Log-based Secure Framework for Machine Learning Inference on Distributed Edge Clusters

Project - II (AI67002) report submitted to

Indian Institute of Technology Kharagpur

in fulfilment for the award

of

M.Tech. (IDDP 5 Year)

in

Electrical Engineering / Artificial Intelligence, Machine Learning, and Applications

by

Hardhik Mohanty
18EE3AI26

Under the guidance of

Dr. Ayantika Chatterjee

CENTRE OF EXCELLENCE IN ARTIFICIAL INTELLIGENCE

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

APRIL, 2023

DECLARATION

I certify that

(a) The work contained in this report has been done by me under the guidance of my supervisor.

(b) The work has not been submitted to any other Institute for any degree or diploma.

(c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

(d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Hardhik Mohanty

Date: April 28, 2023 (Hardhik Mohanty)

Place: Kharagpur (18EE3AI26)

Centre of Excellence in Artificial Intelligence
Indian Institute of Technology,
Kharagpur, India - 721302

CERTIFICATE

This is to certify that the project report entitled **Log-based Secure Framework for Machine Learning Inference on Distributed Edge Clusters**, submitted by **Hardhik Mohanty** (Roll Number: **18EE3AI26**), an IDDP (5 Year) student of the **Centre of Excellence in Artificial Intelligence** towards fulfillment of requirements for the award of the degree of Master of Technology in Electrical Engineering, Artificial Intelligence, Machine Learning, and Applications is a record of bonafide work carried out by him under my supervision and guidance during Spring Semester, 2022-23.

Ayantika Chatterjee

Dr. Ayantika Chatterjee

Centre of Excellence in Artificial Intelligence
Indian Institute of Technology,
Kharagpur

Place: Kharagpur
Date: 28/04/2023

Abstract

This thesis investigates the development and implementation of a secure framework that enables simultaneous execution of log-based anomaly detection and hand grasp recognition using machine learning on Hadoop Distributed File System (HDFS) system in an edge cluster. With the growing importance of Big Data and the Internet of Things (IoT) in various industries, the need for secure and efficient distributed computing systems has become increasingly critical. The proposed system employs a stacked-LSTM model for log-based anomaly detection and a deep learning-based vulnerability detection model to thoroughly examine executable files for anomalies and vulnerabilities. This multi-layered security approach safeguards the system's integrity while allowing it to process hand grasp recognition tasks using Exohand input data. The system architecture consists of a series of steps that involve distributing the executable file among worker nodes, generating log data, employing the stacked-LSTM model to predict anomalies, examining the source code for vulnerabilities, and ultimately predicting hand grasp types using a machine learning model. Experimental results focus on the performance of the secure framework for machine learning inference on the Google Coral Board, a popular edge computing platform. In particular, the secure framework obtained an accuracy of 98.73% and 83.56% in log-based anomaly detection task and hand grasp recognition task, respectively. After full-integer quantization, the memory requirement of the secure framework reduced by 73.33% and the processing speed-up increased by 5-fold. However, a drop in accuracy is observed in the secure framework due to model compression. The findings demonstrate a trade-off between memory reduction, speed-up performance, and accuracy when compression techniques, such as hybrid quantization and full-integer quantization, are applied to the machine learning models. This trade-off highlights the importance of balancing resource efficiency with accuracy in real-world applications.

Keywords: Log Anomaly Detection, Big Data, Hadoop, Edge Computing, Deep Learning, Security.

Contents

Chapter 1

Introduction

1.1 Motivation

Big data and the Internet of Things (IoT) are two technologies that have emerged as crucial tools for businesses to improve their operations and decision-making capabilities. Big data refers to the vast amounts of data that are generated through various digital interactions and IoT devices, while IoT refers to the network of physical devices, vehicles, and buildings that are embedded with sensors, software, and network connectivity to enable the collection and exchange of data [1]. The importance of big data and IoT lies in their ability to capture and analyze large amounts of data in real-time, providing valuable insights that can help businesses optimize their processes, reduce costs, and improve customer satisfaction. By harnessing the power of big data and IoT, businesses can gain a competitive edge by making data-driven decisions and implementing predictive analytics to forecast trends and identify opportunities [2].

Furthermore, Raspberry Pi clusters have become increasingly relevant in the big data world due to their affordability, scalability, and flexibility. A Raspberry Pi is a small, low-cost computer that is ideal for use in IoT devices due to its compact size, low power consumption, and versatile functionality. By clustering multiple Raspberry Pi devices together, businesses can create a scalable and cost-effective computing platform that can handle large amounts of data [3]. In a Raspberry Pi cluster, each device can be configured to perform specific tasks such as data processing, storage, or networking. This allows businesses to customize their computing infrastructure based on their specific needs and requirements. Raspberry Pi clusters are also highly scalable, as businesses can easily add or remove devices as their computing needs change. One of the key benefits of using a Raspberry Pi cluster in the big data world is its ability to handle large volumes of data. With the rise of IoT devices, there is an ever-increasing amount of data being generated, which can be challenging to manage and process. A Raspberry Pi cluster can provide the computing power needed to handle this data, enabling businesses to process and analyze data in real-time [4].

Hadoop is an open-source distributed processing framework that allows for the processing of large data sets across a network of computers. One of the core components of Hadoop is the Hadoop Distributed File System (HDFS), which is a distributed file system that can store and manage large volumes of data across a cluster of machines [5]. The key advantages of using Hadoop in the context of IoT and big data is its ability to handle unstructured data. IoT devices often generate large volumes of unstructured data, such as sensor readings or log files, which can be challenging to manage and process. Hadoop provides a way to organize and process this data, enabling businesses to gain valuable insights that can be used to optimize their operations and improve their decision-making capabilities [6]. Hadoop has also made it possible to perform tasks such as predictive analytics, machine learning, and data mining on massive amounts of data.

HDFS can be operated on a cluster of edge devices such as Raspberry Pi clusters. In this setup, the Raspberry Pi devices act as the nodes in the Hadoop cluster with identical computing resources and power [7]. Each device can be configured to run HDFS and the data can be distributed across the nodes in the cluster. This allows for distributed processing of data on the edge, which can be useful in scenarios where data needs to be processed in real-time or where internet connectivity may be limited. However, it is important to note that running HDFS on a cluster of edge devices can be challenging [8]. The limited computing resources and storage capacity of edge devices make it difficult to handle large volumes of data. Additionally, the network connectivity between the nodes in the cluster can be unreliable, which can lead to attacks and vulnerabilities impact the performance of HDFS. Therefore, security and optimization techniques are required to ensure that HDFS runs effectively on a cluster of edge devices.

Running HDFS on edge clusters such as Raspberry Pi clusters can make data processing faster and more efficient, but it also exposes the system to various security risks. The following are some of the attacks that could target an HDFS running on a Raspberry Pi cluster [9]:

1. Distributed denial-of-service (DDoS) attacks: In a DDoS attack, multiple devices are used to send a massive amount of traffic to a targeted network or server, overwhelming the system and causing it to crash. HDFS running on a Raspberry Pi cluster can be vulnerable to such attacks since the limited resources of the edge devices can be easily overwhelmed.

2. Man-in-the-middle (MIM) attacks: In an MIM attack, an attacker intercepts the communication between two devices and can either steal data or modify it before sending it to its intended destination. This type of attack is particularly dangerous for HDFS running on edge clusters since the communication between the devices may not be as secure as in a traditional data center.

3. Malware attacks: Malware can be introduced into the system by exploiting vulnerabilities in the devices or through social engineering attacks. Once inside the system, the malware can steal data, delete files, or disrupt the HDFS and the entire Raspberry Pi cluster.

4. Brute-force attacks: A brute-force attack involves trying all possible combinations of passwords until the correct one is found. If the HDFS system has weak passwords, it can be easily compromised through a brute-force attack.

To protect against these types of attacks, it is important to implement strong security measures, such as encrypting communication channels, deploying firewalls, and regularly updating the system with the latest security patches. Additionally, access control mechanisms such as authentication and authorization should be implemented to restrict access to the HDFS system. Finally, it is important to regularly monitor the HDFS system by analyzing its log data for any unusual activity or anomaly and promptly respond to any security incidents.

Log-based anomaly detection is a technique used to detect anomalies or unusual behavior in the log data generated by a system [10]. In the case of HDFS running on an edge cluster like a Raspberry Pi cluster, log-based anomaly detection can be used to monitor the system's log data and identify any unusual patterns that may indicate a security breach or system malfunction. To implement log-based anomaly detection on an HDFS system running on an edge cluster, one could use an open-source tool like Apache NiFi or Apache Flink, which can ingest, process, and analyze log data in real-time [11]. These tools can be deployed on the edge devices and configured to monitor the HDFS system's log data. The log data can be fed into a machine learning model that is trained to identify anomalous behavior. The machine learning model can be periodically retrained on the most recent log data to improve its accuracy and effectiveness.

In addition to log-based anomaly detection, one could ensure secure implementation of a legitimate machine learning application like hand grasp recognition on the same edge cluster. To implement hand grasp recognition on an edge cluster running HDFS, one could use a deep learning framework like TensorFlow or PyTorch, which can be deployed on the edge devices. The deep learning model can be trained on a dataset of labeled dataset of different hand grasp types. Subsequently, the trained model can be deployed on the edge devices and used to classify hand grasp types in real-time. The output of the hand grasp recognition task can be stored in the HDFS system, allowing for further analysis and processing [12]. By running both log-based anomaly detection and a legitimate machine learning application like hand grasp recognition on the same HDFS system running on an edge cluster, one can leverage the power of distributed computing to perform both security monitoring and data processing tasks efficiently. However, it is important to ensure that the resources of the edge devices are allocated appropriately to prevent any performance issues or resource contention.

1.2 Problem Relevance

Large and complex software-intensive systems, including big data systems and online service systems, generate many execution logs for debugging purposes. Failure of their services can have severe consequences, from a poor user experience [13] to significant financial losses [14]. These systems are becoming more vulnerable to faults and vulnerabilities that attackers can exploit to launch attacks, and these attacks are also becoming more sophisticated. Therefore, detecting anomalies in system logs is a critical responsibility in developing a secure and reliable computer system. A log is a message that tracks the execution of the system and contains useful monitoring data such as the verbosity level, event time, and raw message content [15]. Modern computer system frameworks generate vast amounts of log data, often millions of lines per hour, which can reach up to 50 GB in size [16]. Human analysis of such large and complex log datasets can be time-consuming and error-prone, highlighting the need for automated data-driven solutions.

Many automated log-based ways to detecting system abnormalities have been developed throughout the years. These projects extract relevant information from logs and employ data mining and machine learning approaches to analyse log data and discover system irregularities. Despite promising findings on balanced data, they suffer from inadequate number of anomalous examples in real-world datasets [17]. Furthermore, they only identify anomalies that match the previously learned abnormal class, limiting the identification of new anomalies. Another approach named unsupervised outlier identification [18] overcomes the paucity of abnormalities as well as the frequent label unavailability. In contaminated datasets, these clustering approaches separate the nominal samples from anomalous samples. However, traditional machine learning algorithms struggle to extract the temporal information in discrete log data.

