Enhancing Performance of Intelligent IoT Applications in Edge-Cloud Continuum

A Thesis Submitted

in Partial Fulfilment of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

by

Ashish Kumar Kaushal

(2019csz0003)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY ROPAR

SEPTEMBER, 2024

Dedicated to my family !!

Declaration of Originality

I hereby declare that the work which is being presented in the thesis entitled **Enhancing Performance of Intelligent IoT Applications in Edge-Cloud Continuum** has been solely authored by me. It presents the result of my own independent investigation/research conducted during the time period from July 2019 to August 2024 under the supervision of Dr. Nitin Auluck and Dr. Balwinder Sodhi at IIT Ropar. To the best of my knowledge, it is an original work, both in terms of research content and narrative, and has not been submitted or accepted elsewhere, in part or in full, for the award of any degree, diploma, fellowship, associateship, or similar title of any university or institution. Further, due credit has been attributed to the relevant state-of-the-art and collaborations (if any) with appropriate citations and acknowledgments, in line with established ethical norms and practices. I also declare that any idea/data/fact/source stated in my thesis has not been fabricated/falsified/misrepresented. All the principles of academic honesty and integrity have been followed. I fully understand that if the thesis is found to be unoriginal, fabricated, or plagiarized, the Institute reserves the right to withdraw the thesis from its archive and revoke the associated Degree conferred. Additionally, the Institute also reserves the right to appraise all concerned sections of society of the matter for their information and necessary action (if any). If accepted, I hereby consent for my thesis to be available online in the Institute's Open Access repository, inter-library loan, and the title & abstract to be made available to outside organizations.

Signature

Ashish Kumar Kaushal
2019csz0003
PhD
Department of CSE
Indian Institute of Technology Ropar
Rupnagar, Punjab 140001

Date: September 10, 2024

Acknowledgement

First of all, I begin by expressing my deepest gratitude to almighty God, whose unending guidance and grace have sustained me throughout this entire journey of PhD. I also want to thank IIT Ropar for providing me with the platform and all the facilities required for completing my PhD.

I am profoundly grateful to my supervisor, Dr. Nitin Auluck, for his continuous support and guidance, as well as the freedom he gave me in pursuing my ideas. I thank him for his patience and mentorship even when I faced significant challenges. I also would like to thank my co-supervisor Dr Balwinder Sodhi for his valuable contributions and support.

I must extend my special gratitude to Prof. Omer Rana for being an extraordinary mentor and collaborator. I have learned a lot of things from him throughout my journey of PhD. His willingness to share expert insights have significantly contributed to my both professional and personal growth as a researcher. I am also thankful to my collaborator and friend Osama Almurshed for all the expertise, suggestions, advice, and brainstorming sessions which played an important role in completing our work. I am indebted to my other collaborators Bharadwaj Veeravalli, Areej Alabbas, Asmail Muftah, Osama Almoghamis for being a part of my work and providing their timely expertise.

I am immensely grateful to all of my doctoral committee members: Dr. Sudarshan Iyenger (chairperson), Dr. Basant Subba, Dr. Jagpreet Singh, and Dr. Brijesh Kumbhani for evaluating my work and providing me valuable feedback that improved the quality of my work.

Special thanks to my friends Akash Anil, Hsuvas Borkakoty, Amit Gajbhiye, Nitesh Kumar, Vidushi Agarwal, Abhishek Singh Sambyal, Armaan Garg, Gulshan Sharma, Pooja Bharadwaj, Sahil Kumar, Tirtha Das, Sarbjeet Kaur, Pankaj Singla, Saweta Garg, Aakriti Gupta for being a source of motivation and inspiration throughout the journey.

I also extend my deepest appreciation to my family for their unconditional love, understanding, and encouragement. Their faith in me has consistently provided strength and resilience throughout the challenging times of this academic journey.

This achievement would not have been possible without all of you. Thank you for being a part of my PhD journey.

– Ashish

Certificate

This is to certify that the thesis entitled **Enhancing Performance of Intelligent IoT Applications in Edge-Cloud Continuum**, submitted by **Ashish Kumar Kaushal (2019csz0003)** for the award of the degree of **Doctor of Philosophy** of Indian Institute of Technology Ropar, is a record of bonafide research work carried out under our guidance and supervision. To the best of my knowledge and belief, the work presented in this thesis is original and has not been submitted, either in part or full, for the award of any other degree, diploma, fellowship, associateship or similar title of any university or institution. In our opinion, the thesis has reached the standard fulfilling the requirements of the regulations relating to the Degree.

Signature of the Supervisor
Dr. Nitin Auluck
Department of CSE
Indian Institute of Technology Ropar
Rupnagar, Punjab 140001
Date: December 17, 2024

Signature of the Supervisor
Dr. Balwinder Sodhi
Department of CSE
Indian Institute of Technology Ropar
Rupnagar, Punjab 140001
Date: December 17, 2024

Lay Summary

As the Internet of Things (IoT) continues to grow, more devices than ever are connected and communicating every day, from smart home appliances to industrial sensors. This leads to a large amount of data being generated everywhere, all the time. Traditionally, this data was sent back to large central computers (cloud) for processing. However, to make things faster and more efficient, a lot of this processing is now being done closer to where the data is actually generated – at the "edge" of the network, like on local devices or nearby servers.

This thesis aims to make intelligent IoT systems work better, faster, and more reliably by tackling a few key challenges. Initially, the thesis presents a smart way to decide where and how tasks/jobs should be allocated within this new edge-to-cloud environment to save time and increase performance, especially when immediate decisions are needed by IoT applications. It also includes designing a method to efficiently distribute the task load on all the available resources considering various factors that directly affect the efficiency of the environment. The factors monitor the total running time and resources utilised of the system and try to minimise them. Another approach is to develop new methods to improve how these systems learn and make decisions by training them more efficiently. This involves choosing the most important parts of a system that need learning and removing unnecessary parts, making everything run smoother and quicker. Lastly, the thesis explore how to better manage all the data these devices use and generate, ensuring that it is stored safely and can be accessed quickly, even if something goes wrong. By considering these scenarios, this thesis aims to enhance how smart devices operate and communicate, making them better suited to the needs of our fast-paced, data-driven world.

Abstract

The rapid expansion of the Internet of Things (IoT) has resulted in a paradigm shift of computing from centralised cloud to edge environments, where data processing is performed closer to the source. However, the deployment of intelligent IoT applications within this edge-cloud continuum presents unique challenges, including resource management, data processing efficiency, and maintaining system reliability. This thesis focuses on enhancing the performance of intelligent applications by designing approaches for optimising task allocation, load distribution, Machine Learning (ML) operations, and data management in the IoT infrastructure.

The thesis aims to design a framework that supports efficient and cost-effective operation of IoT applications across the edge-cloud continuum. I first propose an algorithm for dynamic task allocation that emphasises on minimising the completion time while maximising the task execution performance. By formulating the algorithm that dynamically allocates tasks based on real-time analytics and system state, the approach effectively reduces execution latency and enhances the accuracy of real-time decision-making processes. In addition to task allocation, this thesis presents a load distribution framework for IoT applications deployed on edge computing infrastructure. The mechanism prioritises completion time, waiting time, resource utilisation, evaluation overhead, failure rate, and provides a strategic approach that classifies tasks and computational resources into categories such as restricted, public; and private, shared. This results in a security-aware load distribution mechanism that handles IoT-based tasks in real-time. In order to optimise the ML and Artificial Intelligence (AI) operations, the thesis introduces an approach to select layers for model training using a genetic algorithm. This method determines the optimal configuration of active and inactive layers which enhances the model efficiency and adaptability during training phases. A pruning mechanism is also developed which utilises heatmap to identify performance-critical features and simplifies the model by eliminating non-essential features. This dual approach significantly reduces computational overhead and execution time while preserving the essential analytical capabilities of the model and maintaining its accuracy. To handle IoT-based data, the thesis also proposes a methodology that ensures optimal storage, access, and recovery of data and model files in case any data loss or system failure occurs. All these methods are designed to enhance the resilience of the IoT system, ensuring that their performance, data integrity, and availability are maintained even under adverse conditions. Through mathematical formulation of the problems and implementation via simulation and testbed, I validate the feasibility and performance of proposed frameworks on an agricultural (weed detection) use-case scenario.

Keywords: Edge-Cloud Continuum, Internet of Things, Load Distribution, Machine Learning, Serverless, Task Allocation.

List of Publications

Journal

1. **A. Kaushal**, O. Almurshed, O. Almoghamis, A. Alabbas, N. Auluck, B. Veeravalli and O. Rana, "SHIELD: A Secure Heuristic Integrated Environment for Load Distribution in Rural-AI," in *Elsevier Future Generation Computer Systems (FGCS) Journal*, 2024, vol. 161, pp. 286-301.

Conference

1. **A. Kaushal**, O. Almurshed, A. Alabbas, N. Auluck and O. Rana, "An Edge-Cloud Infrastructure for Weed Detection in Precision Agriculture," in *The 21st IEEE International Conference on Pervasive Intelligence and Computing (PiCom)*, Abu Dhabi, UAE, 2023, pp. 0269-0276.

Under Review

1. **A. Kaushal**, O. Almurshed, A. Muftah, P. Kundan, T. Abhijith, N. Auluck and O. Rana, "Towards a Sustainable Optimisation Approach for Machine Learning Tasks in Internet of Things," in *IEEE Internet of Things Journal*.
2. **A. Kaushal**, O. Almurshed, P. Kundan, T. Abhijith, N. Auluck and O. Rana, "Enhancing Data Management in Machine Learning Applications: Resilience and Storage Optimisation of Serverless Edge-Cloud," in *IEEE Internet of Things Magazine*.

List of Abbreviations

AI	Artificial Intelligence
CNN	Convolutional Neural Network
EC	Erasure Coding
FAO	Food and Agriculture Organisation
FaaS	Function as a Service
FL	Federated Learning
FLOP	Floating Point Operation
FR	Failure Rate
FSU	Field Side Unit
GCP	Google Cloud Platform
GHG	Greenhouse Gas
GradCAM	Gradient-weighted Class Activation Mapping
HMAC	Hash-Based Message Authentication Code
IIT	Indian Institute of Technology
ILP	Integer Linear Programming
IoT	Internet of Things
LRC	Locally Recoverable Codes
LLM	Large Language Model
MD	Mobile Device
ML	Machine Learning
MCSD	Multitype Cloud Storage Dataset
MTBF	Mean Time between Failures
MTTF	Mean Time to Failure
MTTR	Mean Time to Recovery
OW	OpenWhisk
QoS	Quality of Service
RPi	Raspberry Pi
RS	Reed Solomon
RU	Resource Utilisation
SBC	Single Board Computer
SC	System Cost
SDG	Sustainable Development Goal
SFC	Service Function Chain
UAV	Unmanned Aerial Vehicle
WSN	Wireless Sensor Network

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The chapter presents an overview of the Internet of Things (IoT), edge-cloud continuum architecture, and associated services with them. It also highlights the necessity for enhancing and improving the performance of intelligent IoT applications using edge and cloud resources, considering the rise in data volumes and computational demands of real-life applications.

1.1 Internet of Things

The IoT is a vast network of devices that are connected to the internet where each node is capable of collecting and sharing data across a network without requiring direct human-to-human or human-to-machine interaction. This network integrates various types of objects embedded with electronics, software, sensors, actuators, and connectivity to enable the exchange of information with other devices and systems [1]. These devices can range from smart home appliances to sophisticated industrial machines.

IoT networks have the ability to enhance operational efficiency, improve human decision-making, and increase the value of services through automation and data integration. The primary element of an IoT network is a sensor that gathers data from the nearby environment. This data can include everything from temperature readings to complex changes in the air, depending on the sensor and its purpose. Once collected, the data is sent to computing devices located closer to the IoT nodes, which pre-process the data to reduce latency and network congestion [2]. After initial processing or computation, the data can be sent for further analysis and storage, enabling more comprehensive data management and task execution. Figure 1.1 below shows the mechanism for handling tasks and data within an IoT infrastructure.

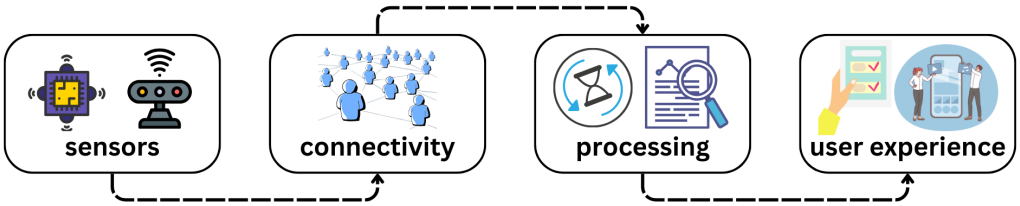

Figure 1.1: Execution mechanism within an IoT network.

The growth of IoT has been rising exponentially, leading to a surge in the number of connected devices in IoT infrastructures. Each year, billions of new devices are connected which generates large volumes of data very rapidly. This rapid expansion is largely driven

by the decreasing cost of sensors and connectivity hardware, as well as the increasing availability of computational resource nodes [3]. The data produced by connected devices is also becoming a critical element for businesses and governments as they can use it to make more informed decisions, optimise operations, and predict future market trends.
However, the huge amount of generated data also presents significant challenges that necessitate innovative approaches for efficient IoT management. As the number of connected devices continues to grow at a rapid pace, the volume of data they produce is enormous and continues to increase exponentially. This large amount of data requires sophisticated strategies for storage, processing, and analysis to ensure it can be used effectively and efficiently used within task execution infrastructures [4].

1.2 Edge-Cloud Continuum

In order to handle the large amount of data, the traditional approach for computation is cloud computing. It is a form of computing that relies on shared computing resources rather than having local servers or personal devices for handling the execution of applications. It offers a variety of services such as servers, storage, databases, networking, software, analytics, and intelligence over the internet to enable faster innovation, flexible resources, and economies of scale. Cloud computing is known for its high availability, ensuring that services are accessible nearly 100% of the time from anywhere in the world. This is highly important for services that require continuous uptime in applications [5]. They also have rapid deployment capabilities which allow organisations to launch applications and services easily while minimising the initialisation time. Moreover, the cloud providers manage the maintenance and service updates, which reduces the burden of managing IT operations, allowing them to focus more on features and strategic actions of the application.
However, the adoption of cloud computing also introduces several challenges. High latency is a significant issue, particularly for applications requiring real-time processing, as data travels to and from the cloud servers which usually causes delays. Security vulnerabilities are another critical concern; despite robust security measures by cloud providers, the risk of data breaches and other cyber threats persists due to the external management of data. Vendor lock-in is another drawback where owners may find it difficult and costly to switch providers once they have committed to one, due to compatibility issues and contractual limitations [6]. Additionally, cloud computing also relies heavily on internet connectivity, and any instability in the connection can disrupt access to services, potentially leading to significant operational delays and losses.
To handle these primary challenges associated with cloud service providers, edge computing provides another computational approach. It is a distributed computing paradigm that improves processing by bringing computation closer to the data source. This approach is fundamentally distributed as it utilises numerous edge devices to handle computations locally. This decentralisation helps reduce the distance data or tasks traverse

and significantly minimises latency and bandwidth. It also improves the privacy and security of data by retaining it at the edge (much closer in comparison to centralised data centres). Edge computing also offers robust support for mobile environments where connectivity might be intermittent [7]. Mobile devices, vehicles, and other moving entities can also maintain operational effectiveness by processing data locally, ensuring that they continue to function efficiently irrespective of the network status of a central server.

However, edge computing comes with its own set of challenges. The devices used at the edge typically have limited computational power and storage capacity, which might restrict the complexity of tasks they can handle and require fallback to more powerful central computing resources for more demanding processes. Additionally, managing an edge computing infrastructure introduces significant complexity due to the need to coordinate a vast number of diverse devices and technologies [8]. This setup involves consistent updates, security management, and maintaining connectivity across heterogeneous devices, which can complicate standardisation and broad-scale deployment. Despite these challenges, the strategic advantages of edge computing – particularly its capacity for low latency and enhanced local processing – continue to drive its adoption in fields requiring quick, local data processing and decision-making.

To handle the challenges associated with both cloud and edge computing, a popular approach is to utilise the edge-cloud continuum for executing tasks. The edge-cloud continuum represents a transformative approach in computing that takes the distinct advantages of each approach, optimising data processing and storage across a continuum from the data source to the cloud. The continuum allows data to be processed at the most appropriate location depending on the performance, cost, and efficiency considerations, selected specifically based on the needs of each application. The edge-cloud continuum offers a dynamic and flexible approach to data processing for modern applications, balancing the need for rapid response capabilities at the edge with the powerful processing and analytic capabilities of the cloud. This integrated approach is particularly well-suited to the demands of modern IoT applications in smart cities, healthcare etc, where diverse data processing needs a versatile and adaptive solution.

1.3 Architecture for Edge-Cloud Continuum

The architecture of the edge-cloud continuum is designed to leverage both the localised processing power of edge computing and the large resource pool of cloud computing. This hybrid architecture ensures that data is processed efficiently, quickly, and at the scale required by modern IoT applications [9]. The architecture typically comprises multiple layers where each layer serves a specific function in the continuum from data generation to actionable insights. Figure 1.2 shows an architecture for edge-cloud continuum which can be used for executing IoT applications.

The first layer of the architecture is an end-user layer, which consists of the physical devices and sensors that initially capture data. This layer is the closest to the physical world

and includes a wide array of IoT devices like cameras, environmental sensors, wearable devices, and industrial machines. The primary role of these devices is to perform basic data collection and preliminary processing. They can filter and pre-process the data to reduce the volume that needs to be transmitted to higher layers, addressing issues of bandwidth and communication latency effectively. Once the data is preprocessed and filtered, it is forwarded to the next layer for execution.

