146–151, 2006.
- [12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.
- [13] O. Goldreich, S. Micali, and A. Wigderson, “How to play any mental game, or a completeness theorem for protocols with honest majority,” in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pp. 307–328, 2019.
- [14] C. Orlandi, A. Piva, and M. Barni, “Oblivious neural network computing via homomorphic encryption,” *EURASIP Journal on Information Security*, vol. 2007, pp. 1–11, 2007.
- [15] T. Chen and S. Zhong, “Privacy-preserving backpropagation neural network learning,” *IEEE Transactions on Neural Networks*, vol. 20, no. 10, pp. 1554–1564, 2009.
- [16] M. Naehrig, K. Lauter, and V. Vaikuntanathan, “Can homomorphic encryption be practical?,” in *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pp. 113–124, 2011.
- [17] G. E. Dahl, D. Yu, L. Deng, and A. Acero, “Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition,” *IEEE Transactions on audio, speech, and language processing*, vol. 20, no. 1, pp. 30–42, 2011.
- [18] T. Graepel, K. Lauter, and M. Naehrig, “ML confidential: Machine learning on encrypted data,” in *International Conference on Information Security and Cryptology*, pp. 1–21, Springer, 2012.
- [19] L. J. Aslett, P. M. Esperança, and C. C. Holmes, “Encrypted statistical machine learning: new privacy preserving methods,” *arXiv preprint arXiv:1508.06845*, 2015.
- [20] P. Xie, M. Bilenko, T. Finley, R. Gilad-Bachrach, K. Lauter, and M. Naehrig, “Crypto-nets: Neural networks over encrypted data,” *arXiv preprint arXiv:1412.6181*, 2014.
- [21] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, “Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy,” in *International Conference on Machine Learning*, pp. 201–210, PMLR, 2016.
- [22] Y. Aono, T. Hayashi, L. Wang, S. Moriai, *et al.*, “Privacy-preserving deep learning via additively homomorphic encryption,” *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2017.

- [23] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, “Low latency privacy preserving inference,” in *International Conference on Machine Learning*, pp. 812–821, PMLR, 2019.
- [24] M. Minelli, *Fully homomorphic encryption for machine learning*. PhD thesis, PSL Research University, 2018.
- [25] J. Suh and T. Tanaka, “Sarsa (0) reinforcement learning over fully homomorphic encryption,” *arXiv preprint arXiv:2002.00506*, 2020.
- [26] J. Park, D. S. Kim, and H. Lim, “Privacy-preserving reinforcement learning using homomorphic encryption in cloud computing infrastructures,” *IEEE Access*, vol. 8, pp. 203564–203579, 2020.
- [27] K. Laine, “Simple encrypted arithmetic library 2.3.1,” *Microsoft Research* <https://www.microsoft.com/en-us/research/uploads/prod/2017/11/sealmanual-2-3-1.pdf>, 2017.
- [28] A. Narang and D. Gupta, “Different encryption algorithms in cloud,” *International Journal of Engineering, Science and Mathematics*, vol. 7, no. 4, pp. 429–432, 2018.
- [29] N. Ruangchaijatupon and P. Krishnamurthy, “Encryption and power consumption in wireless lans-n,” in *The Third IEEE workshop on wireless LANS*, pp. 148–152, 2001.
- [30] M. A. Will and R. K. Ko, “Secure fpga as a service—towards secure data processing by physicalizing the cloud,” in *2017 IEEE Trustcom/BigDataSE/ICCESS*, pp. 449–455, IEEE, 2017.
- [31] F. Armknecht, C. Boyd, C. Carr, K. Gjøsteen, A. Jäschke, C. A. Reuter, and M. Strand, “A guide to fully homomorphic encryption.” Cryptology ePrint Archive, Report 2015/1192, 2015. <https://eprint.iacr.org/2015/1192>.
- [32] Z. Brakerski and V. Vaikuntanathan, “Efficient fully homomorphic encryption from (standard) lwe,” *SIAM Journal on Computing*, vol. 43, no. 2, pp. 831–871, 2014.
- [33] J. Fan and F. Vercauteren, “Somewhat practical fully homomorphic encryption,” *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, 2012.
- [34] Z. Brakerski and V. Vaikuntanathan, “Fully homomorphic encryption from ring-lwe and security for key dependent messages,” in *Annual cryptology conference*, pp. 505–524, Springer, 2011.
- [35] S. Huang, “Introduction to various reinforcement learning algorithms. part i (q-learning, sarsa, dqn, ddpg),” *Towards Data Science, Towards Data Science*, vol. 12, 2018.
- [36] D. Zhao, H. Wang, K. Shao, and Y. Zhu, “Deep reinforcement learning with experience replay based on sarsa,” in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–6, IEEE, 2016.

- [37] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [38] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, “ngraph-he2: A high-throughput framework for neural network inference on encrypted data,” in *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pp. 45–56, 2019.
- [39] A. Karpathy, J. Johnson, and L. Fei-Fei, “Visualizing and understanding recurrent networks,” *arXiv preprint arXiv:1506.02078*, 2015.