Recently, deep learning models, specifically long short-term memory (LSTM) models, have become popular for detecting anomalies in logs due to their ability to handle sequential data. LSTMs are a type of recurrent neural network (RNN) architecture that simulates long-term dependencies more accurately than traditional RNNs. They overcome limitations such as short-term information storage, limited control over the amount of information to be carried forward, and issues of exploding and vanishing gradient problems. LSTMs have been successfully used for a variety of tasks, including machine translation, language modeling, audio and video data analysis, emotion identification, and acoustic model construction [19], [20]. While LSTMs can only use past context, bidirectional long short-term memory (BiLSTM) models can use both past and future contexts by processing incoming data in both forward and backward directions. BiLSTMs have been utilized for text recognition and categorization, among other applications [21]. However, deploying deep learning models like LSTMs and BiLSTMs in resource-constrained environments, such as embedded solutions and edge devices, is a challenging and complex task.

The push for intelligence at the edge has led to a shift towards more in-situ and decentralized analytics in ubiquitous systems. Although deep learning training can occur in the cloud, inference tasks must now be performed at the edge due to factors such as cost, latency, bandwidth, and privacy restrictions [22]. It has been challenging to run advanced deep learning algorithms on devices with limited resources, as these models require significant computational, memory, and energy usage. Specialized hardware accelerator platforms have been developed by companies such as Google and NVIDIA to support deep learning inference at the edge. The Google Edge TPU is an example of such a platform, capable of 4 trillion operations per second while using only 2 watts of electricity [23]. The focus of our work is on the Edge TPU's ability to detect abnormal system execution using log data.

Some popular accelerating techniques for model inference at the edge include designing lightweight DNNs with fewer layers [24] or pruning an existing DNNs by removing parts of the neural network that have minimal impact on the inference process [25]. While these models offer quicker inference and better resource efficiency, they lack the flexibility to be further optimized or adapted to the environment's conditions. Machine learning models are increasingly susceptible to adversarial attacks, where attackers modify input or environmental conditions to undermine the model's accuracy. It's crucial to assess the vulnerabilities of these deep learning models deployed in edge-based scenarios. In our research work, we propose a secure framework for executing real world machine learning applications. Our objective is to simultaneously execute the log-based anomaly detection algorithm and the essential machine learning inference on resource-constraint edge platform. As a case study, we have considered the hand grasp recognition as a real-world machine learning application. The research aims to achieve efficient compression of the security framework while maintaining acceptable accuracy, enabling the essential machine learning application to operate effectively within the same edge platform.

Grasping objects is a crucial activity in human life, as it enables people to move things from one place to another. It includes grasping a pen or grasping a bottle of water, for example. The way humans plan their movements to grasp an object is based on their past experience, called synergy. Grasping is also important in the automation industry for precise pick and place operations, as well as in physiotherapy or rehabilitation. Two things need to be ensured for a proper grasp: tracking the trajectory of grasping and applying appropriate force to the object [26]. A stable grasp means that the object does not slip away, and no excessive force is applied. There are various robotic hand manipulators available for simple to complex grasping tasks, but controlling their multiple degrees of freedom and generating a trajectory for grasping becomes a challenging problem [27].

Accidents and infections can cause some people to lose part of their upper limb. Exoskeleton prosthetic hands (EPH) are made to allow persons who have had their hands amputated keep the look and feel of their natural hands. But in order to create an autonomous system for the amputee, precise control of EPH utilising signals from the human body or mind is necessary [28]. Researchers have investigated the use of electromyogram (EMG) and electroencephalogram (EEG) data in this respect. Electrodes are positioned around the scalp to produce the EEG signal which is used to record the brain's electrical activity. While the EMG records electrical impulses produced by skeletal muscles [29]. In our work, we develop a secure framework for distributed edge devices to recognise hand grasps. A trade-off between performance and flexibility is often necessary for suitable models for edge devices since they need certain topologies and learning methodologies. The two main focus of research in the edge domain are post-training adaptability and deployment topology optimization [30].

The development of edge-efficient models has lately sparked an industry shift towards a distributed computing architecture. In fact, a sizable share of dispersed IoT solutions are edge devices with incorporated deep learning components [31]. The pace of data interchange between edge devices is increasing as network technology evolves rapidly. Due to vulnerabilities in the device's software or firmware, intruders are able to take control of the device and perform distributed denial of service (DDoS) attacks against other targets in addition to stealing the information acquired from the device for use in spam, phishing, and the sale of personal data [32]. Common users would fail to identify the device under attack or control until system or network resources are compromised or network services are momentarily disrupted. Therefore, we intend to increase the security of distributed IoT edge devices and prevent attacks in the future by examining network log and source code vulnerabilities.

1.3 Research Contributions

Our research aims to design a log-based secure framework for machine learning inference on distributed edge clusters. Specifically, the research objectives and contributions of our research work can be stated as follows:

- Develop a secure log-based framework for hand grasp recognition on distributed edge clusters, utilizing a machine learning model that is integrated with a log-based anomaly detection mechanism.
- Conduct a comparative analysis of the proposed secure framework with existing hand grasp recognition methods, including support vector machines and k nearest neighbors, in terms of performance and accuracy.
- Implement an effective mapping of log templates to reduce dimensionality of transformed semantic vectors and address the issue of neglecting semantic similarities among log templates.
- Deploy and execute the proposed secure machine learning framework on commercially available edge devices, leveraging faster computations and minimizing memory consumption.
- Evaluate the performance of the proposed secure machine learning framework, focusing on accuracy, security, and scalability.
- Ensure the privacy and security of user data on distributed edge clusters, preventing unauthorized access and preserving confidentiality.

1.4 Report Organization

The remainder of the report is organized as follows: Chapter-2 presents the related works, while Chapter-3 provides the preliminaries and background of the research work. Next, Chapter-4 describes the proposed secure framework in detail, including the overall system architecture, log-based anomaly detection, and choice of edge platform for deployment. In Chapter-5, we present a case study on machine learning-based hand grasp recognition on the edge and compare its performance with existing algorithms. Next in Chapter-6, we evaluate the performance of the proposed secure framework and investigate its memory reduction, speed-up, and accuracy drop when deployed to an edge platform. Finally, the research work is concluded and a list of future directions is mentioned in Chapter-7.

Chapter 2

Literature Survey

2.1 Log-based Anomaly Detection

Rule-based techniques

Earlier methods for anomaly detection in log data utilized rule-based techniques. For example, Cinque *et al.* [33] presented a new rule-based logging technique for investigation of software anomalies. The proposed method made use of design elements to facilitate the appropriate insertion of logging instructions within the source code of a particular software system. Moreover, Yen *et al.* [34] described a unique approach for effectively mining and obtaining information from raw system logs created by a range of network devices in a big company. Based on behavioural analysis, the suggested approach detected suspicious security issues in hosts. In the same line of research, Yu *et al.* [17] proposed CloudSeer for analyzing log-based workflows in cloud infrastructures. CloudSeer examines concatenated log sequence data for execution issues in an effective and efficient manner. It generates task-automaton instances that represent probable incorrect sequences as indicators for additional investigation. However, the rule-based techniques can only detect previously analysed anomalous cases and need extensive manual engineering.

Unsupervised learning methods

Previous works have also explored unsupervised learning approaches for log anomaly detection. For example, Xu *et al.* [35] suggested utilising the Principal Component Analysis (PCA) algorithm, which assumes that various sessions in a log file may be easily identified by a session-id linked to each log record. Similarly, Lou *et al.* [36] described a comprehensive technique for identifying system anomalies using console log analysis. To mine the linear correlations among log events, the authors used invariant mining and event count vectors. Moreover, Oprea *et al.* [37] presented a combined technique based on k-nearest neighbour classification algorithm and the K-prototype clustering. This method divided datasets into various groups by analyzing system logs and using the K-prototype clustering methodology. Following the filtration of the seemingly normal occurrences which frequently appear as clusters with significant coherence, other events are identified as potential anomalies that necessitate further inquiry. This research, however, makes the unrealistic notion that log abnormalities are brought on by typographical errors in log files.

Deep learning approaches

Recurrent neural networks have shown enhanced performance in time series prediction in recent years, and authors have employed them for log sequence prediction applications. Deep learning approaches' successful implementation has led to a variety of novel approaches for log-based anomaly identification. For instance, Vinayakumar *et al.* [38] employed a stacked-LSTM model to describe log samples of abnormal and normal events because of its ability to instantly learn temporal relationship with sparse representations. The authors created a time-series using log samples of normal and abnormal events that happened in one minute intervals, with the goal of classifying and detecting the occurrences as normal or anomalous. Furthermore, Du *et al.* [39] presented the long short term memory based architecture named DeepLog to represent a system log as a natural language pattern for denial of service attack detection. DeepLog learned log patterns from standard processing and automatically recognised anomalies when log patterns diverged the trained model. The authors also showed how to gradually modify the DeepLog model in an online mode so that it can change over time to accommodate new log patterns. Similarly, Zhang *et al.* [14] automated the creation of feature sequences for multi-source log messages using clustering techniques, and then input those sequences into the LSTM for the early prediction of hardware and software faults. In the same line of research, Daniluk *et al.* [40] presented a neural language model that provided distinct representations for the key and value of a differentiable memory, as well as for storing the next-word distribution. This work also incorporated attention mechanism into LSTM to precisely model complicated log sequences and improve performance accuracy.

NLP-based techniques

Based on the notion that log is a natural language sequence, some researchers have used NLP approaches to evaluate log data. Zhang *et al.* [41] presented LogRobust, a novel log-based anomaly detection technique. To detect abnormalities, they treated a log event as a fixed-dimension semantic vector and used an attention-based Bi-LSTM classification model. Similarly, Meng *et al.* [13] introduced LogAnomaly, a comprehensive log anomaly detection method that employs an unique template2Vec approach to extract the semantic content of log templates and solve the issues encountered by new kinds of logs at runtime, as well as combining logs' sequential and quantitative patterns.

2.2 DL-based Hand Grasp Recognition

Hand grasp recognition is a field of research that aims to develop algorithms and systems capable of accurately identifying and classifying different types of hand grasps. The ability to recognize hand grasps can facilitate the development of more intuitive and natural interfaces for controlling machines or prosthetic devices, as well as help to monitor and diagnose motor disorders. In recent years, there has been significant progress in the development of hand grasp recognition techniques, driven by advances in machine learning and computer vision. For instance, Kim *et al.* [42] presented a technique to recognize hand grasps in real-time by employing two-stage convolutional neural networks (CNNs) that are trained using a weakly supervised dataset. The two-stage CNN architecture first detects the location of the hand and then identifies the grasp type. In a weakly supervised dataset, the grasp types are not specifically labeled in the training data. The authors conducted experiments to demonstrate the accuracy of their proposed approach in recognizing various grasp types. They concluded that their method has potential applications in areas such as human-robot interaction, rehabilitation engineering, and robot grasping. Similarly, Lin *et al.* [43] presented a new method for recognizing hand grasping movements using raw electromyographic signals. The proposed method utilized multidimensional uncertainty-aware models to enhance the robustness of the recognition system for long-term usage. The authors conducted experiments on a dataset consisting of electromyographic signals collected from 10 participants performing various grasp types, and the results showed that the proposed method achieved higher accuracy and robustness compared to other state-of-the-art methods.