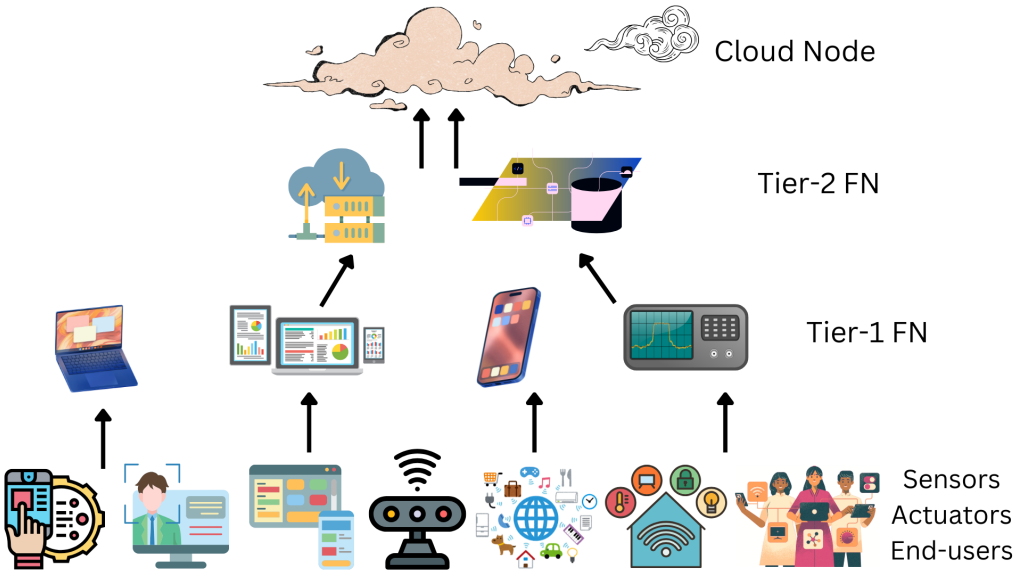

Figure 1.2: Architecture for the edge-cloud continuum.

The middle layer is the fog layer which acts as an intermediary between the edge nodes and the cloud, providing additional processing resources closer to the edge of the network. This layer facilitates more complex processing that may not be feasible at the edge due to resource constraints but still requires lower latency than cloud processing. This layer can manage tasks like advanced analytics, longer-term data retention, and more detailed real-time processing, distributing these tasks more efficiently across the network. It is also possible to have multiple hierarchies within this fog layer, such that each higher layer can have higher computational and storage capacity whereas the lower layers have better response time and lower execution delay. At the top of the hierarchy is the cloud layer, which provides powerful computational resources and massive storage capabilities. This layer is ideal for execution that requires intensive computation or involves vast amounts of data that need long-term storage. The cloud can perform complex analytics, machine learning model training, and comprehensive data mining tasks. Data that requires no immediate action but is valuable for historical analysis, pattern recognition, and predictive analytics is typically handled at this layer.

1.4 Integrating ML, AI with Edge and Cloud

Integrating Machine Learning (ML) and Artificial Intelligence (AI) with both edge and cloud computing has transformed the way data-driven insights are generated and utilised

by creating a more responsive and intelligent network infrastructure. This integration utilises the strengths of ML and AI to enhance the capabilities at the edge-cloud continuum, enabling smarter decision-making processes and more efficient operations across various sectors [10].

At the edge level, ML-AI integration focuses on real-time data processing and immediate decision-making. Since edge devices operate close to the data sources, they are well positioned to implement ML models that require quick responses, such as facial recognition systems, threat detection in farms, immediate traffic flow optimisation, or predictive maintenance sensors on industrial equipment. These applications benefit from the low latency provided by edge computing, as they often need to function in near real-time without the delay that would come from sending data to a distant cloud server. However, edge devices typically have limited computational power and storage capacity, which restricts the complexity of the ML models they can deploy. To address this limitation, simplified or compressed versions of models are often used at the edge, derived from more complex models trained in the cloud [11]. This approach allows edge devices to execute AI-driven tasks efficiently, using models that are periodically updated and refined in the cloud.

The cloud layer also has a crucial role in this integrated setup by providing the resources necessary for more intensive ML tasks, such as training large-scale models and performing complex analytics that are not time-sensitive. The cloud's powerful computational capabilities and vast storage options make it ideal for handling high-load AI tasks, including large language models (LLMs), deep learning models, and large-scale data mining frameworks. These tasks benefit from the cloud's ability to aggregate and analyse data collected from multiple edge devices, enabling more comprehensive insights and model improvements.

Moreover, this integration requires continuous management and coordination to ensure that AI models are deployed effectively across the network [12]. These layers handle model updates, data synchronisation, and resource allocation, ensuring that ML tasks are carried out efficiently and securely, balancing between edge and cloud based on specific application needs and network conditions. Integrating ML, AI with edge and cloud computing not only enhances operational efficiencies but also opens up new possibilities for innovation in fields ranging from autonomous driving and smart cities to healthcare and industrial automation. This dynamic cooperation of computing and AI technologies can be foundational in designing the next generation of smart, connected IoT applications for our everyday requirements.

1.5 IoT Applications Utilising Edge and Cloud

The integration of IoT with the edge-cloud continuum can improve various industries by enabling more efficient, responsive, and intelligent applications. These applications leverage the strengths of both edge computing for real-time execution and cloud computing

for intensive data analysis. Several key applications in IoT utilising the edge-cloud continuum are as follows:

- **Healthcare Monitoring System:** In the healthcare sector, IoT devices utilise edge computing to monitor patient health metrics in real-time. Wearable devices can track vital signs like heart rate and blood pressure, and edge devices can process this data immediately to detect anomalies. This setup allows for prompt medical responses, potentially saving lives in emergency situations. The cloud nodes can aggregate patient data over time, providing improved health analytics, research, and personalised medicine by analysing trends and improving diagnostic algorithms [13].

- **Smart Cities:** IoT applications in smart cities [14] make extensive use of edge-cloud architecture to enhance urban management and quality of life. Sensors and IoT devices deployed throughout a city can monitor traffic flow, air quality, and energy usage in real-time. At the edge, data is processed locally on roadside units to enable immediate responses, such as adjusting traffic lights, optimising traffic flow, or triggering alerts for pollution control. More comprehensive data analysis and planning tasks are handled in the cloud, facilitating long-term urban planning and resource management.

- **Precision Agriculture:** IoT devices are used extensively in the agriculture sector to monitor soil conditions, crop health, yield prediction, and weather conditions. Sensors can provide immediate data to edge nodes, which can process the information to give real-time feedback to actuation systems and optimise resource usage [15]. Cloud computing contributes by analysing seasonal data and longer-term trends to assist in crop planning, disease prevention strategies, and yield optimisation techniques. Moreover, in rural agricultural environments where network connection is inconsistent and unreliable, edge-cloud infrastructure can play a very crucial role. It can provide local resources for computation and storage along with better security and reliability.

- **Industrial Automation:** IoT in industrial automation relies on the edge-cloud continuum for monitoring and optimising manufacturing processes [16]. Sensors on machinery can detect operational anomalies or inefficiencies and process this information at the edge to make immediate adjustments or shut down equipment if a malfunction is detected. Meanwhile, data collected across the devices deployed in a factory can be sent to the cloud for predictive maintenance, overall efficiency improvements, and integration with enterprise resource planning systems.

- **Autonomous Vehicles:** Autonomous vehicles are equipped with numerous sensors that generate vast amounts of data crucial for safe operation on the roads. Processing this data on the edge allows for immediate decision-making essential for navigation and obstacle avoidance. The cloud plays a role in updating navigation maps, sharing traffic, music library, weather updates, and improving algorithms based on

aggregated data from multiple vehicles, enhancing the overall execution and safety of autonomous driving systems [17].

1.6 Challenges in Edge-Cloud Infrastructures

Deploying IoT applications within the edge-cloud continuum presents several significant challenges that can impact the efficiency and effectiveness of formulated systems. A few of the critical challenges are as follows:

- Limited connectivity is a major challenge in rural or remote environments where network infrastructure is often insufficient. This constraint restricts the ability of IoT devices to communicate effectively with edge or cloud servers, limiting real-time data processing and decision-making capabilities [18]. As connectivity is crucial for the operation of IoT systems, any inconsistency or interruption can degrade the performance and reliability of the entire IoT ecosystem.

- Variable load distribution is also challenging in IoT applications due to the dynamic nature of IoT data generation and consumption [19]. For instance, certain events or times may trigger high volumes of data traffic, which can over-utilise the system if not managed properly. This variability can cause performance bottlenecks, especially when the infrastructure is already under strain, and can complicate the management of resources across the network.

- Efficiency in resource allocation is crucial in environments where resources are inherently limited and expensive. Edge computing devices, while beneficial for processing data locally and reducing latency, often have less computational power and storage capacity compared to centralised cloud data centres [20]. Efficiently utilising these limited resources, while ensuring optimal performance across the IoT network requires careful management and can be a complex task to achieve consistently.

- Deployment of AI and ML tasks in IoT applications introduces additional complexities. As the volume and complexity of data generated by IoT devices grow, deploying advanced AI and ML models becomes more challenging [21]. These models often require significant computational resources for training and inference, which might not be readily available in edge scenarios. The large size of modern AI models also poses a challenge, as they need to be accommodated within the constraints of the available infrastructure.

- Optimising ML operations to run effectively on resource-constrained edge nodes is another issue in IoT-based infrastructures [22]. Traditional AI models and operations are designed for environments with an abundance of computational resources. Adapting these to work efficiently within the limitations of edge

computing environments requires significant modifications and optimisations, which can be a complex and time-intensive process.

- Data management in edge-cloud environments involves handling the storage, processing, and security of large volumes of data. Managing this data effectively is critical for performance but becomes challenging as the data is distributed across a multitude of devices and locations [23]. Ensuring data integrity, privacy, and timely access in such a distributed setup is a critical task that requires sophisticated data management strategies.

1.7 Research Objectives of the Thesis

The thesis aims to explore the fundamental challenges associated with deploying and managing ML and AI based IoT applications on edge-cloud infrastructures. I have evaluated the specific aspects of IoT systems to improve the overall resilience, efficiency, and effectiveness of the edge-cloud infrastructures. To achieve this, the following research objectives have been considered:

How can I enhance the resilience of task placement in IoT applications where network connectivity is limited and the edge-cloud infrastructure is unreliable? Execution resilience in this context refers to the system's ability to maintain operational stability and performance despite disruptions or poor network conditions. The aim is to develop a technique that allows IoT tasks to be dynamically adjusted and optimally placed within a network of edge-cloud nodes. This includes exploring how tasks can be efficiently allocated or handled locally when connectivity to a central server or cloud is unavailable.

Can I develop a mechanism for distributing the IoT based task load on edge computing nodes without compromising the performance or resource utilisation? This research question explores the design of a mechanism that optimally distributes the workload among available edge computing nodes. The method aims to ensure that no single node is over-utilised by demands, which could compromise both performance and resource utilisation. The challenge is to achieve a balance that maximises the efficiency of resource use while maintaining or enhancing the performance of IoT applications. This involves creating adaptive algorithms that can adjust in real time to changes in the network or application demands, thereby maintaining a balanced operation across the network.

Is there an approach to improve the execution and deployment of ML and AI operations in IoT based applications? This question addresses the improvement while executing machine learning and artificial intelligence operations within IoT frameworks. As IoT devices generate vast amounts of data, efficiently processing this data to extract valuable insights becomes a critical factor. My aim is to explore new methods which optimise ML and AI algorithms for better suitability in IoT setups, particularly focusing on minimising latency, reducing computational overhead, and ensuring that these operations can run effectively even on computationally limited edge computing devices.

How can I improve the storage and access of data, ML model files in IoT based applications in case any loss or failure occurs? The research question seeks to enhance the management of data, specifically for ML and AI based tasks within IoT applications operating on edge-cloud infrastructures. Effective data management is crucial for maintaining the performance of IoT systems, which includes ensuring data integrity, minimising latency in data access, and optimising storage across a distributed network. The aim is to explore techniques for improving data synchronisation and accessibility, ensuring that data is efficiently available to support advanced analytics and real-time decision-making processes.

1.8 Primary Contributions

The main contributions of the thesis are as follows:

- Propose an architectural framework for a real-time image classification problem with intermittent network connectivity. This problem is then mapped to a weed detection use case – widely considered significant in precision agriculture. To demonstrate the efficacy of our conceptual architecture, two ML models based on *ResNet-50* and *MobileNetV2* have been trained for identifying weeds, using images captured from a robot mounted camera. Light versions of these two models have been generated respectively using the model quantization technique. A rule-based algorithm is also formulated for decision-making, taking into account where to perform this classification: locally or remotely.

- Formulate a security-aware load balancing framework for edge infrastructure that can be used to support rural environments. The infrastructure is designed to distribute computational tasks across multiple resources while also ensuring data privacy and confidentiality. Our framework divides the tasks into independent categories: restricted and public. These tasks are allocated to two different resources: private and shared, based on security requirements and load characteristics of the node. A two-function heuristic pipeline is designed to make the resource allocation decision for each task, taking account of factors that have a direct influence on execution performance. Completion time, waiting time, failure rate, resource utilisation, security, and management overhead are the key factors used during evaluation.

- Design two distinct methodologies that enhance the efficiency of ML operations in IoT applications – The first method aims to reduce the computational demands associated with backpropagation by systematically freezing certain layers of the neural network. This involves fixing the parameters of selected layers to prevent them from updating during training. The primary objective is to cut down on the time required for both the current training session and any subsequent adjustments to model in the future. Another method focuses on refining the architecture of an ML model by adjusting its parameter count. This strategy uses an automated process to

identify and prioritise the most significant parameters for a specific dataset on which it is trained. Subsequently, parameters that are found less critical are eliminated from the model through a pruning procedure. This approach enhances the overall efficiency of ML operations for both the training and inference phases, by reducing the model's complexity.

- Presents an approach for managing the distribution, storage, and access of substantial amounts of data generated by IoT devices within ML and edge computing environments. The designed method focuses on data reduction at the source, standardising formats for compatibility, efficient data compression, and replication techniques, using the Reed-Solomon erasure coding technique. The aim is to focus on balancing storage efficiency with network and processing demands. It enhances efficiency by storing the static structure of the ML model separately from its dynamic parameters. A single instance of the model's static structure is maintained, and marita coding is applied to the frequently updated parameters. This approach minimises redundancy and storage requirements, ensuring reliable protection of dynamic data elements against data loss.

1.9 Structure of the Thesis

The thesis has been organised in the following chapters: Chapter 1 describes the introduction to IoT and provides an overview of the edge-cloud continuum architecture. It also highlights the challenges associated with edge-cloud infrastructures while deploying IoT applications and identifies the research objectives that could improve their performance. Chapter 2 provides the details about existing work done in the domain of performance enhancement in edge-cloud frameworks. Chapter 3 explores the allocation of tasks within an edge-cloud infrastructure, which is crucial for optimising the execution and scheduling of tasks in IoT applications. The focus is on addressing specific challenges such as inconsistent internet connectivity, communication delays, and network service disruptions, which significantly impact the performance and reliability of IoT systems. Chapter 4 takes the discussion to design a security-aware load balancing framework specifically designed for edge based infrastructure, with an emphasis on enhancing support for rural environments. Chapter 5 explores the impact of AI task execution on IoT based environments. Optimising AI and ML based tasks hold immense potential for revolutionising various industries and domains. By leveraging optimisation techniques and methodologies, significant improvements can be achieved in terms of efficiency, scalability, and performance of the frameworks. Chapter 6 explores the aspect of data management where a substantial amount of data is generated in IoT applications utilising ML, AI, and edge-cloud serverless environments and is prone to failure or data loss. Finally, Chapter 7 provides the conclusion of thesis and scope for future work. Figure 1.3 below graphically shows the organisation and structure of this thesis.

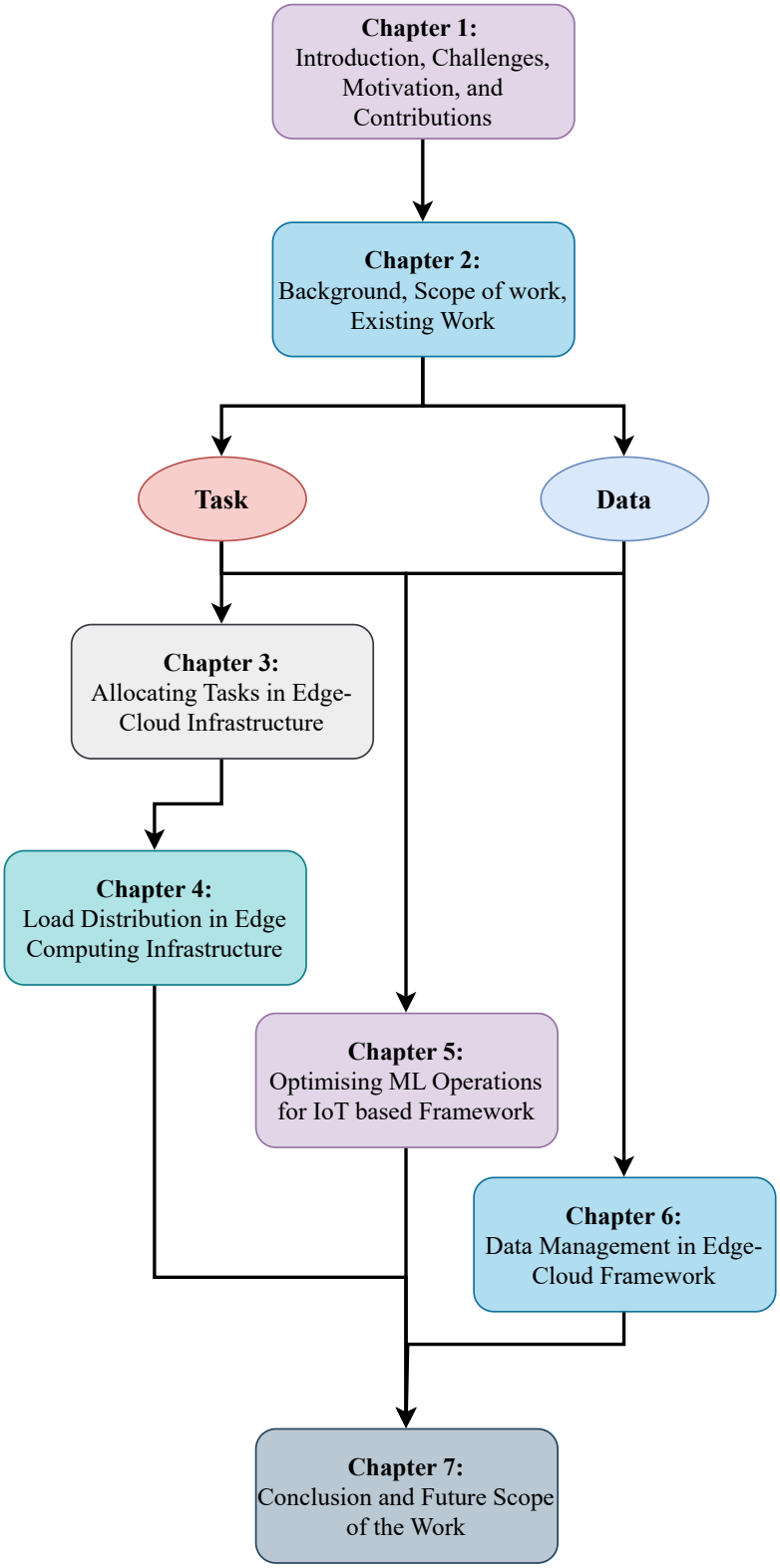

Figure 1.3: Structure of the thesis.