The technology utilized in hand grasp recognition has numerous applications in areas such as robotics, human-computer interaction, and healthcare. For example, Zakia *et al.* [44] explored how deep learning techniques can be used to recognize hand grasps by analyzing force myography (FMG) signals. They created a dataset of FMG signals from 15 individuals performing 10 different hand grasps, which they used to train and test their deep learning model. The researchers compared the performance of different deep learning architectures, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs), and found that the CNN-based model had higher accuracy than the RNN-based model. These results suggest that deep learning could be an effective method for identifying hand grasps using FMG signals, with potential applications in prosthetics and human-machine interfaces.

Moreover, Palermo *et al.* [45] conducted a study on the repeatability of grasp recognition for a robotic hand prosthesis using surface electromyography (sEMG) data. The authors conducted experiments on eight healthy subjects to measure the accuracy of the system in recognizing different grasps, such as precision pinch, power grasp, and lateral pinch. They also evaluated the system's repeatability by testing it on three different days. The results showed high accuracy and repeatability, indicating that sEMG-based control of robotic hand prosthesis is a promising approach for improving the functionality of upper limb prostheses. Similarly, Cognolato *et al.* [46] proposed a multimodal approach that combines eye-tracking and computer vision to improve the control of robotic hand prostheses based on the visuomotor behavior of grasping. The study involved an experiment in which participants perform grasping tasks while their eye movements are recorded. The data is then used to develop a model that predicts the intent of the user and controls the prosthesis accordingly.

The results demonstrated that the proposed approach can improve the accuracy and speed of prosthesis control and enhance the user's overall experience.

2.3 Deep Learning Inference on Edge Clusters

With the increasing number of edge devices in various applications such as autonomous vehicles, drones, and IoT devices, the need for efficient and fast inference on these devices has become crucial. In recent years, there has been significant progress in the development of novel techniques for deploying deep neural network models on edge clusters, utilizing distributed computing and parallel processing. For instance, Suzen *et al.* [47] presented a benchmark analysis of the NVIDIA Jetson TX2, Jetson Nano, and Raspberry Pi using a deep convolution neural network (CNN). The performance of the three devices was evaluated using the VGG16 model, and the accuracy and processing time for image classification were measured. The results showed that the Jetson TX2 has the best performance, followed by the Jetson Nano and the Raspberry Pi. The study provided insights for selecting the appropriate embedded device for applications that require image recognition and processing.

Furthermore, Srinivasan *et al.* [7] proposes a novel approach to implement mobile Raspberry Pi Hadoop clusters that can perform robust and augmented computing. The authors discuss the benefits and limitations of using mobile clusters, and propose a customized architecture for mobile Raspberry Pi Hadoop clusters that can operate in a low-bandwidth environment. The paper also describes the performance evaluation of the proposed approach, demonstrating that the mobile Raspberry Pi Hadoop clusters can provide significant improvement in processing performance compared to single-node systems. Similarly, Komninos *et al.* [48] explored the feasibility of using low-cost Raspberry Pi micro-clusters for edge machine learning in the tourism industry. The study examined the performance of the micro-clusters in terms of accuracy and execution time for various machine learning algorithms, including decision trees and neural networks. The authors found that the Raspberry Pi micro-clusters perform well for edge machine learning tasks in tourism, demonstrating accuracy rates of up to 96% and execution times of a few seconds.

Chapter 3

Background Study

3.1 Single-board Edge Computers

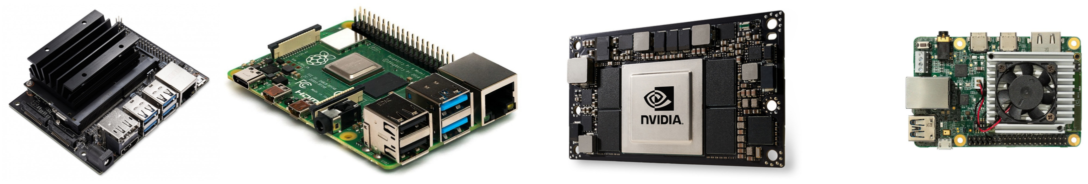
Figure 3.1: Latest Edge Devices in the Market.

Single-board edge computers are compact, low-cost computing devices that are designed to perform a variety of edge computing tasks. They typically feature a small form factor, low power consumption, and are equipped with a range of input/output (I/O) interfaces. Single-board edge computers are particularly useful in situations where traditional computing devices are impractical, such as in IoT deployments, robotics, and automation. In recent years, several single-board edge computers have been developed and released to the market, including the Raspberry Pi 4, Google Coral Board, Jetson Nano, and Jetson TX2 as have been shown in Figure-3.1.

Raspberry Pi 4 is a popular single-board edge computer that is widely used in various applications, including home automation, robotics, and education. It features a Broadcom BCM2711 quad-core Cortex-A72 (ARM v8) 64-bit SoC with a clock speed of 1.5 GHz, up to 8GB of RAM, and a range of I/O interfaces, including USB 3.0, Ethernet, and HDMI. Whereas, the Google Coral Board is a single-board edge computer that is designed for running machine learning models at the edge. It features a Google Edge TPU coprocessor for accelerating machine learning tasks and is powered by a NXP i.MX 8M SoC with a quad-core ARM Cortex-A53 CPU. The board also features Wi-Fi, Bluetooth, and a range of I/O interfaces.

Developed by NVIDIA, the Jetson Nano is a low-cost, small form factor single-board edge computer that is designed for running AI workloads at the edge. It features a quad-core ARM Cortex-A57 CPU, a 128-core NVIDIA Maxwell GPU, and 4GB of LPDDR4 memory. The board also features Gigabit Ethernet, USB 3.0, and a range of I/O interfaces. Another board by Nvidia, the Jetson TX2 is a high-performance single-board edge computer that is designed for running complex AI workloads at the edge. It features a NVIDIA Pascal GPU with 256 CUDA cores, a dual-core Denver 2 CPU, and a quad-core ARM Cortex-A57 CPU. The board also features Gigabit Ethernet, USB 3.0, and a range of I/O interfaces. The Jetson TX2 is commonly used in applications such as autonomous vehicles, robotics, and drones.

3.2 Edge Compression Techniques

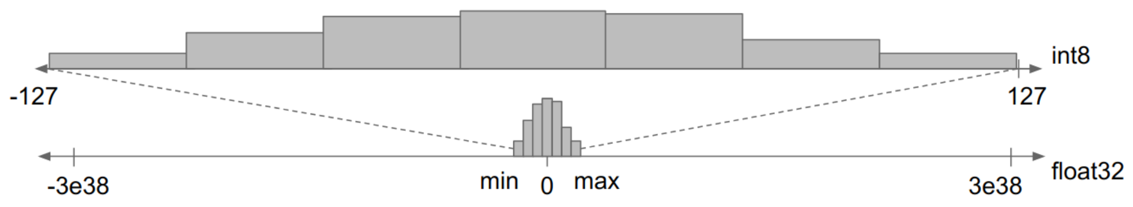

Figure 3.2: Mapping of 32-bit float values to 8-bit integers.

Edge deployment of deep learning models requires the models to be compressed in order to optimize their performance on resource-constrained devices. Compression techniques such as pruning, quantization, and distillation have been proposed to reduce the size of the models while maintaining their accuracy. Among these techniques, full-integer quantization has gained a lot of attention due to its effectiveness in reducing the model size and increasing inference speed.

Full-integer quantization is a technique that involves representing the weights and activations of a deep learning model with integers, typically 8-bit integers. This is in contrast to floating-point representations, which use more bits to represent numbers with fractional parts. Full-integer quantization has the advantage of being more memory-efficient, which is crucial for edge devices with limited memory. In order to perform full-integer quantization, the weights and activations of a model need to be quantized to integers. This is typically done by first scaling the weights and activations to a range that can be represented by integers as demonstrated in Figure-3.2. The scaling factor is then quantized to an integer power of 2, which simplifies the multiplication operation during inference. The quantized values are then stored as integers. The benefits of full-integer quantization include: 1) 4x reduction in model parameters, 2) 1.5x faster execution for convolution models, 3) 2-4x faster on fully-connected & RNN-based models, and 4) Enables execution on ML accelerators.

Full-integer quantization can be combined with other compression techniques such as pruning to further reduce the size of the model. Pruning involves removing the weights with small magnitudes, which results in a sparse model. The sparse model can then be quantized using full-integer quantization to reduce its size even further. One challenge with full-integer quantization is that it can result in a loss of accuracy compared to floating-point representations. However, recent research has shown that full-integer quantization can be performed with minimal loss of accuracy by using techniques such as symmetric quantization and per-channel quantization. Symmetric quantization involves quantizing the weights and activations symmetrically around zero, while per-channel quantization involves quantizing each channel of a tensor separately. These techniques can help to preserve the accuracy of the model while still achieving significant compression.

3.3 Embedded System of the Exohand

The ExoHand is a structure resembling the human hand with potentiometers attached to each finger holder and one for capturing the rotational motion of the thumb. When the fingers are moved, the potentiometer shafts rotate thereby resulting in the change in output voltage. These voltage signals are captured by the microcontroller using its ADC. The microcontroller can then generate PWM signals to control the position of the servomotor. The servomotor is a part of the robotic hand whose fingers are to be controlled. The shaft of the servomotor is then connected to the meta-carpal joint of each finger. Inertial Measurement Units (IMUs) are connected to each finger

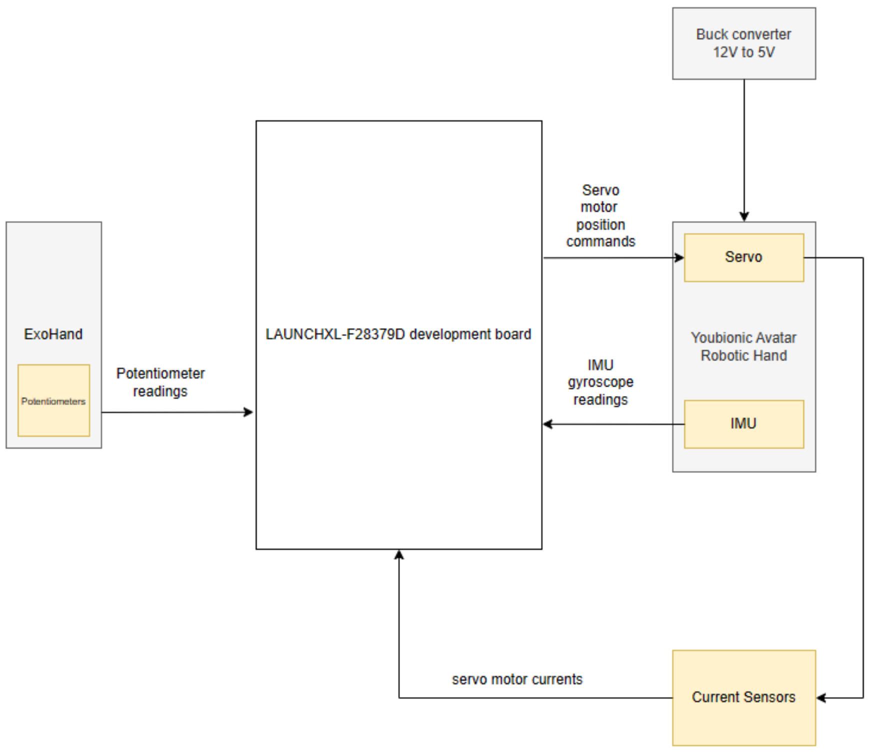

Figure 3.3: Block Diagram of the Exohand System.

to measure the rate of change of angular position using its gyroscopes. The IMUs communicate with the microcontroller through the SPI interface. The servomotors are powered from a 12V wall adapter through a buck converter to reduce voltage level from 12V to 5V. The current through each servomotor is measured by using an OP-AMP based circuit and then fed to the ADC of the microcontroller. The servomotor current is proportional to the torque of the servomotor by means of which we can measure the force exerted by each finger on the object to be grasped. The overall block diagram of the system is shown in Figure-3.3.

3.4 Edge Clusters for Distributed Computing

Edge computing is an emerging paradigm that leverages resources on the edge of the network to perform computations closer to where the data is generated. This approach has become increasingly popular due to the proliferation of IoT devices and the need for real-time processing of data. In recent years, edge clusters have emerged as a promising solution for distributed computing in edge environments. An edge cluster is a set of interconnected computing nodes located at the edge of the network that can work together to execute tasks. These clusters are typically composed of low-cost, energy-efficient computing devices, such as Google Edge TPUs, Raspberry Pis, that can be deployed in large numbers. By pooling these resources, edge clusters provide a scalable and cost-effective solution for distributed computing in edge environments.

Practical application of edge clusters are in the field of blockchains, weather forecasting, smart cities, and industrial internet of things. A edge cluster based blockchain can be used to create a decentralized and distributed ledger system that can store and validate transactions without the need for a central authority. In a blockchain network, transactions are verified and recorded in blocks that are linked together in a chronological chain, creating an immutable record of all transactions. Furthermore, a weather forecasting system built on a distributed edge cluster can be used to process and analyze weather data in real-time, providing more accurate and localized weather forecasts. With a distributed cluster, weather data can be collected from multiple sensors and sources, processed in parallel, and analyzed using machine learning algorithms.

As cities become more connected and intelligent, they generate large volumes of data that need to be processed in real-time to support various applications, such as traffic management, public safety, and environmental monitoring. Edge clusters can be used to deploy distributed computing resources throughout the city to support these applications. For example, a network of edge clusters can be used to process sensor data from traffic cameras and provide real-time traffic updates to drivers. Another application of edge clusters is in the industrial sector. In manufacturing and logistics, there is a growing need for real-time data analysis to optimize operations and improve efficiency. Edge clusters can be used to deploy computing resources closer to the production line or warehouse, enabling real-time analysis of sensor data and machine logs. This can help identify bottlenecks, optimize workflows, and reduce downtime.

3.5 Security Issues and Attacks on Edge Clusters

In recent years, edge computing has emerged as a promising approach to provide computational power and storage capabilities in proximity to the data source, reducing latency and improving efficiency. However, as edge clusters become more widely adopted, they also become a prime target for cyber attacks. In particular, the Hadoop framework, which is commonly used in edge clusters, has been found to have several vulnerabilities that make it susceptible to attacks. One of the main security issues with Hadoop is the lack of authentication and authorization mechanisms. By default, Hadoop does not authenticate users, which means that anyone who can access the Hadoop cluster can execute arbitrary code or access sensitive data. Additionally, Hadoop also does not provide fine-grained access control, which means that a user with access to the cluster can potentially view or modify data that they are not authorized to access.

Another security issue with Hadoop is related to the use of unsecured protocols for communication. Hadoop uses a number of protocols such as HDFS, MapReduce, and YARN, all of which communicate over unsecured channels. This means that an attacker who can intercept the network traffic can eavesdrop on sensitive information, inject malicious code, or launch denial-of-service attacks. Furthermore, Hadoop also has several vulnerabilities that make it susceptible to attacks such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF). These attacks can be used to gain unauthorized access to data, steal sensitive information, or execute arbitrary code on the cluster.

To mitigate these security issues, several best practices can be followed when deploying Hadoop in edge clusters. These include implementing strong authentication and authorization mechanisms, using secure communication protocols such as SSL/TLS, and regularly applying security patches and updates. Additionally, deploying intrusion detection and prevention systems, network log-based anomaly detection framework, and conducting source code vulnerability checks can also help prevent security breaches and protect the integrity of the data stored in the cluster.

Chapter 4

Proposed Secure Framework

4.1 System Architecture

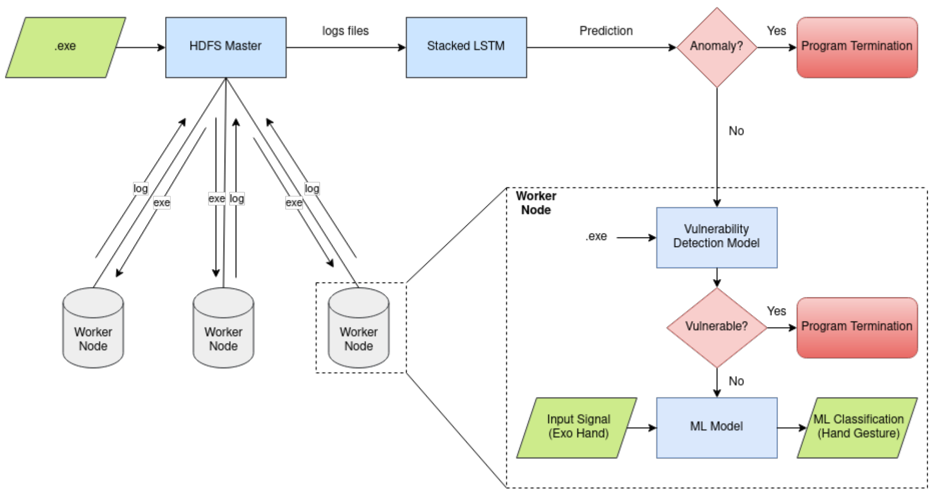

Figure 4.1: System Architecture of Secure Framework for Machine Learning Inference.

Here we highlight and discuss in detail the step-by-step process employed by our proposed secure system to identify anomalies and vulnerabilities in an executable (.exe) file and simultaneously predict the hand grasp type using the Exohand input data.

- Upon receiving the executable file, the Hadoop Distributed File System (HDFS) master node assesses the availability of worker nodes in the cluster. This assessment ensures that the file is efficiently distributed among the worker nodes, enabling parallel processing and optimal resource utilization.

- Within the worker nodes, various file operations, such as reading, writing, and modifying, are executed on the .exe file. These operations generate a series of log data that provide insights into the file's behavior. The master node subsequently collects this log data from the worker nodes for further analysis.

- The master node employs a stacked-LSTM model, a deep learning architecture specifically designed to handle sequential data, to analyze the collected log data. By providing the log files as input, the model predicts the presence of any anomalies in the executable file, such as unexpected patterns or deviations from normal behavior.

- If an anomaly is detected, the program's execution is immediately terminated to prevent any potential security threats. If no anomaly is found, the process proceeds to the next step.

- In the absence of anomalies, the executable file undergoes a conversion process that yields its corresponding source code. This step is crucial for the subsequent vulnerability analysis, as it allows the system to examine the underlying code in its original form.

- To identify potential vulnerabilities in the source code, a deep learning-based vulnerability detection model is utilized. This model has been trained on a vast dataset of known vulnerabilities, allowing it to effectively recognize patterns and indicators of security flaws within the code.

- If a vulnerability is detected, the program's execution is terminated to prevent the exploitation of the security flaw. The developers are then notified to provide a security patch addressing the issue, ensuring the safety and integrity of the system.

- If the source code is deemed to be free of vulnerabilities, it is treated as the required model. This model processes the Exohand input data, which consists of sensor readings capturing hand movements and gestures, to predict the hand grasp type. The prediction enables the system to better understand and interpret the user's intentions, ultimately enhancing its usability and effectiveness.

By following this comprehensive workflow as demonstrated in Figure-4.1, the system ensures that the executable file is thoroughly examined for anomalies and vulnerabilities, safeguarding the system's security and integrity while ultimately predicting the hand grasp type using the Exohand input data.

4.2 Log-based Anomaly Detection

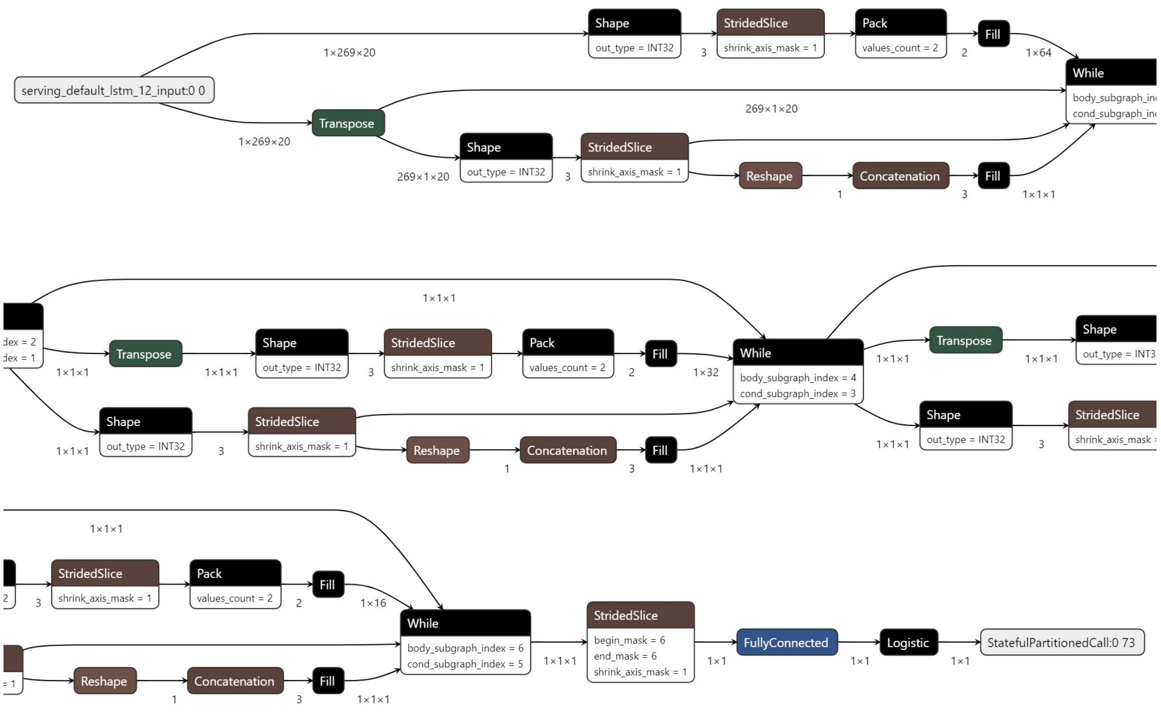

Figure 4.2: Log-based Anomaly Detection Model Architecture

Log-based anomaly detection using stacked Long Short-Term Memory (LSTM) is a popular method for detecting anomalies in system log data. This method uses a deep learning approach to analyze log data, which otherwise can be challenging to analyze manually due to the sheer volume of log data generated by modern systems. In our approach, the raw log data is first parsed to abstract away the message's attributes and extract its log event. Next step involves tokenizing the log messages, converting them to a sequence of integers and then one-hot encoding the messages using the embedding vectors. The preprocessed log data is then used as input to the stacked LSTM network. The stacked LSTM network is a type of recurrent neural network (RNN) that is designed to learn long-term dependencies in sequential data.