Chapter 2

Literature Review

This chapter provides a comprehensive description of existing literature and research towards the performance enhancement of intelligent IoT-based applications and tasks within the edge-cloud continuum. It also highlights the research efforts towards designing the applications using edge computing infrastructures. The categorisation and organisation of the literature review performed in this thesis is shown below in Figure 2.1. The affordability and easy availability of Single-Board Computers (SBC) like Raspberry Pi, Nvidia Jetson (nano), or Google Coral have made it possible to deploy on-field distributed intelligent environments. This rapid growth in affordable IoT and edge-cloud systems has resulted in the wider deployment of such systems in everyday applications. IoT infrastructures generally interconnect a diverse set of devices having sensors and actuators, that utilise communication protocols to exchange and collect data from end-users [24]. Edge and cloud computing [25] provides a task execution framework that critically handles processing and data communication for IoT based tasks. It enables processing of data at the network edge along with a central cloud node [26]. This results in faster response times, higher quality of service and improved security compared to traditional centralised servers [27, 28].

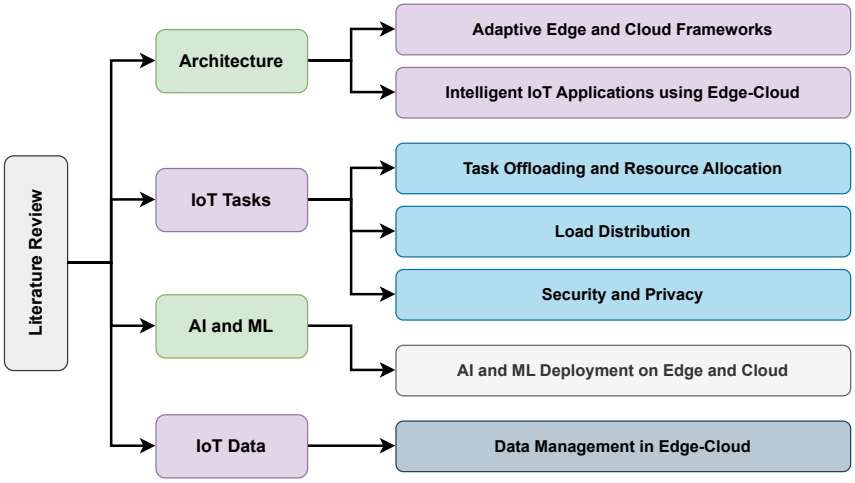

Figure 2.1: Categorisation of the literature review performed in this thesis.

2.1 Adaptive Edge and Cloud Frameworks

Nvidia Jetson and Raspberry Pi are now widely used to support edge-based AI computing. Dependable use of these devices requires addressing operational issues such

as inconsistent internet, communication delays, and service disruptions – that require proactive strategies [20]. Li et al. [29] propose an adaptive resource allocation method for edge-cloud based clusters that can reduce the service cost by adjusting the billing mechanism of resources. By evaluating the impact of shrinking resources and integrity of the data, the blocks of data on the shrink node can be migrated to other available resource nodes in the cluster before they are released. An optimised placement model for resources is designed as a data migration technique in this approach. Fault tolerance is also a crucial requirement in IoT enabled robots, especially as they need to be operated by non-experts. Researchers have extensively focused on fault-tolerance across platform and infrastructure layers, such as the use of self-adaptive systems which provide quick backups and reduced recovery times [30]. Other methods like the greedy nominator heuristic ensure service reliability through service replication [31, 18] in the framework. With the rising demand for AI in rural environments, applications require support for fault tolerance to ensure performance and efficiency in most IoT applications. Recent literature highlights advancements in weed detection and robotic weed management in agriculture, such as studying weed classification using AI [32] or adapting to rural infrastructure to securely train an AI model [33]. However, literature addressing infrastructure unreliability, especially in rural farming areas [20, 34] is limited. Even though researchers are working on optimising machine learning inference to reduce costs [35], delays [36, 35], and balance workload [36], they often lack solutions designed for IoT settings in rural areas.

2.2 Task Offloading and Resource Allocation in Edge and Cloud Systems

The rapid advancement of IoT and its applications has rapidly increased the need for real-time processing and advanced computation on end-user nodes or devices. Despite the increasing power of computing technologies, IoT devices often struggle to fulfil the increasing demands of these applications [37]. To overcome the constraints related to computation, storage, and battery life in IoT devices, computationally heavy tasks can be shifted to nodes having more resources, such as those found in cloud or edge environments. Cloud computing is a well-established infrastructure that supports this offloading of tasks on remote nodes [38]. Cloud computing offers the significant advantage of providing self-availed service and network access to these resources, irrespective of the user's location [39]. This technology facilitates widespread network access to remote resources and has seen broad application and growth alongside the expansion of cloud computing [40].

Various studies have used cloud computing to address large-scale computational challenges. For instance, Sun et al. [41] tackled the task offloading issue in vehicular clouds by creating a learning algorithm that minimises offloading delays based on historic latency data. Chen & Hao [42] explored task offloading in ultra-dense networks using software-defined networks to optimise task execution times and conserve battery life. Similarly, another

work [43] introduced a localised cloud computing model within the IoT setup, enabling the creation of ad-hoc cloud nodes through nearby computing devices for task offloading. Cloud computing also facilitates data-intensive research; Langmead & Nellore [44] utilised it for genomics data analysis, handling vast, stored sequencing data sets that require extensive computing resources. However, despite its many benefits, cloud computing can introduce significant communication delays, which can be challenging for emerging time-sensitive applications due to its centralised and remote infrastructure [38]. Bermejo et al. [45] have also highlighted some limitations of cloud computing, such as the challenges associated with processing location independence in specific networks like IoT and sensor networks, where real-time processing is essential.

With the increasing demand for time-sensitive applications and the rising volume of data, there's a need to position our resource nodes closer to where data is generated and processed. Edge computing has emerged as a solution to this challenge, offering computational and storage computing capabilities at the network's edge, closer to end devices. This setup not only reduces the bandwidth usage of primary cloud networks by allowing local offloading of tasks but also minimises latency, enhances energy efficiency, and provides robust computing power for demanding tasks [38, 37].

In recent years, methods such as fog computing, mobile edge computing, and cloudlets have been developed to operate at the edge of the network. Despite different naming, these approaches function as an intermediary layer between end-users and the cloud, offering quick access to storage, processing power and reducing the delay. Numerous studies have considered edge offloading, especially for applications sensitive to execution delays. Naouri et al. [46] proposed a three-tier offloading architecture, where tasks are offloaded based on their computational and communication needs to minimise delays. Meanwhile, traditional IoT systems, especially those handling multimedia, face significant challenges due to bandwidth constraints. Work done by Long, Cao, Jiang, & Zhang [47] addressed this by designing an edge-based architecture that groups video data to enhance the accuracy of human detection within a strict time frame. Similarly, Zhang et al. [48] explored an edge-computing framework using Unmanned Aerial Vehicles (UAVs), allowing these UAVs to process time-critical tasks for IoT devices efficiently.

However, as edge servers become overloaded with requests, they may struggle to process all tasks immediately. This can potentially lead to delays beyond the tolerable limits for IoT devices [49]. In such scenarios, a combination of edge and cloud resources can be more effective, ensuring tasks are completed within acceptable time limits even when the computational load is high. This integration between edge and cloud computing has been the focus of various research efforts aimed at optimising latency across networks. [50] and [51] studied how to jointly allocate communication and computation resources in systems that integrate edge and cloud computing. This collaborative approach extends to processing tasks not just at the edge or cloud servers, but also directly on the mobile end-user devices. Moreover, Hao et al. [52] introduced an offloading framework that utilises a cognitive engine to manage resources across three layers: the end device, edge-cloud, and

remote-cloud. This system effectively uses resources and processes tasks from intelligent applications, optimising the overall operation and response times.

2.3 Intelligent Applications using IoT, Cloud, and Edge

Chen et al. [53] introduces a novel approach towards a resource-efficient edge framework for emerging intelligent IoT applications, including ad hoc networks for precision agriculture, e-health, and smart homes. The framework focuses on maximising resource efficiency by facilitating optimal task offloading among the local device, adjacent edge nodes, and the nearby cloud node. This ensures that computationally demanding tasks are supported effectively across the network. The use of robotics in agricultural applications has also revolutionised farming practices with precision tasks, automated crop management and data-driven decision-making processes [54]. A multi-robot system usually operates in a centralised or decentralised control setting [55]. A centralised strategy is proposed by Conesa-Munoz et al. [56], that involves administering the workflow of a designed system from a primary base station. The robotic assembly may consist of either aerial or terrestrial vehicles. The complete integration of these platforms allows the execution of autonomous operations in outdoor environments. On the contrary, for decentralised cooperative control of heterogeneous robotic nodes, Dimakos et al. [57] illustrated the interaction of multiple mobile agents comprising a group of unmanned aerial and ground vehicles that allows collaborative operation of drones in a parts delivery scenario to enable the operation of the factories. Ground-based navigation is further adapted to align with the centroid of the group by utilising a Lyapunov-based optimisation approach. Another work [58] introduces a smart system for home environments that administers the various nodes and services within these settings. It activates or deactivates them according to predicted patterns of a user's service usage. This system is designed to enhance the outcomes of a deep learning classification model, particularly while the algorithm continues to learn from user interactions.

Patros et al. [59] propose a solution for rural agricultural challenges of weak connectivity and high latency. Their framework utilises a serverless framework to facilitate federated learning tasks in rural applications. The task requirements are specified by analysing the major challenges in rural agriculture communities of New Zealand. A rural-AI system for pasture weed detection is considered as the proof-of-concept for demonstration and evaluation in this work. Another work by Almurshed et al. [60] examined the ways in which edge-cloud computing can be utilised to address the reliability challenges in rural areas. A self adaptive system using an optimisation strategy called the greedy nominator heuristic is proposed that manages the allocation of federated learning tasks in a rural setting. These approaches yielded effective outcomes and provided efficient allocation of tasks. However, in order to optimally utilise all the available resources at the edge layer, load balancing and distribution also play a crucial role.

2.4 Load Distribution on Edge and Cloud Nodes

Naas et al. [61] proposed a graph partitioning-based data placement strategy for fog infrastructures. It uses a divide and conquer heuristic approach for splitting the allocation problem into sub-parts. The proposed solution reduced the task placement latency and provided a flexible, scalable, less complex method for distributing tasks in a fog network. Other researchers [62] describe a dynamic task offloading mechanism for a resource node where a task can be deployed. Their work also analyses the optimal path for offloading the task to remote fog nodes. The offloading problem is mapped to an Integer Linear Programming model that considers factors such as energy consumption, network delay and link utilisation while making the offloading decision in the framework. Oueis [63] proposed an approach with an aim to enhance user's quality of experience by handling the problem of load balancing in fog computing environments. They addressed the complex scenario where multiple users require computation offloading, necessitating the assignment of all requests through local computation cluster resources. A customisable algorithm for fog clustering is designed that establishes small cell clusters with low complexity and ensures optimised resource management. In another work, researchers [64] introduce a method that handles the load distribution limitations by (i) incorporating load-balancing capabilities directly into the soft network edge, such as virtual switches, which eliminates the need for modifications in the transport layer, customer virtual machines, or network hardware, and (ii) implementing load balancing using finely segmented, nearly uniform data units that align with end-host offload techniques to support high-speed networking. They developed and implemented this load balancing approach (named it Presto) and assessed its performance on a 10 Gbps physical hardware testbed.

Due to the limited computational power of mobile devices (MDs) and the varied and constrained resources of cloudlets, a three-objective model has been developed to simultaneously optimise the time and energy consumption of MDs [65], as well as the load balancing across cloudlets. The authors introduced an efficient method for multi-user, multi-application computational offloading in environments with multiple cloudlets, utilising an enhanced version of the non-dominated sorting genetic algorithm III. A brief comparative analysis of different load distribution approaches is shown below in Table 2.1. Maia et al. [66] explore the idea of load distribution and service placement using a multi-objective meta-heuristic algorithm. They utilised a concept of heuristic initialisation that selects an initial population, with an aim to improve the efficiency of the genetic algorithm. The method is designed to give priority to latency-sensitive applications while simultaneously optimising conflicting objectives. These proposed approaches outperform state of the art algorithms, however, there are other factors such as security and failure risk that can still improve the performance of load balancing and distribution systems. Table 2.1 below shows the comparison of factors considered in existing approaches and our proposed approach SHIELD described in Chapter 4.

Work	*ML Tasks*	*ML Pipeline*	*Security Evaluation*	*Load Balanced*	*Use-Case*	*Failure Examined*	*Serverless Evaluation*	*Workload Management*
[24]	inference	centralised	no	yes	smart city	no	no	nil
[26]	training	distributed	no	no	industry	no	no	nil
[56]	none	none	no	yes	agriculture	no	no	nil
[67]	none	none	no	yes	UAVs	no	no	lyapunov analysis
[59]	training, aggregation	distributed	no	no	agriculture	no	yes	nil
[61]	none	none	no	yes	smart city	no	no	heuristic/graph modelling
[62]	none	none	no	yes	nil	no	no	integer linear program
[68]	none	distributed	availability	no	smart traffic	yes	no	greedy heuristic
[69]	none	none	confidentiality, integrity	yes	nil	no	no	nil
[70]	none	none	no	yes	nil	yes	no	formal modeling
SHIELD	inference, training, aggregation	distributed	confidentiality, integrity, availability	yes	agriculture	yes	yes	meta heuristics

Table 2.1: Comparison of various studies focusing on ML workloads, security, load distribution, and task management in edge-based IoT applications.

2.5 Security and Privacy in Edge-Cloud based Systems

Over recent years, IoT-based systems have started to use ML and AI to improve the efficiency and sustainability of our everyday applications. However, these mechanisms also introduce new security risks and vulnerabilities. Vangala et al. [71] and Zanella et al. [72] highlight the security challenges and issues in smart agriculture. They highlight the importance of designing a comprehensive security infrastructure to support smart agriculture. Wiseman et al. [73] also analyse the reluctance of farmers to share their agri-data and how this is affecting smart agriculture. Another work [74] introduces a framework that includes key security processes for cloud computing utilised in the healthcare sector. This framework begins with collecting general information for security management processes and identifies critical information security processes specifically for healthcare organisations that utilise cloud computing, taking into account the principal risks associated with cloud services and the types of information processed. The framework aims to guide healthcare organisations in prioritising essential ISMS processes, helping them to develop and maintain these processes despite resource constraints. He et al. [75] conducted a comprehensive analysis of the potential for attacks and privacy protection in edge-cloud collaborative systems. They develop a series of new attacks that enable an untrusted cloud to retrieve any inputs entered into the system, regardless of the attacker's access to edge device data, computations, or system querying permissions, and secondly they empirically showed that traditional noise-adding solutions are inadequate against their specified identified attacks leading them to propose two more robust defense strategies.

Researchers [76] also introduce an edge-based framework for data collection where raw data from wireless sensor networks (WSNs) undergoes differential processing by algorithms on edge servers to support privacy-centric computing. In this model, only a minimal portion of critical data is retained on edge and local servers, with the remainder sent to the cloud for storage. This approach offers dual advantages: firstly, it enhances data privacy by ensuring that original data cannot be reconstructed, even if the data stored in the cloud is compromised. Secondly, by adopting a differential storage strategy, the edge-based model transmits less data to the cloud, thereby reducing communication and storage costs compared to conventional methods. Recent developments in load distribution [66, 77, 78, 70] and security [79, 80] identify possible approaches for managing intelligent applications in rural areas. However, existing research still lacks focus on security-aware load balancing infrastructure which considers completion time, task security, load distribution, and failure risks; especially relevant in a rural environment which is more prone to faults and has limited resilience.

2.6 Challenges with AI Sustainability and IoT

In order to understand and analyse the problem of sustainable AI deployments, it is crucial to examine the challenges described by researchers in this domain. The following subchapters highlight the recent work done in the field and discuss their impact and potential on sustainability.

Incorporating AI into the IoT provides high efficiency and intelligence to systems, enabling devices to process data autonomously. This integration enables devices with the ability to make decisions, optimise operations, and provide insights without direct or indirect human intervention. However, utilisation of this potential requires overcoming significant challenges, particularly in adapting AI technologies to the diverse environments where IoT devices operate. Mhaisen et al. [81] provide a survey of recent techniques and strategies designed for handling AI tasks in IoT applications. Another work [82] focuses on security techniques based on ML describing how it can be used for enhancing security in IoT systems. ML-based techniques for authentication, malware detection, offloading, and access control are mainly focused in this work. Bu et al. [83] presents an agriculture system for IoT that utilises AI and cloud computing for making smart decisions such as determining the amount of water needed for irrigation on the fields.