It is called "Stacked" because it consists of multiple layers of LSTM cells, each layer processing the input sequence and passing its output to the next layer. The architecture of the stacked LSTM network typically includes an input layer, one or more LSTM layers, and an output layer. The input layer receives the preprocessed log data and passes it to the first LSTM layer. Each LSTM layer consists of a set of memory cells and gates that control the flow of information through the network.

The output layer of the Stacked-LSTM network is designed to classify the input sequence as either normal or anomalous. During training, the network is fed a set of labeled log data and learns to recognize patterns in the normal data. Once Stacked-LSTM has been trained, the network can be used to classify new log data as either normal or anomalous based on how closely it resembles the learned log patterns. The performance of the stacked LSTM network can be improved by tuning various hyperparameters, such as the number of LSTM layers, the number of cells in each layer, the learning rate, and the batch size. Additionally, techniques such as dropout and early stopping can be used to prevent overfitting and improve generalization. The architecture of our Stacked-LSTM model has been illustrated in Figure-4.2. It consists of three layers with 64, 32 and 16 memory blocks, respectively. The LSTM layers are followed by a Dense layer with 1 neuron which is responsible for predicting the log pattern type. Binary cross entropy loss function is used in the model training process. Furthermore, the adaptive gradient descent method Adam is used for the optimization of the loss function during training and logistic sigmoid is used as the non-linear activation function in the output layer. The Stacked-LSTM model obtains an accuracy of 98.73% when trained for 15 epochs on the Hadoop Distributed File System (HDFS) log dataset.

The HDFS log dataset [18] is a collection of log files generated by the worker nodes of the Apache Hadoop distributed computing framework. HDFS is a distributed file system that is designed to store and manage large amounts of data across a cluster of computers. The HDFS log dataset contains information about the operation of the HDFS system, including metadata operations, data operations, and other system events. The log files contain timestamps, source IP addresses, event types, and other relevant information that can be used to monitor and diagnose the performance of the HDFS system. The HDFS log dataset is typically stored in text format, with each log entry separated by a newline character. The log files may be compressed to save disk space, but they are usually processed in their uncompressed form to facilitate analysis. Furthermore, the HDFS log dataset can be used for a variety of purposes, including monitoring system performance, diagnosing issues and errors, and analyzing user behavior. It can also be used to train machine learning models for anomaly detection or predictive maintenance. However, the HDFS log dataset can be quite large and complex, making it difficult to work with without the proper tools and learning techniques. Hence, we propose the Stacked-LSTM model to effectively learn and extract meaningful insights from the dataset.

4.3 Compressed Security Model

Post-training quantization is a technique used to reduce the size of a trained machine learning model and speed up its inference time by converting the numerical representation of the model's parameters from floating-point numbers (32-bit) to lower precision integers (8-bit). This conversion reduces the memory and computation costs without significantly impacting the model's performance. This process can be applied to various types of models, such as Stacked Long Short-Term Memory (Stacked-LSTM) and Deep Neural Networks (DNN). It is especially important for deploying models on resource-constrained devices like mobile phones, IoT devices, or embedded systems, where the available memory and processing power are limited. In post-training quantization, the weights, biases, and activations of the Stacked-LSTM and DNN models can be quantized to reduce their memory footprint and computational requirements. There are two common approaches to post-training quantization: Hybrid quantization and Full-integer quantization.

In case of Hybrid quantization, the weights of the machine learning model are changed from 32 bit float to 8 bit integer. Whereas, the biases and activation functions are kept as 32 bit float. Here, the computations are performed using both integer and floating point values. For the transformation, no data is required and the quantized model only incurs a small accuracy loss. The benefits of hybrid quantization include: 1) 4x reduction in model parameters, 2) 10-50% faster execution for convolution models, and 3) 2-3x faster on fully-connected & RNN-based models. While on the other hand, Full-integer quantization is a more advanced quantization technique that uses integer-only arithmetic to represent the parameters and activations. The weights of the machine learning model are changed from 32 bit float to

8 bit integer. Here, the computations are performed using only integer values. For the transformation, the conversion method requires unlabeled data and the quantized model incurs a small accuracy loss. Full-integer quantization technique is considered to have the fastest latency. The benefits of full-integer quantization include: 1) 4x reduction in model parameters, 2) 1.5x faster execution for convolution models, 3) 2-4x faster on fully-connected & RNN-based models, and 4) Enables execution on ML accelerators. During machine learning inference on the edge, we employ 8-bit integer-only arithmetic to perform the Stacked-LSTM and DNN computations, which can significantly reduce the model's memory and computation requirements.

4.4 Choice of Edge Platform

Edge Devices	**Performance**	**CPU**	**GPU**	**Memory**	**Storage**	**Power**	**Price**	**Feasibility?**
Raspberry Pi4	13.5 GFLOPS	Quad-core ARM Cortex-A72 64-bit @ 1.5 GHz	Broadcom Video Core VI (32-bit)	8 GB	Micro-SD	2.56W-7.30W	30$	✓
Jetson Nano	472 GFLOPS	Quad-Core ARM Cortex-A57 64-bit @ 1.42 GHz	NVIDIA Maxwell w/128 CUDA cores @ 921 MHz	4 GB	16 GB eMMC	5W-10W	89$	✓
Jetson TX2	1.3 TFLOPS	Quad-Core ARM Cortex-A57 @ 2GHz + Dual-Core NVIDIA Denver2 @ 2GHz	NVIDIA Pascal 256 CUDA cores @ 1300MHz	8 GB	32GB eMMC	7.5W-15W	399$	✓
Google Coral Board	4 TFLOPS	Quad-Core Arm Cortex-A53 processor @ 1.5 GHz	Edge TPU	1GB	8GB eMMC	3.5W	150$	✓

Table 4.1: Technical Specifications of Edge Devices

Table-4.1 presents the technical specifications of different commercially available edge platforms. In our work, the proposed security framework is deployed to the Google Coral Board after full-integer quantization. Google Coral Board is a specialized Edge TPU (Tensor Processing Unit) designed for running machine learning models at the edge, specifically for inferencing tasks. Google released the Edge TPU in 2019, which is a purpose-built ASIC hardware accelerator for machine learning applications with high speed and a low energy footprint, with the goal of conducting machine learning inference at the edge. The Coral Development Board is a single-board computer that includes a detachable system-on-module (SOM) with eMMC, SOC, wireless radios, and the Edge TPU. The Edge TPU is built on a cyclic array design, which enables for very efficient matrix multiplication on a huge scale. While

Raspberry Pi, Jetson Nano, and Jetson TX2 are all versatile single-board computers (SBCs) that can be used for various purposes, including Artificial Intelligence (AI) and Machine Learning, the Google Coral Board stands out for running secure deep learning inference models due to several factors:

- **Dedicated Edge TPU:** The Coral Board's primary advantage is its built-in Edge TPU, which is specifically designed for accelerating TensorFlow Lite models. This ASIC (Application-Specific Integrated Circuit) provides high performance and energy-efficient inferencing. In contrast, Raspberry Pi, Jetson Nano, and Jetson TX2 rely on general-purpose CPUs and GPUs for inferencing, which might not be as optimized for this specific task.
- **Security Features:** The Coral Board includes secure boot and an onboard crypto-coprocessor, which ensures the integrity and authenticity of the software running on the device. These security features protect against unauthorized access, tampering, and other security threats, making the Coral Board a more secure choice for deploying deep learning models in sensitive environments.
- **TensorFlow Lite Support:** The Coral Board is optimized to work with TensorFlow Lite, a lightweight version of TensorFlow designed for edge devices. TensorFlow Lite allows developers to easily convert their existing TensorFlow models into a format compatible with the Edge TPU, thus simplifying the deployment process. While other devices like Raspberry Pi, Jetson Nano, and Jetson TX2 can also run TensorFlow Lite models, they lack the dedicated hardware acceleration provided by the Coral Board.
- **Power Efficiency:** The Coral Board is designed for low-power operation, making it suitable for deploying deep learning models in power-constrained environments, like battery-powered or solar-powered IoT devices. The Edge TPU's power efficiency is superior to general-purpose CPUs and GPUs found in Raspberry Pi, Jetson Nano, and Jetson TX2.
- **Scalability:** The Coral ecosystem provides various options for scaling up or down based on the requirements of the project, such as the Coral USB Accelerator, Coral Dev Board Mini, and Coral System-on-Module. This flexibility allows developers to choose the right solution for their specific use case, while still benefiting from the advantages of the Edge TPU.

- **Ease of Integration:** Google provides extensive documentation and software libraries for the Coral Board, making it easier for developers to integrate deep learning models into their projects. Additionally, Google's extensive ecosystem of tools, such as AutoML Edge, simplifies the process of training and deploying custom models on Coral devices.

While the Google Coral Board might not be the ideal choice for every AI project, its specialized hardware, security features, power efficiency, and TensorFlow Lite support make it a preferred choice for running secure deep learning inference models in edge devices. Additionally, we evaluate the practicality of our secure framework on four distinct edge platforms. We note that utilizing full-integer quantization enables maximal compression of the security model, conserving resources for the legitimate machine learning application's execution. However, further diminishing the weights' precision results in a decrease in accuracy from 98.7% to 86.47%. Consequently, we carefully select the compression method to optimize memory usage while simultaneously achieving an accuracy level suitable for real-world applications.

Chapter 5

ML Case Study: Hand Grasp Recognition

5.1 Feature Extraction

Hand grasp recognition is an essential task in the field of human-robot interaction, rehabilitation, and prosthetic design. It involves identifying the hand gestures made by an individual, such as grasping or releasing an object, based on the signals generated by the muscles in the hand. To accomplish this, researchers use feature extraction techniques to transform the raw electromyography (EMG) signals into a set of relevant features that can be used for classification. Feature extraction is the process of selecting and transforming raw data into a set of meaningful features that can be used for analysis or machine learning tasks. In the context of hand grasp recognition, feature extraction involves selecting a subset of the EMG signals and transforming them into a set of features that are relevant for identifying different hand gestures.

The exohand dataset is a collection of EMG signals recorded from six different potentiometers attached to the exohand performing three different hand grasp tasks. The dataset can be cast as a multivariable time series classification problem by treating the EMG signals from each muscle as a separate time series and the hand grasp object as the class label. This approach allows for the use of deep learning techniques such as recurrent neural networks (RNNs) or long short-term memory (LSTMs) to model the temporal dependencies between the different EMG signals and classify the hand grasp task. To apply RNNs or LSTMs to the exohand dataset, the EMG signals can be preprocessed by applying bandpass filters to remove noise and then segmenting the signals into fixed-length windows. The resulting windows can then be used as input to the RNN or LSTM, which can be trained using techniques such as backpropagation through time to learn the temporal dependencies between the different EMG signals and classify the hand grasp task.

Time-domain features are typically straightforward to implement since they do not require any transformation and can be computed directly from the raw exohand time series data. Consequently, these features have found widespread use in both medical and engineering research and applications. We consider 14 time-domain features for hand grasp recognition namely, Root mean square, Mean absolute value, Variance, Log detector, Average amplitude change, Difference absolute standard deviation value, Kurtosis, Mean absolute difference, Slope, Peak to peak distance, Shannon entropy, Skewness, Median, Interquartile range. These 14 features are then extracted from six individual exohand time series to form a multivariable dataset consisting of 70 (14×5) features. Finally, it is used to train the machine learning and deep learning models to recognize different hand grasps. The 14 different time-domain based features for hand grasp recognition are described as follows:

- **Root mean square (RMS):** The RMS is a measure of the magnitude of the EMG signal and is calculated by taking the square root of the mean of the squared values of the EMG signal.
- **Mean absolute value (MAV):** The MAV is a measure of the average amplitude of the EMG signal and is calculated by taking the mean of the absolute values of the EMG signal.
- **Variance:** The variance is a measure of the spread of the EMG signal and is calculated by taking the average of the squared differences between each value and the mean of the EMG signal.
- **Log detector:** The log detector is a feature that computes the logarithm of the RMS value of the EMG signal. It is useful in detecting low-level muscle activity in the hand.
- **Average amplitude change (AAC):** The AAC is a measure of the rate of change of the EMG signal and is calculated by taking the average of the absolute

differences between consecutive values of the EMG signal.

- **Difference absolute standard deviation value (DASDV):** The DASDV is a measure of the variability of the EMG signal and is calculated by taking the absolute differences between consecutive values of the EMG signal and then computing the standard deviation.
- **Kurtosis:** The kurtosis is a measure of the peakedness of the distribution of the EMG signal and is calculated by comparing the fourth central moment of the distribution with the square of the variance.
- **Mean absolute difference (MAD):** The MAD is a measure of the average absolute difference between consecutive values of the EMG signal.
- **Slope:** The slope is a measure of the rate of change of the EMG signal and is calculated by fitting a line to the EMG signal using linear regression.
- **Peak to peak distance:** The peak to peak distance is a measure of the distance between the highest and lowest values of the EMG signal.
- **Shannon entropy:** The Shannon entropy is a measure of the information content of the EMG signal and is calculated by taking the negative sum of the probability of each value multiplied by the logarithm of that probability.
- **Skewness:** The skewness is a measure of the asymmetry of the distribution of the EMG signal and is calculated by comparing the third central moment of the distribution with the cube of the variance.
- **Median:** The median is a measure of the central tendency of the EMG signal and is the value that divides the distribution into two halves.
- **Interquartile range (IQR):** The IQR is a measure of the spread of the EMG signal and is the difference between the upper and lower quartiles of the distribution.

In general, time-domain based features provide useful information about the EMG signal that can be used for hand grasp recognition. These features capture different aspects of the EMG signal, such as amplitude, variability, and rate of change, and can be used in combination to improve robustess and classification accuracy.

5.2 Deep Learning Model Architecture

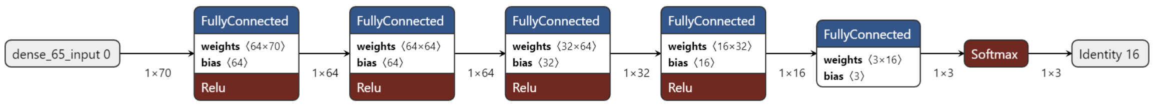

Figure 5.1: Hand Grasp Recognition Model Architecture

Hand grasp recognition is an essential aspect of human-computer interaction, robotics, and assistive technologies. Several machine learning models have been developed for hand grasp recognition, including k-Nearest Neighbour (k-NN) with Dynamic Time Warping (DTW) and Support Vector Machine (SVM). K-Nearest Neighbour (k-NN) is a non-parametric, supervised machine learning algorithm that classifies instances based on their proximity to other instances in the feature space. For hand grasp recognition, the algorithm identifies the k nearest hand grasps examples in the training dataset and assigns the input instance to the class with the highest frequency among the k nearest neighbors. Dynamic Time Warping (DTW) is a time series alignment technique that calculates the optimal alignment between two time series by minimizing the distance between them.

In the context of hand grasp recognition, DTW is used to find the optimal alignment between input gesture sequences and the sequences in the training dataset. The combination of k-NN with DTW allows the algorithm to handle varying speeds and duration of hand grasps making it more robust and flexible. Support Vector Machine (SVM) is a supervised learning algorithm that classifies instances by finding the hyperplane that best separates the different classes in the feature space. In hand grasp recognition, SVM is used to classify hand grasps based on features extracted from the hand movements, such as joint angles or fingertip positions. SVM is particularly effective when the feature space is high-dimensional, and it can handle non-linear relationships between features using kernel functions. It is also known for its generalization capabilities, making it a popular choice for hand grasp recognition tasks.

Deep neural networks (DNNs) are a type of artificial neural network with multiple hidden layers that can learn complex patterns and features from raw input data. DNNs have several advantages over traditional machine learning approaches. DNNs can automatically learn hierarchical feature representations from raw data, eliminating the need for manual feature engineering, which can be time-consuming and error-prone. DNNs are also more robust to variations in input data, such as noise, occlusion, and changes in viewpoint, making them better suited for real-world applications. Furthermore, DNNs can efficiently handle large-scale datasets and high-dimensional input data, allowing them to learn complex patterns and relationships in the data. DNNs can be trained end-to-end, directly mapping raw input data to the desired output, simplifying the overall learning pipeline. Deep learning has been used extensively for hand grasp recognition, and several approaches have been proposed to achieve high accuracy and efficiency. In our work, we employ a DNN with 4 hidden layers with 64, 64, 32, and 16 neurons, respectively, as demonstrated in Figure-5.1.

5.3 Deep Learning Inference at the Edge

Deep learning inference at the edge refers to the process of deploying trained deep learning models on edge devices such as smartphones, IoT devices, or embedded systems to perform real-time computations. This approach offers several advantages, including lower latency, reduced dependence on the cloud, better privacy, and lower power consumption. In this case study, we will discuss hand grasp recognition using deep learning models and deploying these models on a Google Coral board using TensorFlow Lite (TFLite). Hand grasp recognition is a critical component of various applications such as robotics, human-computer interaction, and assistive devices for people with disabilities. Once the deep learning model for hand grasp recognition has been trained, it needs to be optimized for deployment on edge devices. This involves reducing the model's size and computational requirements without significantly compromising its accuracy. We utilize the full-integer quantization to achieve this model compression.

The Google Coral board is an edge TPU (Tensor Processing Unit) designed for efficient on-device machine learning inference. To deploy a deep learning model on the Coral board, it needs to be converted into a TFLite format. First, we install the Edge TPU Compiler and TensorFlow Lite libraries on your development machine. Next, convert the trained TensorFlow model to a TFLite model using the TFLite Converter. This process involves specifying the input/output tensors, quantization mode (int8), and optimization settings. Compile the TFLite model for the Edge TPU using the Edge TPU Compiler. This step produces a TFLite model optimized for the

Coral board's hardware. Deploy the compiled TFLite model on the Coral board by integrating it into an application. The application captures input signals, preprocesses them, and feeds them into the model for inference. The Coral board's TPU accelerates the inference process, yielding real-time hand grasp recognition results.

Deploying deep learning models for hand grasp recognition on edge devices like the Google Coral board offers numerous advantages in terms of latency, privacy, and power consumption. By leveraging TFLite and the Coral board's TPU, developers can create efficient and responsive hand grasp recognition systems suitable for various applications, including robotics, human-computer interaction, and assistive devices.

Chapter 6

Experimental Results

6.1 Stacked-LSTM on HDFS Log Dataset

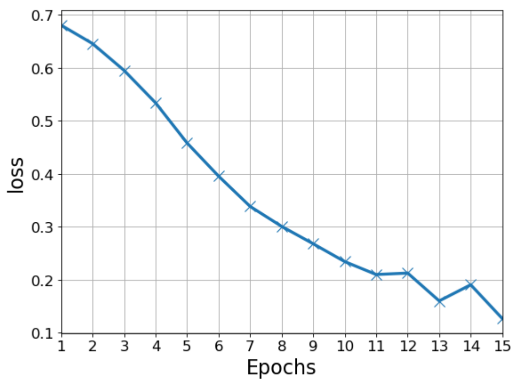

Figure 6.1: Training Loss.

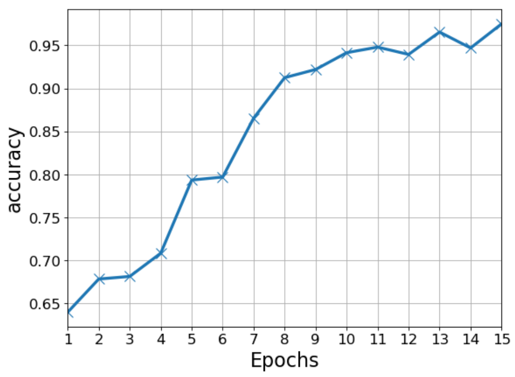

Figure 6.2: Training Accuracy.

In this section, we discuss the performance of the Stacked-LSTM model trained on the HDFS log dataset, as illustrated by Figure-6.1 and Figure-6.2. This dataset comprises 12,000 datapoints, with an equal distribution of normal and anomalous events. It is generated by running Hadoop-based map-reduce processes on over 200 Amazon EC2 nodes and annotated by Hadoop experts. The HDFS log dataset contains 24,396,061 log messages from 29 log events. We create the training set by randomly selecting 6,000 normal and 6,000 anomalous log sequences from the source dataset. Log sequences are organized in HDFS based on the session ID in each message, with an average length of 19 characters. The Stacked-LSTM model, known for its ability to model complex temporal dependencies, achieves an accuracy of 97.075%

and an RMSE loss of 0.1356 when trained on the HDFS log dataset. This makes it highly suitable for analyzing and predicting complex sequential data such as HDFS logs. The model's high accuracy and low loss scores highlight its effectiveness in detecting anomalies in log data.

6.2 Performance on Hand Grasp Classification

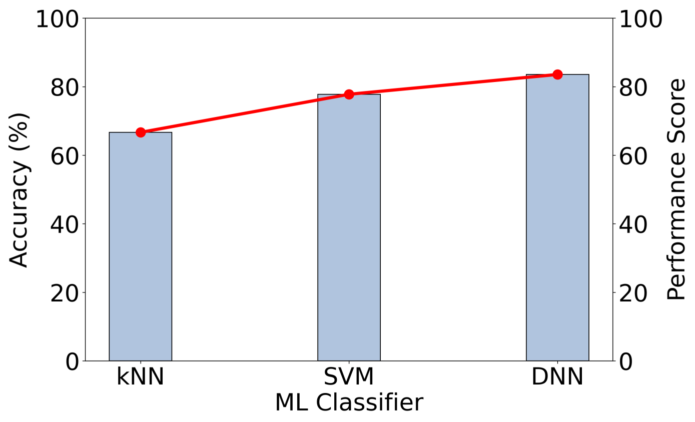

Figure 6.3: Performance Comparison

In this section, Figure-6.3 presents the performance comparison of k-Nearest Neighbors (kNN) with Dynamic Time Warping (DTW), Support Vector Machine (SVM), and Deep Neural Network (DNN) on the Exohand dataset for the hand grasp recognition task. The evaluation metric used for the comparison is accuracy, which is defined as the proportion of correctly classified instances out of the total instances in the dataset. The kNN with DTW algorithm achieved an accuracy of 66.67% on the Exohand dataset. This result indicates that, although the kNN with DTW approach can be effective for certain pattern recognition tasks, its performance on the Exohand dataset for hand grasp recognition is relatively limited. The primary reason for this lower accuracy might be the high variability and complexity of hand movement patterns, which might not be effectively captured by the kNN with DTW algorithm.