As IoT networks expand, so do the computational, memory, and energy demands of AI models used with them, necessitating innovative approaches to manage and reduce this consumption. Larger AI models require very high computational power and memory, leading to increased energy consumption during both training and inference phases. The work by Canziani et al. [84] and Li et al. [85] analyses the trade-offs between model size, performance, and energy efficiency; illustrating how larger models, while potentially more accurate, can significantly have high energy consumption and prominent impact

on the environment. The size of an AI model also affects its deployability in real-world applications, especially in resource constrained environments. Large models may not be feasible for deployment on mobile devices or in edge computing scenarios, where energy efficiency is a critical performance factor. This limitation challenges the scalability of AI solutions and their ability to be deployed sustainably across a diverse range of IoT platforms [84].

Many efforts to create model architectures such as EfficientNet by Tan and Le [86, 87] demonstrate the potential to reduce the impact of large AI models. These architectures aim to maintain or improve performance while reducing the computational demands and energy consumption, addressing the sustainability concerns associated with model size. Hu et al. [88] explore a novel channel pruning method which can be used for compressing large AI models. Another work [89] evaluates the efficiency of model compression within the context of energy-efficient inference. Chen et al. [90] utilise the low-rank approximation to eliminate the redundancy within filter and accelerate the deep neural network pruning process. The researchers [91] proposed a mechanism which identifies the sensitive input values that are highly correlated to the accuracy and applies the low-fidelity quantization on non critical regions to boost the execution performance. Wu et al. [92] propose a method based on dynamic gradient compression and knowledge distillation to execute the federated learning tasks efficiently.

Along with optimisation, another crucial challenge is to extend sustainable AI development beyond optimising model efficiency and training time to cover environmental and societal impacts. This requires including sustainability in AI's lifecycle, adopting new optimised techniques and cultivating a culture that prioritises sustainability, including transparent reporting of environmental impacts and adopting energy-efficient practices. Addressing the sustainability issues related to AI model training and size is vital for its performance advancement, ensuring its societal benefits while minimising environmental damage. The challenge, as illuminated in [93], revolves around designing machine learning models that emphasis on model architecture and strategic size reduction highlights the importance of ensuring efficiency during the early design stages of AI development, aiming for high-performance AI methodologies that provide optimal operation within limited resource conditions.

In deep learning, optimising models is important, particularly when considering the computational cost of training, which is heavily influenced by floating-point operations. Strategies such as quantization, pruning, and layer freezing can serve as important approaches to mitigate these costs and enhance overall performance. A strategic approach for reducing floating point operations (FLOPs) involves converting data from 32-bit floating-point precision (float-32) to 8-bit integer precision (integer-8), followed by performing critical operations in integer-8 and subsequently restoring the output to float-32. This method effectively optimises computational efficiency and memory usage while maintaining the accuracy crucial for deep learning tasks. The effectiveness of this quantization approach has been thoroughly explored with machine learning frameworks

such as TensorFlow, PyTorch – offering specialised tools to facilitate this process.

Based on the concept of computational efficiency, pruning is another strategy for refining model's structure [94]. By targeting and removing less impactful parameters rather than adjusting model precision, pruning enhances efficiency by focusing on the redundancy of features. This technique maintains the model's integrity by emphasising on structural optimisation instead of numerical adjustments of weights and biases. The pruning process reduces the computational complexity, operational costs, and ensures a balance between preserving model performance and achieving efficiency gains. Layer freezing, in addition to quantization and pruning, allows a strategic selection to either train or freeze neural network layers, impacting the learning process without altering any internal parameters. This technique, often achieved through hyperparameter tuning, reduces the number of calculations and FLOPs, thereby reducing computational load and energy consumption of the models. Layer freezing improves training efficiency by allowing training of specific layers and freezing rest of the layers, thereby presenting another approach for optimising AI models.

2.7 Data Management in Edge-Cloud Frameworks

Distributed systems for storage are critical in managing the vast increase in generated data by providing essential support for scalable and reliable data storage solutions [95]. The integration of Erasure Coding (EC) into these systems is crucial for reconstructing the missing data with some added redundant information [96]. It enhances the distribution of data while reducing storage demands and improves durability, making it a highly effective choice for large-scale storage infrastructures. It also surpasses traditional storage methods by enhancing time efficiency, ease of use, and fault resilience. By distributing data across several devices and creating redundant segments, EC maintains data integrity even when individual devices or nodes fail, thus minimising the likelihood of data loss. Wang et al. [97] analyse distributed storage systems and identify issues with replication and recommends EC as a solution for occurring security concerns, data placement, transfer costs, and the expenses associated with maintaining synchronised copies of data. Opara-Martins et al. [98] utilise cloud technologies to manage large volumes of social media data by proposing a multi-node OpenStack cloud system to deliver Hadoop as a service, facilitating its integration within the OpenStack environment. Song et al. [99] introduces FACHS (File Access Characteristics-based Hybrid Storage), a dynamic hybrid storage system that enhances storage redundancy, parallel read/write performance, and storage capacity by combining Reed-Solomon Code, multi-copy, and Locally Recoverable Codes (LRC) based on file access patterns, thereby enhancing efficiency, speed, and overall storage utilisation of the system.

Darrous et al. [100] compare replication and erasure coding algorithms in distributed file systems, focusing on data availability and reliability. They address EC performance challenges by proposing a buffering and merging mechanism to handle encoded

I/O requests and by developing recovery strategies to tackle decoding issues in EC-based distributed file systems. Abebe et al. [101] explore erasure encoding techniques, particularly reed-solomon coding, to identify optimal parameters for reliable and cost-effective storage systems, considering factors such as storage overhead, data accessibility, and retrieval efficiency. Another work [102] present StreamLEC, a fault-tolerant stream machine learning system that employs erasure coding for low-redundancy proactive fault tolerance, enhancing performance in failure recovery and reducing latency. Nachiappan et al. [103] proposes a recovery strategy using EC in data centres to enhance reliability with minimal overhead, optimising data block selection for proactive replication based on failure predictions leading to a significant reduction in storage costs, network usage, and energy consumption.

Chapter 3

Task Allocation in Edge-Cloud Infrastructure

This chapter introduce and explore the allocation of tasks within an edge-cloud infrastructure, which is crucial for optimising the execution and scheduling of tasks in IoT applications. I have considered the weed identification (image classification) task in precision agriculture as an application use case in this chapter.

3.1 Weed Management Task in Precision Agriculture

With a rapid increase in global population, there is an ever-growing need to ramp up sustainable food production. The Food and Agriculture Organisation (FAO) estimates that healthy diets are still unaffordable to over 3B people across the world, and a majority of these people are from low and middle income countries [104]. Efficient weed management in agricultural fields is an important factor that can improve crop productivity. Due to the spatial and temporal heterogeneity of weeds/plants in agricultural fields, many robotic weed control methods (aerial and ground) have been developed for site-specific weed management [105].

The effective use of robotic technology can benefit farmers by improving crop yield and reducing production costs, but it faces many challenges in real-life rural environments. Two challenges that affect the performance of such systems are: network availability and reliability. While network availability ensures that the infrastructure is available and operational at all times, reliability ensures that the infrastructure has been successfully deployed, and is operating error-free [106]. It is expected that a reliable network will maintain a high standard of service, even in the event of system faults [107]. This allows a system to operate without interruption when one or more faults occur. Fault detection and automated correction play a key role in supporting a system's fault tolerance.

Moreover, the utilisation of edge computing resources can also considerably benefit rural infrastructure, as it provides many advantages like low latency, distributed architecture, security and support for real-time execution. These benefits enable the use of edge-cloud infrastructure in many real-world agricultural settings that utilise IoT based systems.

One such application, namely Rural-AI, allows the use of ML within rural communities, with an aim to achieve better performance in terms of productivity, economic growth, impact of climate change and affordability [108]. Rural-AI [20] is the *engineering of cyber-physical systems for enabling sovereign, sustainable AI in locations with limited*

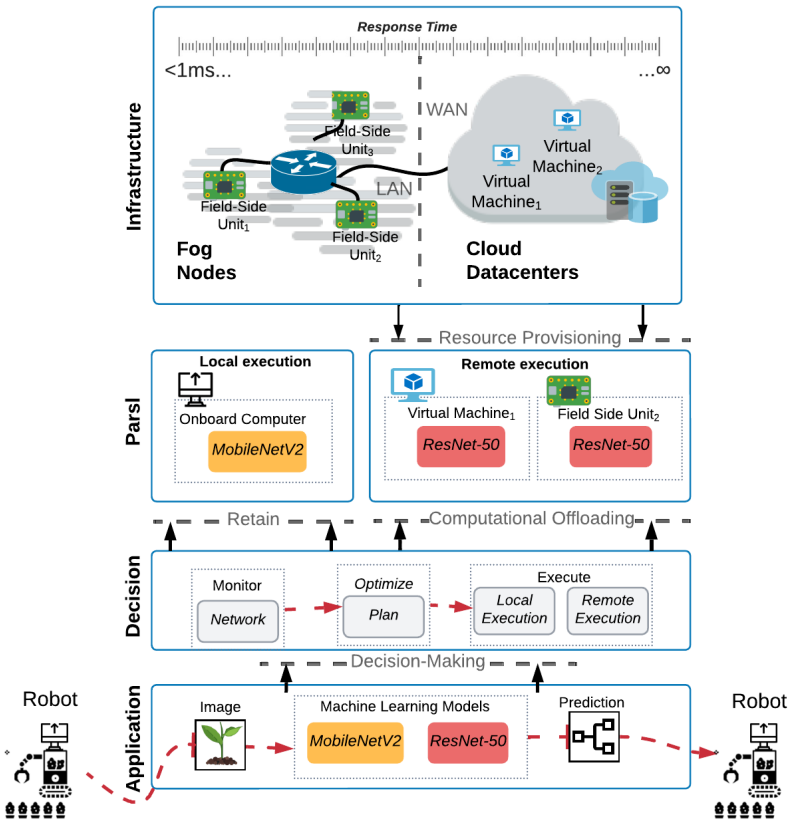

Figure 3.1: Architecture for the task allocation framework.

and/or unreliable power/networking infrastructure. Due to the lack of proper development and infrastructure in rural areas, ML/AI applications cannot be utilised to their full potential. Service outages along with unreliable network connectivity are two key challenges that significantly affect the growth of rural economy. These issues prevent IoT infrastructure from being efficiently deployed and used [109]. Therefore, there is a need for a framework that is capable of providing a *reasonable quality of service* (QoS), even when a connection failure occurs.

A framework for edge-cloud infrastructure that detects connection errors and adapts to such errors is proposed. It also triggers a fault tolerance mechanism to ensure that computational/AI tasks are executed with a minimal loss to performance, even when a network fault occurs. The architecture of our proposed framework is depicted in Figure 3.1. To demonstrate the efficacy of this conceptual architecture, two ML models based on *ResNet-50* and *MobileNetV2* have been trained for identifying weeds, using images captured from a robot mounted camera. Light versions of these two models have been generated respectively using model quantization techniques – a process involving size reduction of the learned model, at the expense of accuracy, e.g. by mapping model parameters from floating-point numbers to low-precision fixed-point numbers [110].

Our primary goal is to prioritise the use of highly accurate pre-trained models for inference. As this inference is carried out remotely on computational units of the farm site, it can take a longer time to execute. However, if we are unable to connect with field side units

due to network faults, local models are utilised for inference. While this local model is moderately accurate, its execution time is significantly lower than the full model. An algorithm based on task deadlines is utilised to determine the location for inference. The task execution is then evaluated using a testbed created for this framework. The main contributions of this chapter are as follows:

- formulation of a real-time image classification problem with intermittent network connectivity. This problem is then mapped to a weed detection use case – widely considered significant in precision agriculture.
- performance analysis of two ML models trained for plant/weed image classification. A rule-based algorithm is also formulated for decision making, taking account of where to perform this classification: locally or remotely.
- development of a testbed consisting of: Raspberry Pi node (RPi), a laptop computer, and a cloud server – to benchmark the performance of the proposed framework.

3.2 System Model

This subchapter includes a description of software systems used within our proposed infrastructure. A case study which makes use of this infrastructure is also described.

3.2.1 Serverless Computing Platforms

A number of serverless platforms are available for evaluation – ranging from those that are: (i) used commercially, such as [1]Amazon Lambda, Google functions, Azure functions, etc; (ii) available as open source systems, such as Apache OpenWhisk, [2]Fissions, [3]OpenFaaS, etc. Some variants include pre-deployed commercial versions of open source platforms, e.g. OpenFaaS Pro, which offers additional features and support.
These platforms differ in the types and range of capabilities they offer, for instance, some utilise an existing pre-deployed platform (e.g. Kubernetes) enabling users to write and offload executable functions. These types of platforms enable users to build and manage their own functions, rather than the infrastructure on which these functions are hosted. Others include support for deploying (and managing) the hosting environment on which these functions are executed (e.g. OpenWhisk). *Parsl* [111] provides a Python-based development environment for functions, and can be hosted on both edge devices (e.g. RPi) and on a high performance computing cluster. This is achieved through the use of custom executors designed for the resource being used in the architecture. *Parsl* also provides the basis for dynamically distributing functions to new devices, using a controller node. A key benefit of *Parsl* is the ability to develop a heterogeneous function hosting environment, which can be modified at run time, especially when node failures occur. A reference to

[1] *https://aws.amazon.com/lambda/*
[2] *https://fission.io/*
[3] *https://www.openfaas.com/*

functions deployed across *Parsl* nodes is hosted in a registry, enabling these references to be updated as new instances of functions are deployed. Lean OpenWhisk represents a streamlined adaptation of the conventional OpenWhisk platform, optimised specifically for serverless frameworks within edge computing environments. Distinctively, Lean OpenWhisk demands fewer resources compared to its original counterpart, incorporating only the fundamental modules essential for executing serverless operations. Instead of Kafka, Lean OpenWhisk utilises an in-memory queue structure, substantially diminishing its overall framework footprint. Moreover, it adopts a more integrated design by placing the Invoker in close proximity to the Controller module. This strategic design adaptation enhances its efficacy on devices with limited resources, such as RPi or Nvidia Jetson, which are frequently deployed in edge or IoT settings. A significant limitation of Lean OpenWhisk is its exclusive compatibility with the x64 and x86 infrastructures, omitting native support for the ARM architecture predominant in RPis. To bridge this gap, I've devised a Docker-based solution, adapting Lean OpenWhisk for deployment on RPi devices built on the ARM framework.

3.2.2 Dataset and ML Models

DeepWeeds [112] is a multiclass image dataset for deep learning consisting of 17,509 images. These images belong to 8 different categories of weeds found in vast regions of northern Australia. This dataset contains 15K training images and 2.5K test images of size 256x256 pixels. Two different ML models: *ResNet-50* [113], and *MobileNetV2* [112] have been trained on the DeepWeeds image classification dataset and utilised for plant/weed identification in this chapter.

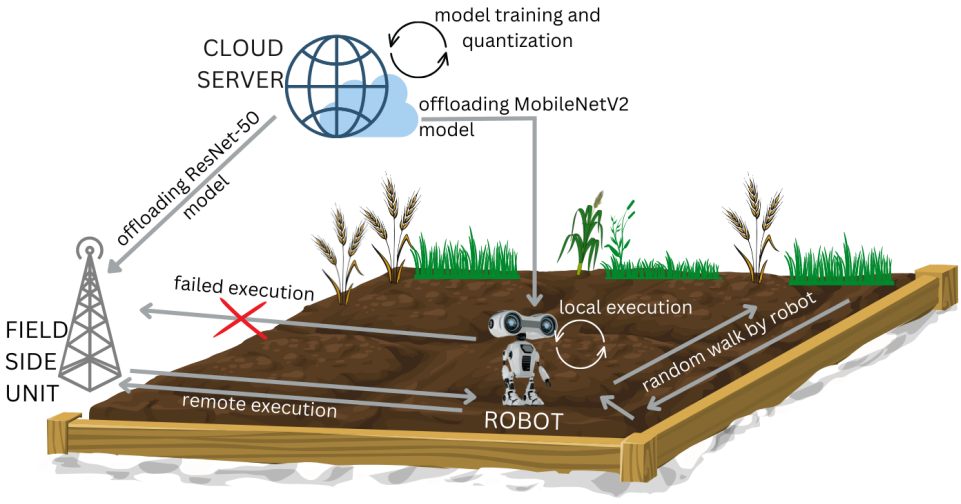

Figure 3.2: Interaction between robot, FSU, and cloud node in the agricultural field.

3.2.3 System Design and Use Case

I develop a three-tier architecture comprising of IoT devices, storage, and computation resources positioned in a hierarchical manner. The control flow of the framework is based

on a master-worker configuration. I consider a rural agriculture use case where robots identify and remove weeds. Robots interact with Field Side Units (FSUs) that are located in proximity to a robot on the same field.

Robots act as the master node, whereas the FSUs are the worker nodes. An interaction between the robot, FSU, and cloud server is depicted in Figure 3.2. Robots move throughout the field by following a 2-D random walk trajectory. Each robot is equipped with a camera to take images of nearby plants. To evaluate whether the image captured is of a weed or a plant, there are two options: the robot can either perform inference locally or forward the data to the on-farm FSU for remote inference. This decision is taken by a robot in real time on the basis of the network quality between robot and FSU and the computational capacity available on the robot. If the plant is identified as a weed, the robot will initiate the weed removal process, otherwise, the robot will move to a different location and repeat the process.