The SVM classifier displayed an improved performance compared to the kNN with DTW algorithm, obtaining an accuracy of 77.78% on the Exohand dataset. This result highlights the ability of the SVM model to generalize better to the hand grasp recognition task. The higher accuracy can be attributed to the kernel-based learning approach used by SVM, which allows it to separate the data points effectively in a higher-dimensional space. The best performance among the compared algorithms was achieved by the DNN model, which attained an accuracy of 83.56% on the Exohand dataset. This superior performance can be attributed to the deep architecture of the neural network, which is capable of capturing complex patterns and representations from the data. The DNN's ability to learn hierarchical feature representations contributes to its strong performance on the hand grasp recognition task.

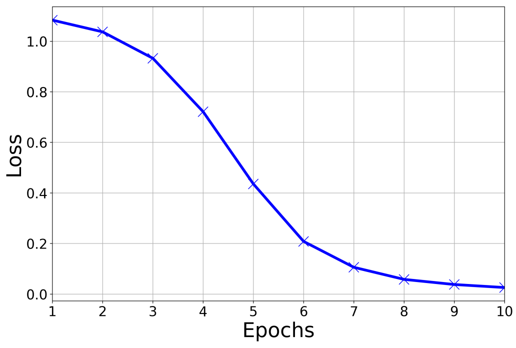

Figure 6.4: DNN Training Loss.

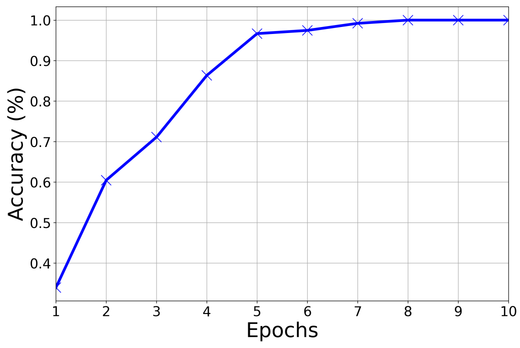

Figure 6.5: DNN Training Accuracy.

In addition to the accuracy scores, we also present two figures illustrating the improvement in DNN training loss and accuracy over 10 epochs. Figure-6.4 shows the training loss, which demonstrates a consistent decrease over the epochs, indicating that the DNN model is effectively learning from the training data. Figure-6.5 presents the training accuracy, which displays an upward trend over the 10 epochs, further confirming that the DNN model is improving its ability to recognize hand grasp patterns in the Exohand dataset as it trains. In conclusion, the performance comparison of the kNN with DTW, SVM, and DNN algorithms on the Exohand dataset demonstrates that the DNN model is the most effective approach for the hand grasp recognition task, followed by the SVM classifier, with kNN with DTW being the least effective. The results suggest that the hierarchical and kernel-based learning approaches are more suitable for complex pattern recognition tasks like hand grasp recognition, as they can better capture the underlying structure and variability in the data. The figures of the DNN training loss and accuracy further emphasize the effectiveness of the deep learning approach for this task.

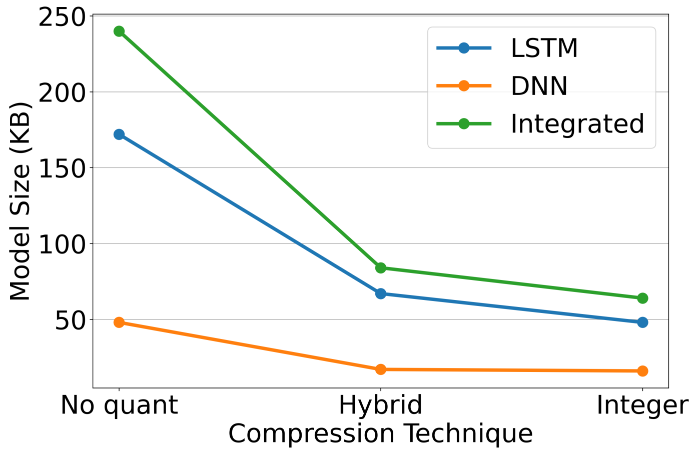

Figure 6.6: Model Size Reduction.

6.3 Memory Reduction after Compression

In this section, we present a comprehensive analysis of memory reduction after applying hybrid quantization and full-integer quantization to the Stacked-LSTM model for log-based anomaly detection, the DNN model for hand grasp recognition, and the secure framework combining these two models as demonstrated in Figure-6.6. The objective of this analysis is to evaluate the effectiveness of the quantization techniques in minimizing memory consumption, facilitating deployment on the Google Coral Board.

For the Stacked-LSTM model, we observed a considerable memory reduction when applying the quantization techniques. The original model consumed 172 KB of memory without any quantization. After applying hybrid quantization, which converts the model's weights from 32-bit float to 8-bit integer while retaining 32-bit float for biases and activation functions, memory consumption was reduced to 67 KB, representing a 61.05% reduction. When full-integer quantization was applied, changing the weights, biases, and activation functions from 32-bit float to 8-bit integer, memory consumption further decreased to 47 KB, indicating a 72.67% reduction compared to the original model size. Similarly, the DNN model experienced a significant reduction in memory consumption. The unquantized model had a size of 48 KB. By applying hybrid quantization, memory consumption dropped to 17 KB, which is a 64.58% reduction. With full-integer quantization, memory consumption slightly decreased to 16 KB, resulting in a 66.67% reduction compared to the original model size.

Lastly, the secure framework that combines both models also showed remarkable memory reduction results. Without quantization, the combined framework occupied 240 KB of memory space. Utilizing hybrid quantization, memory consumption was reduced to 84 KB, signifying a 65% reduction. With the application of full-integer quantization, memory consumption further decreased to 64 KB, demonstrating a 73.33% reduction compared to the original framework size. In conclusion, our analysis reveals that both hybrid quantization and full-integer quantization significantly reduce memory consumption for the Stacked-LSTM model, DNN model, and the secure framework combining the two models. This substantial memory reduction makes it possible to deploy these machine learning models on the Google Coral Board for edge inference, paving the way for efficient and low-resource machine learning applications.

6.4 Speed-Up in Google Coral Board

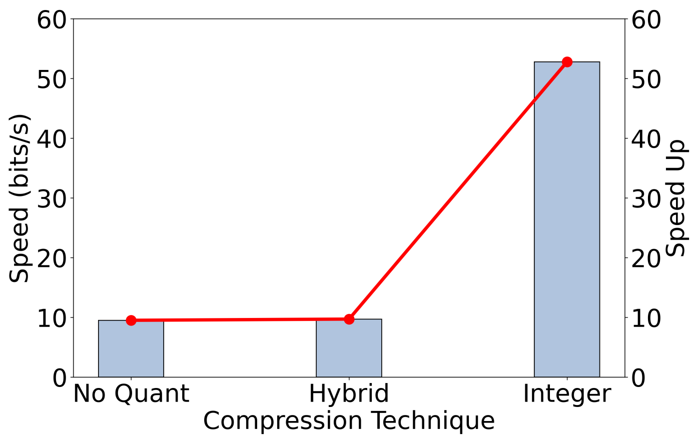

Figure 6.7: Speed Up after Compression.

In this section, Figure-6.7 illustrates the findings related to the speed-up performance of the secure framework for machine learning inference on the Google Coral Board. We analyze the speed improvements after applying no quantization, hybrid quantization, and full-integer quantization to the secure framework that combines the Stacked-LSTM model for log-based anomaly detection and the DNN model for hand grasp recognition. Our analysis revealed that the secure framework exhibited varying levels of speed-up depending on the quantization technique employed. Without any quantization, the framework achieved a speed-up of 9.52 bits/sec. When hybrid quantization was applied, which involved converting the weights of the machine learning models from 32-bit float to 8-bit integer while keeping the biases and activation functions as 32-bit float, the speed-up slightly increased to 9.72 bits/sec. This demonstrates a marginal improvement in performance compared to the non-quantized framework.

However, when full-integer quantization was applied, converting the weights, biases, and activation functions of the machine learning models from 32-bit float to 8-bit integer, the secure framework experienced a substantial increase in speed-up, reaching 52.78 bits/sec. This result highlights a more than five-fold improvement in performance compared to the non-quantized framework and a significant boost over the hybrid quantization approach. In summary, our findings indicate that the application of full-integer quantization to the secure framework for machine learning inference on the Google Coral Board results in a considerable increase in speed-up performance. This improvement not only enables more efficient deployment of the Stacked-LSTM and DNN models on the Coral Board but also facilitates faster and more responsive edge inference for various machine learning applications.

6.5 Performance on Google Coral Board

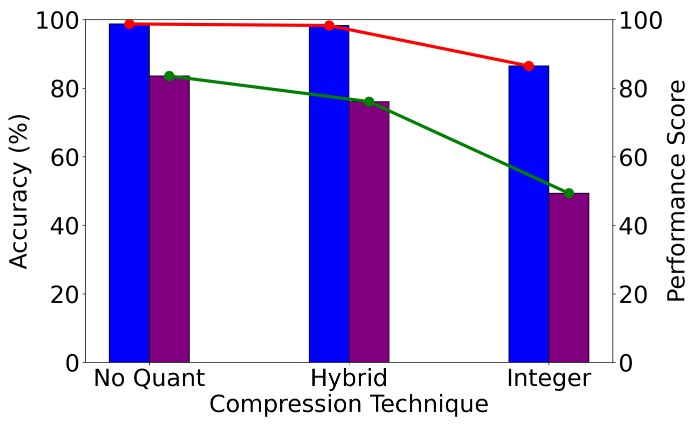

Figure 6.8: Performance of Secure Framework.

In this section, we discuss the performance of the secure framework for machine learning inference on the Google Coral Board, specifically focusing on the accuracy of the log-based anomaly detection model (Stacked-LSTM) and the hand grasp recognition model (DNN) after the application of compression techniques. Figure-6.8 illustrates two bar graphs, blue bar corresponding to Stacked-LSTM and the purple bar corresponding to DNN, which shows in detail the changes in accuracy as a result of employing no quantization, hybrid quantization, and full-integer quantization. Upon analyzing the Stacked-LSTM model for log-based anomaly detection, we observed a decrease in accuracy after applying the compression techniques. The model achieved an accuracy of 98.73% without any quantization. With hybrid quantization, the accuracy dropped slightly to 98.28%. However, when full-integer quantization was applied, the accuracy experienced a more significant decline, registering at 86.47%. This demonstrates that while the compression techniques reduce memory consumption and increase speed-up performance, they also have an impact on the model's accuracy.

A similar trend was observed for the DNN model for hand grasp recognition. The unquantized model obtained an accuracy of 83.56%. After applying hybrid quantization, the accuracy decreased to 76.07%. With full-integer quantization, the accuracy fell more substantially to 49.33%. This result indicates that, like the Stacked-LSTM model, the DNN model's accuracy is negatively affected by the application of compression techniques. Our experimental findings demonstrate that the secure framework for machine learning inference on the Google Coral Board experiences a trade-off between memory reduction, speed-up performance, and accuracy when compression techniques are applied. While hybrid quantization and full-integer quantization lead to significant memory reduction and increased speed-up, the accuracy of both the Stacked-LSTM model for log-based anomaly detection and the DNN model for hand grasp recognition is compromised. This trade-off must be carefully considered when deploying these models on the Google Coral Board for edge inference, balancing the need for efficient and low-resource machine learning applications with the desired level of accuracy.

Chapter 7

Conclusion & Future Work

In conclusion, we highlight that it is crucial to optimize the proposed security model for resource-limited edge devices, ensuring sufficient memory is available for the essential machine learning or non-machine learning applications to be executed on the edge platform. Moreover, these essential applications must achieve acceptable accuracy levels to be deployed in real-world situations while concurrently maintaining effective compression of the corresponding security model. This thesis proposes a secure and efficient framework for running log-based anomaly detection and hand grasp recognition tasks simultaneously on commercially available edge platforms. By employing a stacked-LSTM model for log-based anomaly detection and a deep learning-based vulnerability detection model, the system offers a robust approach to ensuring the safety and reliability of data processing tasks in the context of Big Data and the Internet of Things. The experimental results demonstrate the trade-offs between memory reduction, speed-up performance, and accuracy when deploying machine learning models on the Google Coral Board for edge inference. These insights provide valuable guidance for future research and practical applications, emphasizing the need to balance resource efficiency with the desired level of accuracy in machine learning implementations. By understanding the limitations and potential of various compression techniques, practitioners can make more informed decisions when implementing machine learning applications on edge devices. Furthermore, the proposed framework contributes to the ongoing development and optimization of distributed computing systems, particularly in the realm of secure and efficient data processing.

Building on the findings and insights from this thesis, several avenues for future research can be explored. Some potential directions for future work include:

- **Improved Compression Techniques:** Investigate more advanced compression techniques that can provide better trade-offs between memory reduction, speed-up performance, and accuracy. This research could focus on developing novel algorithms or optimizing existing methods to minimize the impact on model accuracy while maximizing resource efficiency.
- **Enhanced Security Mechanisms:** Explore additional security mechanisms and protocols to strengthen the system's overall security posture. This could involve incorporating secure multi-party computation, homomorphic encryption, or federated learning approaches to protect sensitive data and ensure privacy during the machine learning process.

Bibliography

[1] A. Sestino, M. I. Prete, L. Piper, and G. Guido, "Internet of things and big data as enablers for business digitalization strategies," *Technovation*, vol. 98, p. 102173, 2020.

[2] D. L. Andersen, C. S. A. Ashbrook, and N. B. Karlborg, "Significance of big data analytics and the internet of things (iot) aspects in industrial development, governance and sustainability," *International Journal of Intelligent Networks*, vol. 1, pp. 107–111, 2020.

[3] J. Saffran, G. Garcia, M. A. Souza, P. H. Penna, M. Castro, L. F. Góes, and H. C. Freitas, "A low-cost energy-efficient raspberry pi cluster for data mining algorithms," in *Euro-Par 2016: Parallel Processing Workshops: Euro-Par 2016 International Workshops, Grenoble, France, August 24-26, 2016, Revised Selected Papers.* Springer, 2017, pp. 788–799.

[4] C.-S. Kim and S.-B. Son, "A study on big data cluster in smart factory using raspberry-pi," in *2018 IEEE International Conference on Big Data (Big Data).* IEEE, 2018, pp. 5360–5362.

[5] G. Di Modica and O. Tomarchio, "A hierarchical hadoop framework to process geo-distributed big data," *Big Data and Cognitive Computing*, vol. 6, no. 1, p. 5, 2022.

[6] G. S. Bhathal and A. Singh, "Big data computing with distributed computing frameworks," in *Innovations in Electronics and Communication Engineering: Proceedings of the 7th ICIECE 2018.* Springer, 2019, pp. 467–477.

[7] K. Srinivasan, C.-Y. Chang, C.-H. Huang, M.-H. Chang, A. Sharma, and A. Ankur, "An efficient implementation of mobile raspberry pi hadoop clusters

for robust and augmented computing performance," *Journal of Information Processing Systems*, vol. 14, no. 4, pp. 989–1009, 2018.

[8] E. Badidi, Z. Mahrez, and E. Sabir, "Fog computing for smart cities' big data management and analytics: A review," *Future Internet*, vol. 12, no. 11, p. 190, 2020.

[9] G. S. Bhathal and A. Singh, "Big data: Hadoop framework vulnerabilities, security issues and attacks," *Array*, vol. 1, p. 100002, 2019.

[10] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?" in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 1356–1367.

[11] A. Cakir, Ö. Akın, H. F. Deniz, and A. Yılmaz, "Enabling real time big data solutions for manufacturing at scale," *Journal of Big Data*, vol. 9, no. 1, pp. 1–24, 2022.

[12] I. Yagoub, M. A. Khan, and L. Jiyun, "It equipment monitoring and analyzing system for forecasting and detecting anomalies in log files utilizing machine learning techniques," in *2018 International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD)*. IEEE, 2018, pp. 1–6.

[13] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs." in *IJCAI*, vol. 19, no. 7, 2019, pp. 4739–4745.

[14] K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechrinis, and H. Zhang, "Automated it system failure prediction: A deep learning approach," in *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 2016, pp. 1291–1300.

[15] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*. IEEE, 2016, pp. 207–218.

[16] H. Mi, H. Wang, Y. Zhou, M. R.-T. Lyu, and H. Cai, "Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1245–1255, 2013.

[17] X. Yu, P. Joshi, J. Xu, G. Jin, H. Zhang, and G. Jiang, "Cloudseer: Workflow monitoring of cloud infrastructures via interleaved logs," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 2, pp. 489–502, 2016.

[18] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Largescale system problem detection by mining console logs," *Proceedings of SOSP'09*, 2009.

[19] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 2625–2634.

[20] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," *arXiv preprint arXiv:1402.1128*, 2014.

[21] V. Chavan, A. Malage, K. Mehrotra, and M. K. Gupta, "Printed text recognition using blstm and mdlstm for indian languages," in *2017 Fourth International Conference on Image Information Processing (ICIIP)*. IEEE, 2017, pp. 1–6.

[22] A. M. Kist, "Deep learning on edge tpus," *arXiv preprint arXiv:2108.13732*, 2021.

[23] S. Hosseininoorbin, S. Layeghy, B. Kusy, R. Jurdak, and M. Portmann, "Exploring deep neural networks on edge tpu," *arXiv preprint arXiv:2110.08826*, 2021.

[24] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[25] J. Liu, S. Tripathi, U. Kurup, and M. Shah, "Pruning algorithms to accelerate convolutional neural networks for edge applications: A survey," *arXiv preprint arXiv:2005.04275*, 2020.

[26] A. Bicchi, "Hands for dexterous manipulation and robust grasping: A difficult road toward simplicity," *IEEE Transactions on robotics and automation*, vol. 16, no. 6, pp. 652–662, 2000.

[27] R. Deimel and O. Brock, "A novel type of compliant and underactuated robotic hand for dexterous grasping," *The International Journal of Robotics Research*, vol. 35, no. 1-3, pp. 161–185, 2016.

[28] C. Sapsanis, G. Georgoulas, A. Tzes, and D. Lymberopoulos, "Improving emg based classification of basic hand movements using emd," in *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2013, pp. 5754–5757.

[29] A. Nishad, A. Upadhyay, R. B. Pachori, and U. R. Acharya, "Automated classification of hand movements using tunable-q wavelet transform based filter-bank with surface electromyogram signals," *Future Generation Computer Systems*, vol. 93, pp. 96–110, 2019.

[30] M. Verhelst and B. Moons, "Embedded deep neural network processing: Algorithmic and processor techniques bring deep learning to iot and edge devices," *IEEE Solid-State Circuits Magazine*, vol. 9, no. 4, pp. 55–65, 2017.

[31] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE network*, vol. 32, no. 1, pp. 96–101, 2018.

[32] J. Mirkovic, G. Prier, and P. Reiher, "Attacking ddos at the source, inproc. of the 10th ieee international conference on network protocols (icnp'02)," *Washington DC, USA*, 2002.

[33] M. Cinque, D. Cotroneo, and A. Pecchia, "Event logs for the analysis of software failures: A rule-based approach," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 806–821, 2012.

[34] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, "Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks," in *Proceedings of the* $29^{th} annual computer security applications conference, 2013, pp. 199--208.$

[35] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 117–132.

[36] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *2010 USENIX Annual Technical Conference (USENIX ATC 10)*, 2010.

[37] A. Oprea, Z. Li, T.-F. Yen, S. H. Chin, and S. Alrwais, "Detection of early-stage enterprise infection by mining large-scale log data," in *2015 45th Annual IEEE/I-*

FIP International Conference on Dependable Systems and Networks. IEEE, 2015, pp. 45–56.

[38] R. Vinayakumar, K. Soman, and P. Poornachandran, "Long short-term memory based operation log anomaly detection," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI).* IEEE, 2017, pp. 236–242.

[39] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1285–1298.

[40] M. Daniluk, T. Rocktäschel, J. Welbl, and S. Riedel, "Frustratingly short attention spans in neural language modeling," *arXiv preprint arXiv:1702.04521*, 2017.

[41] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 807–817.

[42] J. W. Kim, S. You, S. H. Ji, and H. S. Kim, "Real-time hand grasp recognition using weakly supervised two-stage convolutional neural networks for understanding manipulation actions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW).* IEEE, 2017, pp. 481–483.

[43] Y. Lin, R. Palaniappan, P. De Wilde, and L. Li, "Robust long-term hand grasp recognition with raw electromyographic signals using multidimensional uncertainty-aware models," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 31, pp. 962–971, 2023.

[44] U. Zakia, X. Jiang, and C. Menon, "Deep learning technique in recognizing hand grasps using fmg signals," in *2020 11th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON).* IEEE, 2020, pp. 0546–0552.

[45] F. Palermo, M. Cognolato, A. Gijsberts, H. Müller, B. Caputo, and M. Atzori, "Repeatability of grasp recognition for robotic hand prosthesis control based

on semg data," in *2017 International Conference on Rehabilitation Robotics (ICORR)*. IEEE, 2017, pp. 1154–1159.

[46] M. Cognolato, M. Atzori, R. Gassert, and H. Müller, "Improving robotic hand prosthesis control with eye tracking and computer vision: A multimodal approach based on the visuomotor behavior of grasping," *Frontiers in artificial intelligence*, vol. 4, p. 199, 2022.

[47] A. A. Süzen, B. Duman, and B. Şen, "Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn," in *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*. IEEE, 2020, pp. 1–5.

[48] A. Komninos, I. Simou, N. Gkorgkolis, and J. Garofalakis, "Performance of raspberry pi microclusters for edge machine learning in tourism," *Network (Mbps)*, vol. 100, no. 100, p. 100, 2019.