3.3 Problem Formulation

A mathematical formulation of our task-to-resource allocation problem is presented in this subchapter. I have designed an Integer Linear Programming (ILP) model that offloads machine learning inference tasks on resource nodes for execution. The decision on where to perform the execution, local node or remote node, can be achieved by optimising the objective functions later described in this subchapter.

3.3.1 Monitoring Constraints

Let M be a set of ML models that are used for performing the inference tasks in this framework. These ML models can be represented mathematically as $M = \{m_1, m_2, m_3, ..., m_p\}$. Let R be a set of resource nodes where the ML models can be deployed for executing the inference tasks. These resources can be mathematically written as $R = \{r_1, r_2, r_3, ..., r_n\}$. I have mainly considered two types of resource nodes in this work, a robot and a FSU. Each resource node r is mapped to one specific ML model m which is used for performing inference on that resource node. The optimal decision of determining a location for inference can be affected by numerous factors such as network quality, resource availability, and ML model characteristics. Therefore, multiple factors for monitoring the resources and ML models have been considered in this chapter. The following constraints are utilised for –

a) resource monitoring: signal quality, uplink/downlink speed, execution rate, available RAM, and storage.

b) ML model monitoring: number of neural network layers, accuracy on test dataset, and quantisation feasibility.

3.3.2 Decision Variables

In order to manage all the monitoring constraints (specified earlier) and make a decision on where to execute the task, the ILP model initialises a set of variables which are used for forming the decision. To ensure that one inference task is offloaded to only one resource node, I have formulated a binary variable x_{i} such that:

$$x_i = \begin{cases} 1, & \text{if inference is performed on node } i \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

where $i \in 1, 2, 3, ..., n$. I have reserved $i = 1$ for local execution on the robot whereas $i = 2, 3, ..., n$, are reserved for remote nodes (FSUs) available for execution.
Similarly, two other variables y_{i} and z_{i} are introduced to manage signal quality and availability of resource nodes respectively. They are represented mathematically as:

$$y_i = \begin{cases} 1, & \text{if } q_i \geq \tau \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

where $i \in 2, 3, ..., n$; q_i is the signal quality at node i, and τ is the signal quality threshold. Here, $i = 1$ is not considered because connection setup is not required for local executions. Variable z_i ensures that the task is only allocated to a resource node if the node is not busy with some other inference task. It can be given as:

$$z_i = \begin{cases} 1, & \text{if node } i \text{ is free} \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

Another important factor that affects the execution time of inference tasks is communication or transmission delay. The transmission time for performing inference on FSU can be evaluated as the ratio of data size transmitted to the transmission rate of link. This includes the total data transmitted in both up-link and down-link for ML inference. Therefore, total approximated transmission time for the framework can be written as:

$$t_i^c = \frac{d_q}{s_u} + \frac{d_o}{s_d} \tag{3.4}$$

where d_q, d_o are the input and output data size of m^{th} inference task. The uplink, downlink speed for offloading the inference task and fetching back the results are given as s_u, s_d respectively. For location $i = 1$, network transmission time is considered to be 0. This is because the location has been reserved for local execution and does not require any transmission of data.
The average time it takes to transmit data depends on the quality of the network connection. If the quality of signal is 100%, speed will be equal to the actual link speed,

otherwise it will be reduced depending on the signal quality. Therefore, the approximated link time for data transmission to i^{th} node can be given as:

$$t_i^l = s_l / q_i \tag{3.5}$$

where s_l is the link speed of network and q_i is the current signal quality to reach node i. The total execution time for establishing the connection and transmitting the data in this framework can be given as:

$$t_i = t_i^c + t_i^l \tag{3.6}$$

In order to determine the size of ML model, I have calculated the number of arithmetic operations in each layer j of neural network that has been deployed at location i. It is described by n_{ij} such that

$$n_{\text{ij}} = \prod_{x=1}^{w} e_{ijk} \tag{3.7}$$

where e_{ijk} is the k^{th} (out of w) element in j^{th} layer of ML model deployed at location i. If l_{ij} is the time it takes to process j^{th} layer of the model at location i then it can be given as

$$l_{ij} = \frac{n_{ij}}{p_i} \tag{3.8}$$

where p_{i} is the rate of processing the model at location i. Therefore, from the previous three equations, I observed that overall time taken to perform a prediction on location i can be given as

$$t_i^p = \sum_{j=1}^{w} (l_{ij}) + t_i \tag{3.9}$$

The accuracy of ML inference task is another crucial aspect to consider when evaluating the performance of this framework. It can be given as the ratio of the number of correct inferences with total number of inferences. Accuracy in this framework can be represented as:

$$A_i^m = p_{cr} / p_{tl} \tag{3.10}$$

Here, p_{cr}, p_{tl} are the correct and total number of inferences recorded at location i.

3.3.3 Objective Function

The objective of this chapter is to minimise the total cost of executing all the ML models at their respective locations while ensuring high performance even if a network fault occurs. This can be achieved by minimising the execution time and maximising the accuracy for performing an ML inference. Therefore, the optimisation problem that needs to be solved

can be represented as:

$$\min \left(\sum_{i=1}^{n} (x_i * y_i * z_i * t_i^p) \right) \tag{3.11}$$

$$\max \left(\sum_{i=1}^{n} (x_i * y_i * z_i * A_i^m) \right) \tag{3.12}$$

subject to constraints

$$\begin{aligned}
&x_i \geq 0 \quad \& \quad x_i \leq 1, \quad \forall i \in \{1, 2, 3, \ldots, n\} \\
&y_i \geq 0 \quad \& \quad y_i \leq 1, \quad \forall i \in \{2, 3, \ldots, n\} \\
&z_i * q_i \geq \tau, \quad \forall i \in \{1, 2, 3, \ldots, n\} \\
&x_i * y_i * z_i \leq 1, \quad \forall i \in \{1, 2, 3, \ldots, n\} \\
&ram_i + str_i \geq d_q + d_o, \quad \forall i \in \{1, 2, 3, \ldots, n\}
\end{aligned}$$

In the case of real-world scenarios, it is difficult to achieve the optimal allocation of resources for large-scale problems. Therefore, there is need for a near-optimal solution that can still meet the problem requirements and balances the trade-off between solution quality and cost of execution. The objective of our work is to develop an approximate heuristic approach that allocates tasks to edge-cloud resources while achieving a balance between minimising inference time and maximising the accuracy of used ML models.

3.4 Proposed Approach

The chapter propose a fault diagnostic mechanism that takes the network quality into consideration whenever a task is executed by a robot. A brief summary of the ML models, execution workflow, and decision algorithm for the proposed framework is provided in this subchapter.

3.4.1 ML Models

Two ML models are considered: *ResNet-50* and *MobileNetV2*. *ResNet-50* is considered a good model for image classification because of its layer depth, residual connection, ease for transfer learning, and good performance. Due to the large number of hyperparameters, full models based on *ResNet-50* are capable of handling complex image classification tasks. However, this comes at the cost of high execution time and computational requirements. Both computational capacity and storage are assumed to be limited in edge environments. TensorFlow Lite is used to perform model quantization and generate a light version of *ResNet-50*, trained on all eight classes of weeds available in the DeepWeeds dataset.
The whole DeepWeeds dataset is gathered from a vast region in northern Australia,

although it is highly unlikely to find all eight weed classes within a single geographic region. Another model is based on *MobileNetV2* on three classes of weeds available in the DeepWeeds dataset: Chinee Apple, Lantana, and Snake weed. I have assumed that only these 3 classes of weeds are found in the specific geographical region. In general, each geographical region will have a different model trained on specific categories of weeds that occur in that region. A light version of *MobileNetV2* model is used for local inference on the robot.

3.4.2 Signal Monitoring

A signal monitor continuously examines the network connectivity between the robot and the FSU. I have divided the 'signal' parameter into three different categories: no signal, low signal, and strong signal. When the robot is unable to establish connection with a FSU, it is considered to have no signal, no distortion between the robot and FSU indicates a strong signal, and low signal falls between these conditions. During our experiments, when the signal strength drops below a threshold value, it is considered as a low signal; otherwise the signal is considered to be a strong signal.

3.4.3 Execution Workflow for Inference Tasks

The light model based on *MobileNetV2* is used for local inference on the robot, whereas the light model based on *ResNet-50* has been used for remote inference on an FSU. *MobileNetV2* offers an accuracy of 62.65%, whereas *ResNet-50* offers an accuracy of 88.64%.

Our deadline-aware approach starts by initially checking the network connection. Depending on the current network signal quality, a deadline is established for each task and the inference process is initiated if the connection is available. The estimated deadline is the approximate time it takes to perform ML inference on a FSU, if the current signal quality is maintained throughout the execution of task. It is calculated by considering the execution time on a FSU, transmission time and link time for completing the inference. The algorithm offloads ML tasks to remote FSUs and starts a timer. If the outcome of inference task is not retrieved before the deadline, the task is immediately discarded. Inference is then performed using a local model. I have assumed that if the results are not retrieved within the deadline, a network fault might have occurred, thereby causing the delay. Executing a model locally ensures that a reasonably accurate result is achieved, even when the inference fails on the FSU. The proposed deadline-aware approach is described in Algorithm 3.1.

Another signal quality-aware approach for the proposed fault tolerance mechanism can be considered. The signal quality of the network is repeatedly monitored and a threshold value is fixed before the execution begins. If signal quality is found below the specified threshold value, it is considered to have low quality for performing the FSU-based execution and inference is directly performed on a local node. This is because a low signal increases the probability of task failure in case any network fault occurs. By adjusting the threshold

Algorithm 3.1 Deadline-Aware Approach

```
1: procedure DEADLINE-AWARE()
2:     (FSU: Field Side Unit)
3:     begin connection setup [robot ↔ FSU]
4:     if connection setup → success then
5:         inference task → calculate deadline
6:         inference (FSU) → start
7:         timer → start
8:         if result [robot ← FSU] & timer ≠ finished then
9:             prediction (FSU) ← success
10:        else if timer = finished then
11:            prediction (FSU) → discard
12:            inference (local) → start
13:            prediction (local) ← success
14:        end if
15:    else connection setup → failure
16:        inference (local) → start
17:        prediction (local) ← success
18:    end if
19: end procedure
```

value, the sensitivity of model performance can be controlled in the framework. In a real-life scenario, the threshold can be specified based on the network requirements of that application.

3.5 Experiments and Simulation

I evaluate the performance of our weed detection model. Multiple faults introduced in the model are captured in the simulation setup and tested over multiple iterations.

3.5.1 Experimental Setup

The experiments have been evaluated on an edge-cloud environment that consists of one cloud node and multiple edge nodes connected over the internet. A Google Cloud Platform (GCP) server is utilised to host the cloud node. This is an NVIDIA Tesla T4 GPU Computing Accelerator, 16GB GDDR6, 585MHz 2560 CUDA cores with PCIe 3.0 x 16. The edge nodes are Raspberry Pi (RPi) 4 Model B, Quad core Cortex-A72 (ARM v8) 64-bit SoC, 1.5GHz, 4GB LPDDR4 RAM, 64GB storage. Both edge and cloud nodes are connected to a FSU – a Dell Latitude 5420 laptop, Core i7-1185G7, 3.00GHz, 8–16GB RAM and 512GB storage running a 64-bit Ubuntu 22.04.1 LTS. I also make use of an open source serverless platform, utilising Lean OpenWhisk Invokers running with a maximum of 3GB memory.

Figure 3.3: Weed image: (a) original (b) blurred (c) black patched.

3.5.2 Testing ML Model Capabilities

In a real-world application, it is not possible for the robot to capture perfect images of plants and weeds all the time. Sometimes, due to extraneous factors such as weather conditions and crop density, the images captured are either unclear or have obstructed line of sight. This results in unfavourable conditions for model evaluation and can affect system performance and efficiency. To test the ML model capabilities for environmental variations, I have intentionally injected noise in the images and then performed evaluation of the generated models. The noise has been injected in images in mainly two forms: blur and black patch. The blur function is applied on the entire image, whereas two random size black patches are applied at a random location in the image. The blurred image can simulate a situation when there is dust, rain drops on the camera lens that make the entire image unclear. In a similar manner, the black patched image can replicate a scenario when a leaf/insect is covering part of the lens or there is a stem obstructing the line of sight to the actual object. A sample weed image with blur and black patch is described in Figure 3.3. It is important to note that I have not performed any training using noisy images. However, blur and black patch noise were injected in all the test images of DeepWeeds dataset, and then *ResNet-50* and *MobileNetV2* were used for evaluation. The accuracy observed with and without noisy images for *ResNet-50* and *MobileNetV2* models are shown in Table 3.1.

Table 3.1: Model accuracy with and without noisy images.

	w/o noise	***with blur***	***with black patch***
ResNet-50	88.64 %	52.05 %	57.77 %
MobileNetV2	62.65 %	39.36 %	42.91 %

For a real-life application where environmental variations cause unfavourable conditions for ML model inference, it can be noticed that *ResNet-50* is a better option for performing task evaluation than *MobileNetV2*.

3.5.3 Node Selection for Training and Inference

I measured training time for one epoch of weed identification model using one image of DeepWeeds image classification dataset and analysed the performance on edge, FSU, and cloud nodes. The benchmarked results for model training are described below in Table 3.2.

Table 3.2: Time for training 1 epoch on 1 image.

	ResNet-50	***MobileNetV2***
Cloud	10.71 sec	04.15 sec
FSU	43.36 sec	16.19 sec
Edge	142.23 sec	38.64 sec

It can be observed that the training time on edge node, FSU is 4x and 10x more respectively, than the cloud node, for both models. Therefore, I have selected the cloud node for training our two ML models and generate light versions of these models. It takes around 64 sec, 42 sec to generate light models of *ResNet-50* and *MobileNetV2* respectively. A stress test to analyse the performance capabilities of two models on an edge node and FSU is also presented in this subchapter. I ran concurrent inferences of weed identification tasks and observed the change in their waiting and execution times. Tables 3.3 and 3.4 describe the average waiting and execution time for one inference when n concurrent inferences are performed.

Table 3.3: Concurrent ML inferences on the edge node.

No. of tasks	***ResNet-50***		***MobileNetV2***	
	Waiting time	***Execution time***	***Waiting time***	***Execution time***
2	18.76 sec	05.36 sec	11.30 sec	0.52 sec
4	21.25 sec	15.90 sec	23.33 sec	1.00 sec
6	33.66 sec	31.94 sec	38.97 sec	1.37 sec
8	53.80 sec	35.62 sec	50.97 sec	1.80 sec
10	∞	∞	79.05 sec	2.08 sec
14	∞	∞	124.59 sec	5.43 sec

The symbol ∞ is used to describe the event when inference is unable to complete due to system crash, or other execution failure (as it did not terminate). I was able to achieve 8, 14 concurrent executions for *ResNet-50* and *MobileNetV2* models respectively on the edge node, whereas on the FSU, the number of successful concurrent executions observed was 34 and 40 respectively. I also noticed that waiting and execution time on the edge node (robot) is much higher than FSU for both models. In terms of a real-life application, where I might have to perform multiple executions at the same time, *MobileNetV2* is a better option for task execution.

If I consider the following four factors for decision making: (1) model adaptation to a harsh environment, (2) number of concurrent executions possible, (3) average execution time, and (4) average waiting time, there is a need to establish the trade-off between

choosing a more accurate model or reduce time to develop a model. Therefore, in this work, I have decided to deploy the highly accurate *ResNet-50* model on the FSU (as a primary option), but if the network connection is unreliable/ unavailable, I can resort to utilising the *MobileNetV2* model for local inference, which offers faster processing than the former (with reasonable accuracy).

Table 3.4: Concurrent ML inferences on the FSU.

No. of tasks	***ResNet-50***		***MobileNetV2***	
	Waiting time	***Execution time***	***Waiting time***	***Execution time***
2	1.65 sec	0.20 sec	1.24 sec	0.02 sec
6	1.98 sec	0.22 sec	2.14 sec	0.02 sec
12	3.91 sec	0.47 sec	3.39 sec	0.06 sec
18	5.92 sec	1.24 sec	6.02 sec	0.06 sec
26	9.38 sec	1.48 sec	9.39 sec	0.08 sec
34	12.53 sec	2.86 sec	12.99 sec	0.12 sec
40	∞	∞	13.85 sec	0.13 sec

3.5.4 Execution Workflow

This subchapter describes the runtime workflow of the implemented fault-tolerant framework. Initially, the model training is performed on the Google GCP server and the data is then offloaded from server to RPi and FSU before beginning the execution. In this framework, it is assumed that the ML models used for inference have already been loaded onto the robot, FSU, and incur no extra latency in the execution process. The interaction between laptop and RPi is managed using *Parsl* executors. I have used the *HighThroughputExecutor* and constructed an Ad-Hoc cluster configuration for communication between them. The network connection between RPi and laptop is simulated. Network faults are induced by varying the signal quality of the network connection. Multiple iterations of inference and fault models have been tested and results have been formalised by averaging their values.

Table 3.5: Simulation Parameters.

Variable	***Values/Range***
threshold	(5% - 95%)
link quality	(0% - 100%)
ResNet-50 accuracy	88.64%
MobileNetV2 accuracy	62.65%
image size	(10.18KB - 35.11KB)
preprocessing time	(4.44ms - 59.31ms)
ResNet-50 time	(1.2sec - 4.59sec)
MobileNetV2 time	(0.11sec - 0.35sec)
link transfer rate	1Mbps

3.5.5 Communication and Monitoring Setup

It is difficult to estimate an exact signal strength throughout the communication channels. Therefore, I have estimated the signal strength on controller node and used it as a reference point throughout the inference. In simulation, patterns are generated through a random process for determining the signal quality of each task. For an ideal circumstance, the time it takes to send data over the network is given as ($link\ time = data\ size/link\ speed$). The link transfer speed considered in this work is 1Mbps. Link delay is determined as ($link\ delay = link\ time/quality$). When the quality is set to 1.0, the wireless link is utilised at full capacity and the execution is performed at the fastest possible speed. When quality drops below 1.0, the link delay increases and the task takes more time to execute. This technique has been utilised to estimate variable network latency in the wireless network. A list of all the variables considered in this simulation are also described in Table 3.5.

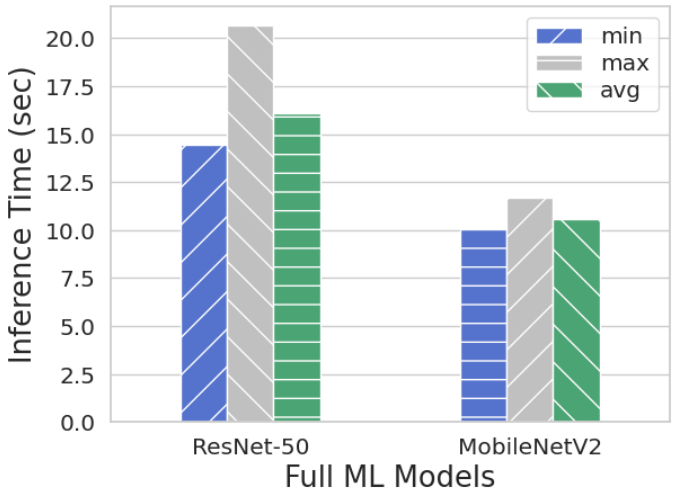

Figure 3.4: Average inference time on full models.

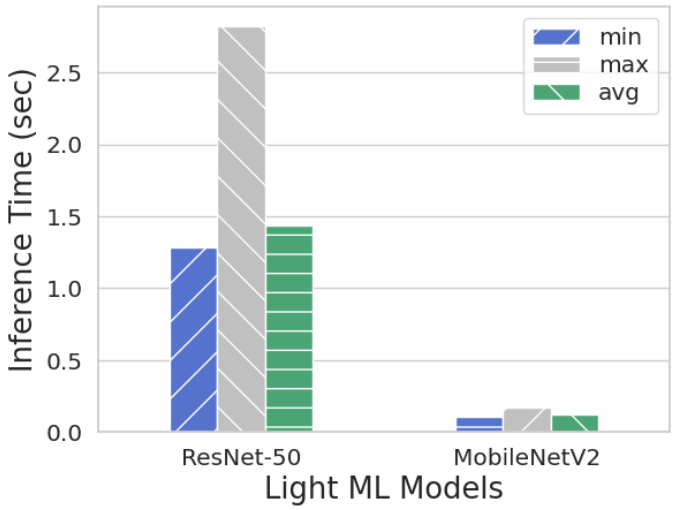

Figure 3.5: Average inference time on lightweight models.

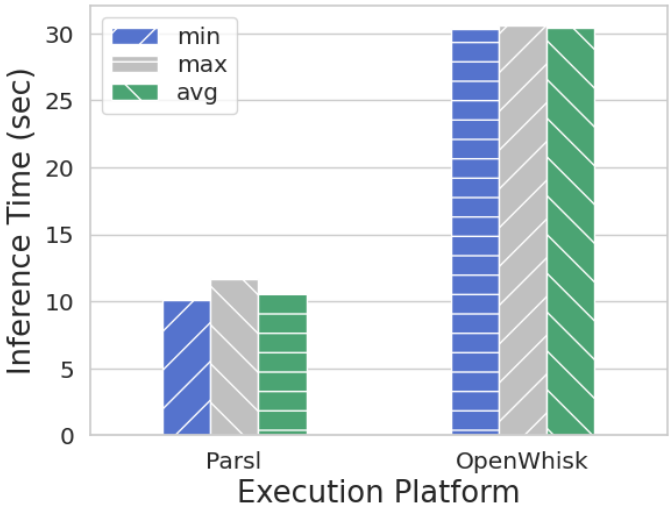

Figure 3.6: Average inference time on 2 platforms.

3.6 Results and Evaluation

This subchapter describes the experimental results and evaluation of our weed inference framework. I have simulated an unreliable connection by adjusting the signal threshold and testing it with the proposed algorithm. The performance of the system is evaluated on the basis of two key parameters: time and accuracy.

3.6.1 Results

Figure 3.4 and 3.5 display the average time taken to perform image inference on full and light models, respectively. I have only utilised lightweight models in this work, but in order to justify not using full models, I have performed inference on full models as well. The results show that inference time on full models are almost 10x in comparison to the light models. Therefore, light models are a better choice over full models for weed/plant inference. TensorFlow Lite models have a smaller file size compared to TensorFlow, and the light model can be directly accessed without the need for additional parsing or unpacking steps, which in turn speeds up the inference process. As a result, this allows a time efficient and effective execution of ML inference tasks on resource constrained devices, having low memory and less computational power in comparison to cloud nodes. Note that the execution time for *MobileNetV2* based model is less than *ResNet-50* model – as *ResNet-50* has 177 layers and about 25.5M parameters, while *MobileNetV2* has 156 layers and only 3.5M parameters.

A comparison of inference time on two different platforms, namely *Parsl* and *Lean OpenWhisk*, is shown in Figure 3.6. I wanted to test our model execution on another comparable serverless platform. Lean OpenWhisk is a lightweight version of the open-source OpenWhisk serverless computing platform that can be deployed on the edge layer and offers all the basic functionalities of the full version of OpenWhisk. I selected the locally executed *MobileNetV2* model as a task for this evaluation. The results show that

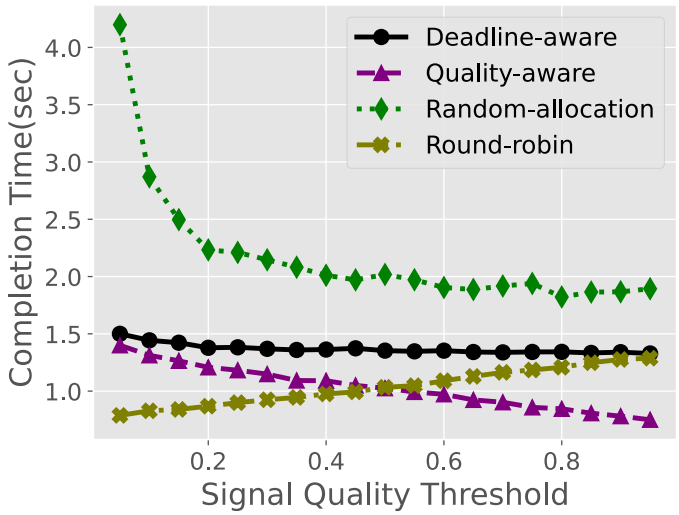

Figure 3.7: Effect on completion time with change in signal quality.

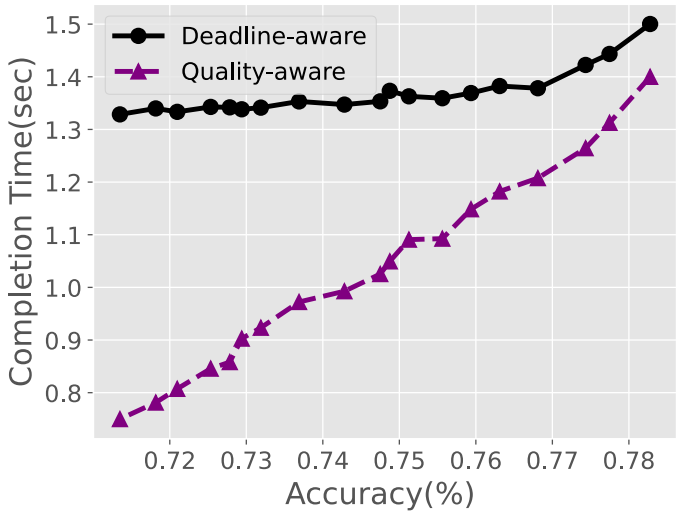

Figure 3.8: Effect on completion time with change in accuracy.

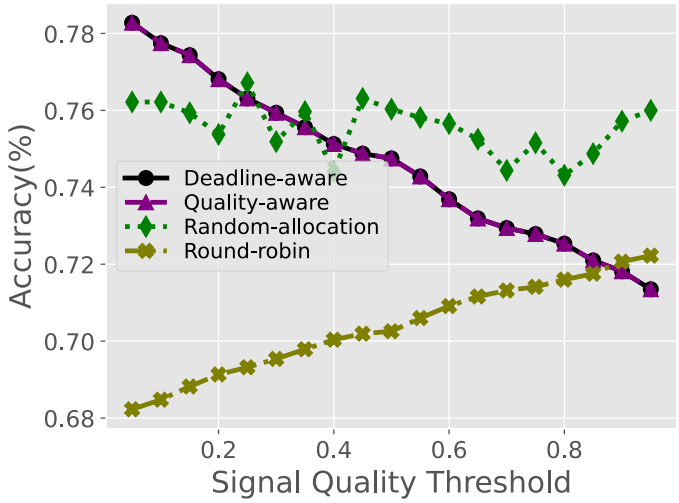

Figure 3.9: Effect on accuracy with change in signal quality.

execution time on *Parsl* is significantly less in comparison to OpenWhisk. This is because Docker instances are created and initialised for performing execution on the OpenWhisk platform. However, *Parsl* functions can be directly run on the node using pre-developed executors. I realised that using these custom executors and dynamic function distribution makes *Parsl* a good platform for performing real-time function execution.

Figure 3.7 illustrates the completion time of deadline-aware and quality-aware methods put forward in the proposed weed inference model. As the threshold for signal quality increases, the completion time shortens, as more inference tasks will utilise the local model for predictions instead of the full model. The completion time for the deadline-aware approach is significantly higher in comparison to signal quality-aware approach. Using the deadline-aware approach, the system has to wait for the deadline to expire before executing the local model for prediction, whereas with the signal based approach, it immediately runs the local predictions if the signal quality is below the specified threshold. Moreover, the completion time for the random allocation approach is high in this experimentation. This is because the algorithm selected full models for most of the evaluations, resulting in high execution time. It can also be verified by high accuracy of random allocation approach observed in Figure 3.9 (because full models are more accurate). For round-robin approach, the completion time increased from 0.78sec to 1.28sec with an increase in signal quality. At low threshold values, most of the jobs are unable to complete execution because of poor signal quality. However, as the signal quality increases, both completion time and accuracy increase – as the approach selects full models for execution alternatively and successfully completes the execution.

The Figure 3.8 illustrates the impact of increasing accuracy on task completion time. It is observed that completion time increases as high accuracy is achieved by the evaluation model. To improve accuracy, I need to perform the inference by utilising the full model (which takes more time to execute). Even at low accuracy, the computation time for the deadline-aware approach is much higher than that offered by the quality-aware approach. This is because the deadline-aware approach uses a local model for inference after the deadline has expired, which adds extra latency to the overall execution time of the proposed framework.

It can be seen in Figure 3.9 that for low signal quality threshold, the accuracy of inference is high. This is because for most of the inferences, current signal quality will be above the threshold and it will be using the full model for inference. However, as the threshold value increases, a higher number of jobs will be running the local model which is less accurate. Hence, the accuracy decreases with an increase in the signal threshold. Along with that, it can also be observed that both approaches give almost the same accuracies with changing threshold values. This is because, in both approaches, the threshold is a measure that mainly affects the decision on where the execution will happen. For signal quality-aware approach, the threshold decides whether to execute the job locally or remotely, whereas in the deadline-aware approach, the threshold is used for estimating the time it will take to execute the job remotely. The estimated time is then compared with job deadline to

evaluate whether the remote execution is a success or not. Therefore, the same model (either local or remote) is picked for evaluation in both the approaches. The accuracy for random allocation is high but the completion time for this approach is also higher because of randomly picking full models for execution. For the round-robin approach, the full models are selected and successfully executed with an increase in signal threshold. This is because a high threshold value ensures a higher chance of successful task execution (with the current signal quality).

3.6.2 Analysis

Based on the experimental results, following observations can be made: (i) The execution time for full models is almost 10 times that of the light models. (ii) *ResNet-50* based models are more accurate in comparison to *MobileNetV2* models, but take longer to execute. (iii) The signal quality-aware approach generates better results in terms of completion time, whereas the deadline-aware approach yields more accurate results (if completed within the deadline). (iv) Changing the threshold value significantly affects the completion time of the signal quality-aware approach, whereas the completion time of the deadline-aware approach is less affected by the threshold value. (v) Inference accuracy of the framework decreases with an increase in signal quality threshold. (vi) A trade-off between accuracy and execution time can be achieved based on user application requirements.

3.7 Summary

In this chapter, an edge-cloud framework that can be used with mobile agricultural robots under intermittent network connectivity is proposed. The approach [114] is aimed at addressing network faults, such as unreliable connections and service outages, that can significantly affect the performance of precision agriculture applications. Using on-board ML models for classification and inference, the robot analyses plants/weeds by taking images through a robot-mounted camera. For demonstration, two ML models were trained for weed identification and prediction using the DeepWeeds image classification dataset with two types of noise. I evaluated our algorithm using experiments performed on a testbed, demonstrating that the approach provides accurate predictions under variable network signal quality. The proposed approach offers better performance in terms of completion time, whereas a more traditional deadline-aware approach is more accurate but takes longer to execute. Although task allocation has been considered in this chapter, the task arrival and its uneven load can significantly affect the performance of edge-cloud IoT systems. The next chapter explores the load distribution aspect in IoT-based applications.

Chapter 4

Load Distribution in Edge Computing Environment

This chapter describes a framework that distributes the task load across various edge computing nodes available in rural environments. The distribution is performed by considering multiple factors that directly affect the performance of execution frameworks. Three different FL-based workflows performing the weed detection task has been considered during evaluation of the framework.

4.1 Distributing Task Load in Rural-AI

The use of technology to automate agriculture processes has revolutionised farming, leading to improved utilisation of resources and decision-making [115, 116]. Agricultural automation has also provided substantial progress towards attaining the UN Sustainable Development Goals (SDGs), especially those pertaining to environmental sustainability, improving agricultural yield, and reducing emissions from farms [117].

While technology offers several benefits in agriculture, rural communities have limited access to data communications and computational infrastructure [118], compared to urban environments. Edge-based infrastructure provides a computing architecture that can be used to enhance agricultural operations and decision-making processes in rural areas. It involves deploying computation nodes near a data source at the edge of the communication network for data processing tasks [119]. This infrastructure enables real-time data processing, dynamic allocation of computational resources, low-latency communication, and storage (including data caching) at the edge layer. Recent work [120, 121] proposes edge computing-based frameworks for precision agriculture. Moreover, the integration of edge infrastructure with Internet of Things (IoT) can be transformative for agriculture. IoT devices can collect critical data (such as soil nutrients, crop and weed details, and weather conditions) and offload that task to nearby edge devices for decision-making [122]. This allows farmers to take immediate action on the fields, directly enhancing agricultural productivity and sustainability.

Unreliable connectivity and environmental variations are factors that affect the performance of rural agriculture. It is therefore important to efficiently utilise these edge-cloud resources in a resource-constrained rural environment [123]. Load balancing [124, 125] techniques can be used to distribute computing tasks and data processing workloads across multiple computational resources, such as edge devices and

cloud servers. Load balancing also supports efficient resource utilisation, enhances system performance, and ensures efficient data processing in agricultural operations.

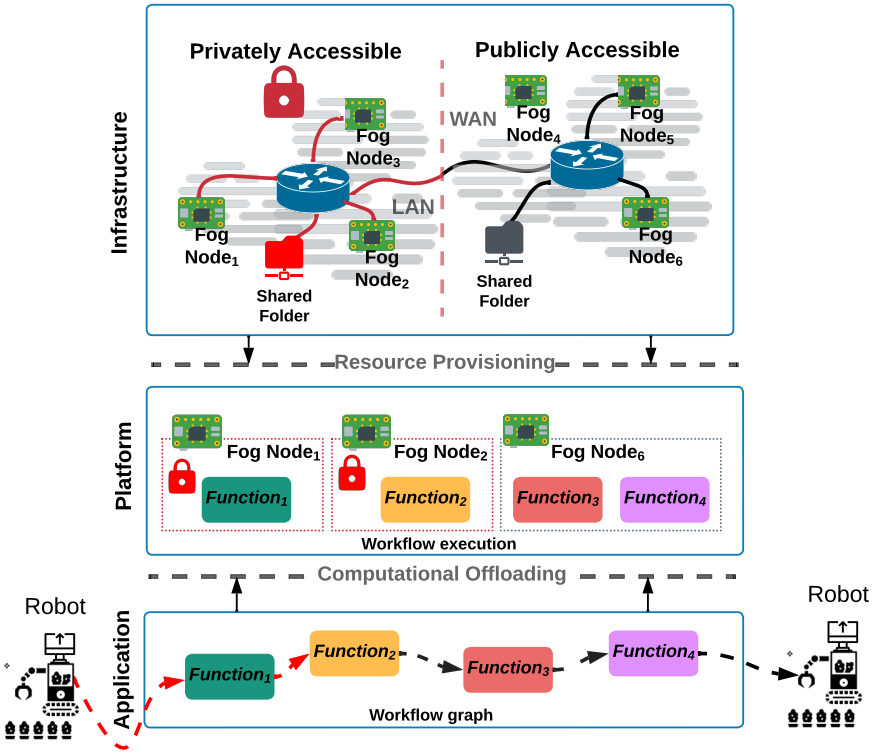

Figure 4.1: Architecture for the proposed load balancing framework. Private nodes (red links) available in high-security zone handle sensitive tasks and shared nodes handle less restricted tasks on the public network.

Farmers have started embracing tools and technologies that are transforming their agricultural practices, reshaping interactions within the agriculture and food sectors. However, farmers are still concerned about privacy, security, and ownership of agricultural data [126, 127, 128]. This can arise due to: (i) farming practices that give economic benefit to individual farmers; (ii) requirement to report greenhouse gas (GHG) emissions; (iii) pre-negotiated energy pricing and tariffs; (iv) use of water and other associated resources (e.g. seed, pesticides, and fertilisers) used on the farm. Moreover, lack of regular software updates, limited technical expertise, poor data management practices and weak access control mechanisms can also contribute towards the vulnerability of rural agricultural data. Considering the limited infrastructure available in rural areas, it is also crucial to optimally utilise all the available resources and ensure that some specified resources are not under-utilised or over-utilised.

I propose a security-aware load balancing framework for edge infrastructure that can be used to support rural environments. The infrastructure is designed to distribute computational tasks across multiple resources while also ensuring data privacy and confidentiality. It divides the tasks into independent categories: restricted and public. These tasks are allocated to two different resources: private and shared, based on security requirements and load characteristics of the node on which these tasks are executed.

A heuristic approach is designed to perform the resource allocation decision for each task, taking into account factors that have a direct influence on execution performance. Completion time, failure rate, resource utilisation, security, and resource management overhead are the key factors used for evaluation. The proposed architecture for the framework is shown in Figure 4.1 of this subchapter. The primary contributions of this chapter are as follows:

- A load balancing strategy for rural infrastructure is designed that also ensure privacy and security of user data. A weed detection use case is used to demonstrate how this approach can be used in practice.
- A mathematical optimisation approach is used to determine the objective functions that affect execution performance, utilising an edge-based load balancing framework.
- Three variants of weed detection functions for evaluation of the proposed framework are analysed: federated learning (FL) based local model training, global model aggregation, and model prediction.
- SHIELD *(Secure Heuristic Integrated Environment for Load Distribution)* is proposed as a framework that allocates tasks on available resources considering waiting time, failure rate, security and other attributes that can be used as a basis of comparison between different allocation strategies.
- Design of a testbed utilising Raspberry Pi and a laptop-hosted server (which can be deployed at the farm). Two Python-based software systems are used: Parsl and OpenWhisk (OW), for evaluating the performance of the proposed framework.

4.2 System Model

This subchapter outlines the key components considered while designing the proposed load balancing framework. A brief description of edge-based infrastructure, a real-life use-case, evaluation task, and communication module utilised are also explained in the following subchapters.

4.2.1 Three-tier Edge-Cloud Architecture

This work implements a three-tier edge-cloud infrastructure, leveraging the strengths of a layered edge computing and storage system. The foundation of this architecture is an edge layer, consisting of physical hardware and communication devices. These components are crucial for collecting real-time, application-specific data. Although these devices can perform basic data processing and storage, their capabilities in this regard are limited.
The middle layer in our architecture is the fog layer, which plays a crucial role in enhancing the computational and storage capacities of the system. This layer is composed of fog nodes, such as base stations or gateways, situated in proximity to the data-generating edge nodes. The strategic placement of these fog nodes is key to their functionality as

intermediaries between the edge devices and the cloud layer. These nodes support faster processing for applications that require lower levels of computation comparatively, thereby reducing latency compared to cloud-based models.
At the top of the architecture is a cloud layer, which offers substantial computational and storage capabilities. This hierarchical arrangement means that as one ascends from the edge to the cloud, there is an increase in computational power and storage capacity. However, this also results in increased network latency and complexity in execution. Therefore, this architecture allows for a strategic trade-off based on the specific needs of the application. By determining the optimal number of layers to be utilised, the architecture can be designed to balance immediate data processing needs at the edge while more complex computational tasks are offloaded to the cloud node. This adaptability makes the architecture suitable for a wide range of applications, optimising efficiency and performance.

4.2.2 Adaptive Control Pipeline

The designed system adapts with the execution environment through a sequence of steps involving data collection, monitoring, analysis, and planning the modifications. The phases of the adaptive control pipeline are as follows:

- *Collecting data:* It includes monitoring of the deployment environment and its performance. All the decision variables that can affect the control mechanism are selected and their corresponding data is collected.

- *Analysing data:* This phase is used to understand the current state of system and its associated decision variables. This could involve identifying patterns, predicting future states, or selecting optimal nodes.

- *Formulating decision:* Based on the analysis of all the selected variables, a decision is formulated by the system to adjust its operation and execution process.

- *Implementing action:* The system then implements the decisions, adjusting its operation as per threshold and test conditions on selected variables. This involves migrating tasks between nodes, avoiding risky nodes, or initiating recovery procedures for failed tasks.

4.2.3 Task and Resource Classification

In order to limit the access of tasks on different layers of resources, I have divided them into multiple categories in this work. The categorisation of tasks is as follows:

- *public tasks:* These are the set of tasks that have minimal or near zero requirements in terms of security and do not need protection. It includes tasks that are processing agri-data with minimal security concerns, such as general environmental data or non-sensitive operational information. These tasks do not contain any sensitive

information, making them suitable for execution on a less secure, more openly accessible environment.

- *restricted tasks:* This category of tasks might contain some sensitive information, such as high-resolution images that could reveal confidential details about farming operations or farm infrastructure. The critical nature of these tasks necessitates execution in secure locations where data integrity and confidentiality are prioritised. Monitoring these restricted tasks is essential for maintaining the privacy and competitive edge of farming operations, as unauthorised access to this data could lead to exploitation that could directly or indirectly harm the farmers.

For executing these tasks on resource nodes, two primary categories of resources are utilised that ensure their security and accessibility needs:

- *private resources:* These resources are exclusively utilised for processing restricted tasks due to their high level of security and controlled access. They are owned and managed by individual farmers or farming organisations that own and monitor the fields where data is generated. Private resources are deployed to ensure the confidentiality and integrity of sensitive data, making them ideal for tasks that require a higher degree of privacy and protection.

- *shared resources:* These sets of resources are shared between a group of farmers or a local community that owns and manages the arable land in that neighbouring area. They are highly cost-effective and scalable but mainly used for public tasks due to their lower levels of data protection and confidentiality. These resources do not require stringent security measures, allowing for efficient resource utilisation among multiple agricultural farms.

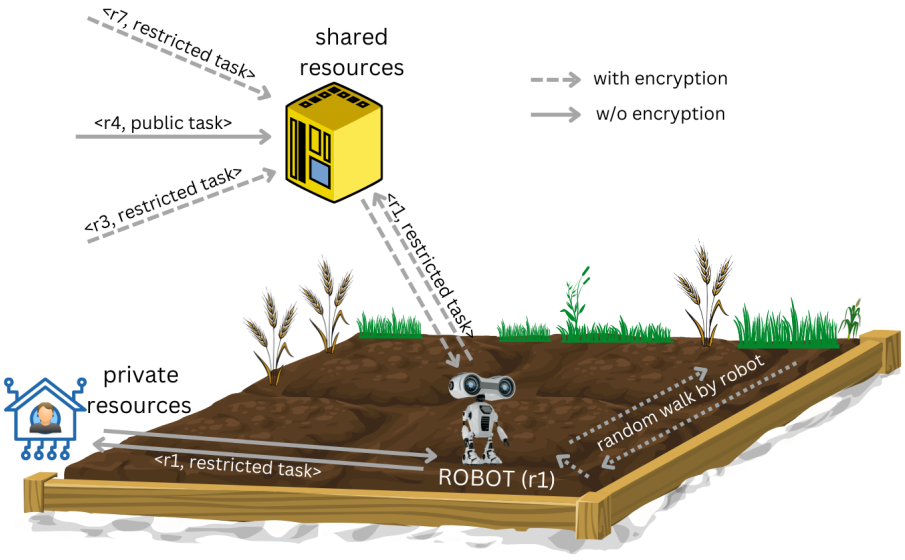

Figure 4.2: Load distribution of tasks in the agricultural field.

4.2.4 Real-World Use Case

For evaluation of our proposed framework, the edge-based architecture has been mapped to a real-life agricultural use case. I assumed that a high-resolution camera is mounted on a robot that takes pictures of weeds and plants on the field. The robot can take a random walk throughout the field and keep taking pictures. The aim of the robot is to detect and remove weeds from agricultural fields. The setup follows a master-worker configuration such that the robot acts as the master whereas the computation nodes act as the workers. There are a few computation nodes that are deployed on the field and owned by farmers; they are the private nodes for that particular farmer. Along with this, the farmer can also evaluate tasks on a shared set of resources owned by a group of farmers having fields in the nearby area. These shared resources are set up together by the local community of farmers. For limiting the access of farmers to the two categories of resources, I have divided tasks into two types as well: restricted and public. If the images captured for analysis contain visual data that could reveal sensitive information about the farming operations or farm infrastructure, they might be valuable to competitors or unauthorised individuals who could exploit it for their own purposes. These images when sent for evaluation (whether it's weed or plant) need security and are considered as a restricted task. However, if the image does not capture any personal information that directly or indirectly affects the farmer, it's considered as a public task. A graphical description of our load distribution approach in an agricultural field is shown in Figure 4.2.

4.2.5 Functional Requirements

Rural agricultural applications require a robust and adaptable system that can respond to real-time environmental variations that affect the infrastructure. The key requirements for designing our application are as follows:

- *Ideal pick for execution:* A system can be designed to utilise all the available resources while executing a task. A two-phase heuristic pipeline is deployed that uses four unique constraints for selecting ideal resource nodes for execution.

- *Mobility of nodes:* The mobility factor can impact application performance and data exchange over a wireless network. Therefore, the robot's location is designed to influence the nature, size, and latency of supported task execution workflow.

- *Addressing workflow necessities:* Different FL workflows exhibit heterogeneity in their computational demands, requiring different times for completing the execution. In scenarios where workflows are interdependent, the outcome of one workflow serves as the input to another process in FL. Three different workflows have been considered for evaluation in our approach.

- *Protecting data rights:* The edge-based framework is designed with consideration of protecting farmers' data security and privacy. I have utilised encryption, hash-based

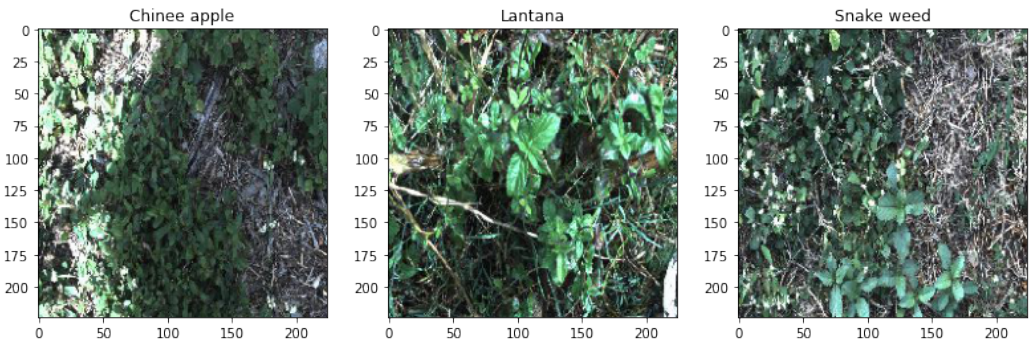

Figure 4.3: Three sample images from the DeepWeeds dataset.

message authentication code (HMAC), and access control mechanisms to ensure confidentiality, availability, and integrity of data.

4.2.6 ML Model Description

In order to perform weed detection on the proposed framework, I have leveraged the *ResNet-50* [68] convolutional neural network model to classify weed species using the DeepWeeds [69] image classification dataset. This dataset includes nearly 17.5K agricultural images (15K training, 2.5K test images), providing a broad and diverse set of observations for eight primary weed species from a wide Australian region, enabling the comprehensive training of our weed detection model. Three sample images from the DeepWeeds dataset are shown in Figure 4.3. The *ResNet-50* model, known for its residual learning framework, was specifically selected for its efficacy in handling complex, high-dimensional data. I then fine-tuned the model parameters to optimise its ability to discern unique patterns within the images, thereby enhancing its weed classification accuracy. The functions that formulate execution workflows utilised in this framework, can be given as:

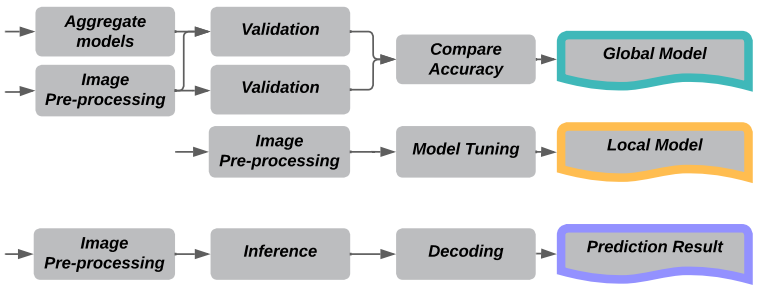

Figure 4.4: SFC for global, local, and prediction workflow.

(1)- Pre-process images: This initial phase involves manipulating an image to make it suitable for training or validation. A function reshapes the data into a 224x224 image size format, which aligns with the requirements of our deep learning model. *(2)- Fine-tune model:* This phase focuses on fine-tuning our learning model. It involves initialising the model with weights from previous iterations, leading to the creation of a new model

specifically trained on our DeepWeeds image dataset. *(3)- Aggregate model:* The step includes averaging the weights across all neural network models, providing a comprehensive and combined model representation. *(4)- Validation:* The newly trained model is tested using a new dataset to evaluate its performance and accuracy. *(5)- Accuracy comparison:* This function uses the validation results as input and compares the accuracy of the previous model with the new one. The model having higher accuracy is then considered as the global model for future evaluations. *(6)- Inference model:* The function utilises the trained model for performing prediction on new and unseen data. It is used to test the real-world applicability of the trained models on the specific applications it is trained for. *(7)- Data decoding:* This function handles the encryption and decryption of data before it is sent to a resource node for execution. An algorithm is designed to decide whether data encryption is required or not.
Figure 4.4 describes the service function chain (SFC) for task variants utilised in this work. I have evaluated the framework on these three weed detection tasks:

- *Local model training:* The task includes training the FL model on local data of each node. It includes pre-processing the end-user data and fine tuning it over a pre-trained model of another dataset. In our case, the weed detection model was fine-tuned over a model pre-trained on the ImageNet [129] dataset.

- *Global model aggregation:* This task collects local models from all the nodes and aggregates them into a single global model. It includes averaging models, pre-processing, performing validation, and comparing accuracies. The model having higher accuracy is selected as a global model for upcoming tasks.

- *Prediction model:* It uses the trained model for performing evaluation on new unseen data. The data belongs to real-world applications which might need security, therefore, the data encryption is also performed before it is sent for execution.

4.2.7 Serverless Computing Platforms

Serverless architecture provides a distributed computing infrastructure where the management tasks such as managing servers, runtime environments, and the underlying operating system are entrusted to a third-party service provider. Instead of pre-allocating resources, serverless architecture dynamically provisions and scales resources for applications by automatically responding to incoming event triggers or requests. By using this approach, users can focus more on designing the functionality and logic of their applications rather than infrastructure, making it a cost-effective and efficient model that charges only for the compute time consumed by the applications.
Numerous serverless computing platforms, serving a wide range of use cases and requirements are available in both commercial and open-source settings for the end-users. AWS Lambda, MS Azure Functions, and Google Cloud Functions are a few platforms that are used commercially whereas Apache OpenWhisk, Kuberless, OpenFaaS, and Knative are a few of the open-source options available for deployment. Commercial platforms are

often available to users with built-in integration capabilities of their respective ecosystems, such as database services, analytics tools, and machine learning platforms but they incur high execution costs, implementation restrictions on the users. Alternately, open-source serverless platforms are freely available for anyone to use, modify, and distribute but they require more hands-on management and operational knowledge, especially for scaling and maintaining the infrastructure. Parsl and OpenWhisk are two serverless computing platforms that have been utilised in this chapter.

Parsl [130] is a Python-based scripting library that allows parallel execution of applications across multiple cores and nodes utilising its pre-developed executors. All the functions available for execution are wrapped using Python apps and are linked to a shared input/output data source such that parallel execution of tasks can be performed on a specific resource node. A configuration object, specifying where and how tasks should be executed, enables running a Parsl application on any machine. It can be deployed on a resource-constrained device like RPi, Nvidia Jetson or it can be assigned to a high-performance computing cluster in the cloud layer. Our work uses Parsl for configuring an ad-hoc cluster that offers a dynamic workflow for real-time task execution on worker nodes deployed for remote execution of tasks. Parsl also allows the dynamic distribution of task load by utilising a controller node. Moreover, function hosting in Parsl supports a heterogeneous environment that can be modified at runtime, especially in the event of a node failure. A registry is maintained for all deployed functions and updated with the arrival or completion of any existing function.

OpenWhisk [131] is a freely available (under Apache Licence 2.0) serverless computing platform designed to automatically run functions in response to specific triggers or events. OpenWhisk operates on a Function-as-a-service (FaaS) model, which offers cloud-based infrastructure and server management for applications. In this platform, functions are labelled as “actions”, and their executions are termed as “activations”. End-users have to register the actions, which can be triggered by an event. An event can be an HTTP request, a timer, or an external resource. Actions can be coded in a variety of programming languages like Python, Java, GO, Swift, or can be integrated as Docker images. OpenWhisk operates on the assumption that there is a linear relationship between a container's memory utilisation and its CPU utilisation. Thus, developers only have the option to determine the RAM size (memory setting) for executing their actions. Developers can also provide an action chain, which permits the sequential calling of one action by another, resulting in complete execution of the entire SFC. The OpenWhisk architecture consists of two main components: (1) the Controller and (2) the Invoker. Both of these components are built over Nginx, CouchDB, and Kafka. Initially, the system is accessed by Nginx, an HTTP server, and a reverse proxy server that allocates incoming HTTP requests to the Controller node. The Controller node is responsible for the authentication and authorisation of all incoming OpenWhisk API requests. CouchDB, an open-source data store securely keeps rules, definitions of triggers, user credentials, metadata, activation, and actions. Kafka is a streaming platform that handles the real-time data exchange

between the Controller and the Invoker. OpenWhisk also has a commercial version of its platform, it is called IBM Cloud Function. I chose Parsl and OpenWhisk as our serverless platforms to demonstrate the compatibility of our approach with different environments that efficiently manage and offload workflows. In subchapters 6 and 8, I have presented detailed results from our experiments with these platforms, validating our approach's applicability in managing distributed serverless computing tasks.

4.2.8 The CIA Triad

For ensuring the protection of sensitive agricultural data, the fundamental concept of CIA triad is utilised in this chapter. It consists of mainly three elements: *Confidentiality*, *Integrity*, and *Availability*.

Confidentiality is the first component of triad which is ensured through the implementation of advanced encryption methods. Encryption helps to safeguard sensitive agricultural data by transforming it into an unreadable format, decipherable only by authorised end-users (or a group of farmers) possessing the correct decryption key. The second principle, Integrity, is ensured through the use of HMAC in our framework. Using HMAC, a hash function is applied to the secret key and contents of the agri-task. Any changes, intentional or accidental, in the data, will result in a different hash value. Therefore, verifying this value before and after transmission can ensure that the data has not been tampered with throughout communication, preserving its integrity. Availability is another crucial factor which ensures that the system's services are accessible to end-users all the time. To achieve this, I allocated the task on two best locations selected via hf2() heuristic function. It provides an alternate solution in case one of the task executions fails. The techniques used in this chapter to ensure CIA are based on well-known methods utilised for data protection [132]. I have used these established security methods to enhance the effectiveness and reliability of our proposed approach. Integrating these security principles can significantly improve the protection of sensitive agricultural data within an edge-based infrastructure.

4.3 Problem Formulation

In this subchapter, I present a quantitative objective function for our proposed architecture which schedules IoT jobs in a load balanced manner. I have interchangeably used the term ‘task’ or ‘job’ in this chapter. Both these terms refer to the execution of a weed detection function. I have a set of J jobs and R resources such that a job can be sent on any available resource for execution. All the symbols of this formulation are described in Table 4.1. Each job can have a unique size and each resource can have a different capacity, which will result in different execution times' on different resources (depending on their capacities). The time taken to complete a task/job j on a resource r is known as the completion time, t_{cmp}. This can be represented as:

Table 4.1: Symbol Table for the problem formulation.

Symbol	***Description***
N, N'	total no. of tasks, no. of tasks failed
t_r, t_p	restricted, public security tag
J, R	set of jobs, set of resources
j, r	a task/job, a resource
t_{st}, t_{exec}, t_{cmp}	start, execution, completion time
dl_{trans}, dl_{prop}, dl_{proc}	transmission, propagation, processing delay
dl_{queu}, dl_{lat}, dl_{comm}	queuing, network, communication delay
t_{add}	additional time
len, bw	data packet length, network bandwidth
dis, vl	distance, velocity
njq, awt	no. of tasks in queue, average waiting time per job
j_r, j_p	restricted, public task
j_r', j_p'	restricted, public task failed execution
c_{set}, c_{exec}, c_{comm}	setup, execution, communication costs
FR, SC, RU	failure rate, system cost, resource utilisation
τ, cap	time unit, total execution capacity

$$t_{\text{cmp}} = t_{\text{st}} + t_{\text{exec}} + dl_{\text{lat}} \tag{4.1}$$

Here t_{st} is the time when j starts executing on r, t_{exec} is the time it takes to execute j on r, and dl_{lat} is the network latency of executing j on r. Here, network latency dl_{lat} is the combination of various delays that arise in the network when j is executed on r. This is represented as:

$$dl_{\text{lat}} = dl_{\text{trans}} + dl_{\text{prop}} + dl_{\text{proc}} + dl_{\text{queu}} \tag{4.2}$$

where dl_{trans} is the time taken to transmit data from user to the transmission medium, dl_{prop} is the time taken for propagation of data through the transmission medium, dl_{proc} is the time taken by processor to process the users' data, and dl_{queu} is the delay incurred by waiting of data packets in a queue. Since high performance computational hardware is easily available nowadays, I have considered that $dl_{proc} \approx 0$ in this chapter. Thus, the network latency can be written as:

$$dl_{\text{lat}} = dl_{\text{trans}} + dl_{\text{prop}} + dl_{\text{queu}} \tag{4.3}$$

$$dl_{\text{lat}} = \frac{len}{bw} + \frac{dis}{vl} + (njq \times awt) \tag{4.4}$$

such that *len* is the length of data packet, *bw* is the bandwidth of adopted IoT network, *dis* is the distance that a data packet travels, *vl* gives the velocity of communication channel, *njq* is the total number of jobs in the queue, and *awt* gives the average waiting time for one job in the queue. In this model, dl_{trans}, dl_{prop} and dl_{queu} together depict the communication delay in the network. This combination of delays is often used interchangeably with dl_{comm} in this chapter.

It is necessary that the task allocated on the node begins the execution on time. However, due to system delay or improper garbage collection, the system faces some delay in beginning the task execution. This results in task failure and may impact overall completion time. To handle this issue, I have also considered additional time on resource nodes in this framework. It can be mathematically given as:

$$t_{add} = t_{exec} \times failure\ rate \tag{4.5}$$

such that $failure\ rate$ gives the failure rate of the resource node where task j is sent for execution.

The tasks in the proposed architecture are categorised as restricted and public, as described previously. Let a restricted job be denoted by j_r. Similarly, a public job task is given by j_p respectively. The total number of tasks to be executed is given by N such that:

$$N = \sum j_{\mathrm{r}} + \sum j_{\mathrm{p}} \tag{4.6}$$

Furthermore, a restricted and public job that failed execution on the node is denoted by ${j_{\mathrm{r}}}'$ and ${j_{\mathrm{p}}}'$ respectively. The total number of jobs that failed their execution are given by N' such that:

$$N' = \sum {j_{\mathrm{r}}}' + \sum {j_{\mathrm{p}}}' \tag{4.7}$$

To analyse the performance of task execution, I use three metrics, namely: Failure Rate (FR), System Cost (SC), and Resource Utilisation (RU). FR is the ratio of number of jobs that failed execution to the total number of jobs available for execution. Formally, it can be given as:

$$FR = \frac{\sum {j_{\mathrm{r}}}' + \sum {j_{\mathrm{p}}}'}{\sum j_{\mathrm{r}} + \sum j_{\mathrm{p}}} = \frac{N'}{N} \tag{4.8}$$

The objective function that needs to be solved for FR can be given as:

$$\underset{N'}{\operatorname{argmin}}\, FR = \{FR | FR = f(N')\} \tag{4.9}$$

Another crucial parameter to measure the performance of an architecture is the software and hardware interactions within the network. This factor is given by the SC. It includes the set-up cost, total execution cost, and cost of communication latency. SC can be presented mathematically as:

$$SC = c_{\mathrm{set}} + c_{\mathrm{exec}} + c_{\mathrm{comm}} \tag{4.10}$$

Here c_{set} denotes the initial set-up cost, c_{exec} denotes the aggregated execution cost of all the jobs executed on resource r. Finally, c_{comm} denotes the total cost of communication latency in the network. Since SC is directly proportional to the running time of a resource, it needs to be minimised.

If τ is the standard unit of time considered in this work. then the objective function for

SC can be given as:

$$\underset{\tau}{\operatorname{argmin}}\, SC = \{SC | SC = f(\tau)\} \tag{4.11}$$

Moreover, it is also necessary to keep track of the RU of all available resource nodes. RU is the ratio of total execution time with the total capacity available in our network. In order to achieve an optimal utilisation of a resource, RU needs to be maximized. RU is represented mathematically as:

$$RU = \frac{\sum t_{\text{exec}}}{cap} \tag{4.12}$$

Here *cap* gives the total execution capacity of the available resources. The optimisation function for calculation of RU can be given as:

$$\underset{t_{\text{exec}}}{\operatorname{argmax}}\, RU = \{RU | RU = f(t_{\text{exec}})\} \tag{4.13}$$

Therefore, the optimisation problem for maximising the performance parameter PR can be presented as: Minimise FR, Minimise SC, and Maximise RU. By combining equation $9, 11, 13$; PR can be written mathematically in the following form:

$$PR = max\left(\underset{N'}{\operatorname{argmin}}\, FR, \underset{\tau}{\operatorname{argmin}}\, SC, \underset{t_{\text{exec}}}{\operatorname{argmax}}\, RU \right) \tag{4.14}$$

Note that by maximising resource utilisation, the aim is to optimally utilise resources for task execution. Similarly, by minimising the failure rate and total cost, I aim to avoid failure of tasks and also remove the expenditures which do not provide added value to our task execution framework. Therefore, there is need for a load distribution framework that allocates the tasks on different resource nodes while ensuring that the resource utilisation is maximised whereas the failure rate and total execution cost is minimised.

4.4 The SHIELD Framework

This subchapter describes the details of our load balancing framework proposed in this study. It is assumed that there is a uniform mix of restricted and public tasks available for execution. The term resource is used to signify a node where the execution of tasks can take place. In order to monitor the utilisation of resources, a few new terms have been defined in this work.

Definition-1: *b-score* - is a balance score that estimates whether a particular resource node is over-utilised or under-utilised in the framework. It is measured by calculating the difference between current utilisation (utz) of the node and average utilisation (utz_{avg}) of the infrastructure. The *b-score* is calculated every time a new task is submitted for execution. For every task t_i evaluated on resource r, b-score can be mathematically given as:

$$b - score(i, r) = utz_{\text{r}} - utz_{\text{avg}} \tag{4.15}$$

Definition-2: *dynamic k-value* - In order to reduce the execution complexity, instead of

iterating over all the nodes, a more efficient approach involves selecting k best nodes and evaluating them for further performance enhancement. However, I observed that different functions performed better with different sets of k-values. For determining the dynamic k-value, I have utilised a softplus activation function in this work. Given the following:
(1)- function $Softplus(x) = \ln(1 + e^x)$
(2)- task execution time (t_{exec})
(3)- total resources (R)
(4)- maximum task execution time (t_{emax})
(5)- load constant (c_l)

The system is designed to calculate a value k, which is related to the resource allocation based on task execution time of our system. The softplus function serves as an approximation to the rectifier function, compressing extreme values into a smoother range. The calculation starts by determining δ_T, as:

$$\delta_t = t_{exec} - t_{emax} \tag{4.16}$$

The δ_t undergoes the softplus transformation and is normalised by:

$$\frac{\text{softplus}(\delta_t)}{softplus(t_{emax})} \tag{4.17}$$

Lastly, the calculation is scaled with R and added to 10% of the total resources already available. The load constant c_l is used to adjust this outcome. The final k-value can be represented as:

$$k = \left\lfloor R \times \left(0.1 + c_l \times \frac{\text{softplus}(\delta_t)}{\text{softplus}(t_{emax})}\right)\right\rfloor \tag{4.18}$$

The value for load-constant in this work has been fixed at 0.9 (derived experimentally). Therefore, for the initial 50s of function execution, the allocation will remain constant at 10%. Afterwards, it will be scaled accordingly using equation 4.18. Figure 4.5 shows the relationship between the dynamic k-value and the average execution time of the function.

4.4.1 Heuristic Function Pipeline

With an aim to optimally allocate public/restricted tasks on available secure/shared resources, this work uses a two-step heuristic function pipeline for decision-making on where to execute the task. They are as follows:
hf1(): *filtering the load* - After determining the waiting time, failure rate, and b-score for every resource node, the first heuristic function of the pipeline selects the best k nodes that have the minimum values for all three variables, prioritising them in the order of waiting time, failure rate, and b-score among all nodes available for execution. These k nodes are expected to be the nodes that have been least utilised until the current task execution. All the initial experimentation in this work has been done with $k = 20$. Afterwards, the softplus activation was deployed to determine the optimal k-value that will obtain the best

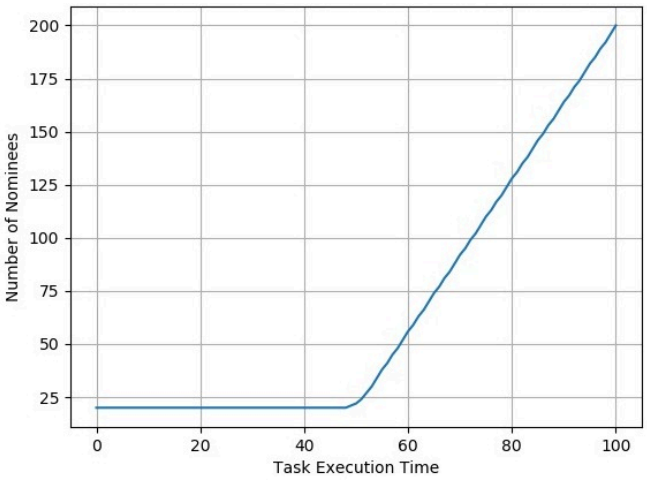

Figure 4.5: Correlation between dynamic k-value and average execution time of the function.

performance results during performance evaluation.

hf2(): *evaluating objective functions* - This function calculates the completion time of a task on all nodes in the selected pool of resources. It includes task execution time and total additional time if the task is executed on a selected node. I have calculated the additional time using the formula: $additional\ time = t_{exec}$ x $failure\ rate$. When a failure occurs during task execution, the node enters the recovery phase increasing the average execution time. Additional time manages the risk by reducing the probability of allocating tasks to a risky node where the failure rate is higher. At last, two best nodes with minimum total execution time are selected and the process is repeated for every incoming task.

4.4.2 Adaptive Cryptographic Measures for Public Networks

The security solution I implemented uses the file system to transfer data between different resource nodes. In instances where this file system is situated on a publicly accessed network, nodes on the public network must be equipped with security measures. Confidentiality is primarily achieved through encryption and decryption mechanisms. On the other hand, to maintain integrity, tokens such as the HMAC value are relayed via a secure channel. This secure channel, exemplified by the SSH connection is utilised by Parsl functions as an adaptive out-of-band channel that adjusts in real-time based on security needs and conditions.

Figure 4.6 depicts the model's tuning procedure. This involves adding a security layer to nodes that are accessible through public networks. Notably, Fog Node_6 is reachable via such a network. As a result, a cryptographic approach is required to safeguard its integrity and privacy. In this setup, the robot handles encryption and HMAC generation, while Fog Node_6 manages decryption and verification.

To further enhance the security and robustness of our system during interactions with publicly accessible networks, I introduced a series of functions that implement cryptographic encryption:

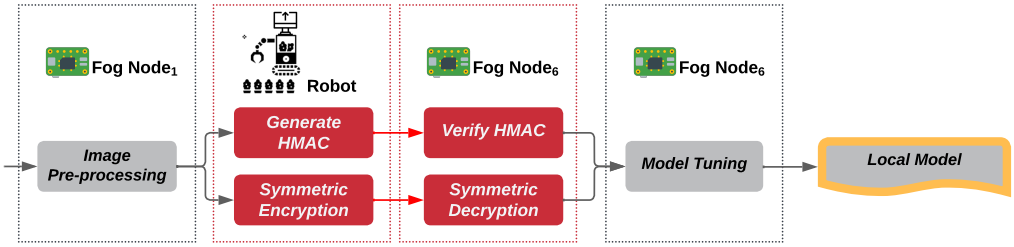

Figure 4.6: Model tuning for public network access. Robot encrypts and generates HMAC and Fog Node$_6$ manages decryption and verification.

Generate HMAC: This function creates an HMAC designed for a particular message and key using a specified SHA algorithm. This HMAC is fundamental in confirming the integrity and authenticity of a message, defending against unauthorised changes or interference.

Verify HMAC: This function validates the authenticity of a received HMAC by comparing it with a newly created HMAC for a designated message. This creation process makes use of a cryptographic key and the SHA algorithm. A successful match results in a 'true' outcome, validating the message's integrity. On the other hand, a mismatch leads to a 'false' outcome, hinting towards a potential security lapse or interference.

Symmetric Encryption: Through this function, a message M undergoes symmetric encryption, using the Advanced Encryption Standard (AES) in Cipher Block Chaining (CBC) mode. An auxiliary cryptographic key assists this process, and a random initialisation vector (IV) further enhances the encryption. The output includes the IV and the consequent ciphertext C, preparing it for a potential scenario of decryption.

Symmetric Decryption: This function serves as the complement of an encryption process, decrypting a given ciphertext C using AES in CBC mode. Initially, it extracts the IV and ciphertext from C. Then the decryption process continues by leveraging the key and IV. Any padding is subsequently stripped away, generating the original plaintext message M. When integrating cryptographic functions into the workflows, there is an added overhead to the execution times of various tasks. For instance, the *pre-processing* function experiences an overhead of about 0.72s, and the *model tuning* function results in an increase of roughly 1.74s. The *compare accuracy* function has the most significant increase, with an additional time of 3.61s. In contrast, the *decoding* function has a minimal overhead of just 0.03s. Note that our SHIELD mechanism incorporates these overheads by considering them as a part of the node's execution duration. This approach provides a comprehensive view of the time factor while implementing cryptographic security measures on public networks.

4.4.3 Load Balancing Algorithm

The proposed algorithm is designed to identify two best nodes for task execution and allocate the tasks on those nodes. A set of tasks, a set of resource nodes, and security tags associated with each task are considered as inputs for the algorithm. The procedure starts by selecting tasks one at a time, in the order of their arrival. For each selected task t_i, the

algorithm evaluates its waiting time on all the available nodes and selects the node that has the lowest waiting time. If the waiting time of multiple nodes is similar, I evaluate the failure rate of those nodes and select the nodes that have minimum failure rate. If the nodes have similar failure rates as well, I evaluate the b-score of those nodes and select nodes with minimum b-score among all available nodes. A softplus activation function is deployed to identify the optimal k-value for executing task t_i in this framework. The first heuristic function (hf1) is then invoked to select the k-best resource nodes based on the previously mentioned three factors in the same sequence. The second heuristic function is then utilised to select two best nodes (r_1 and r_2) where the task will be finally allocated. The decision was taken by considering the execution time and additional time on selected k-nodes. The algorithm then continues examining the task t_i and resources r_1 and r_2. If the task is private, the algorithm first checks whether the selected node is secure or not, and then executes the task. The task is encrypted before execution if the node is a shared node. On the other hand, if the arrived task is public, it is not encrypted on either private or shared resource node and is directly sent for execution. A pseudocode for the proposed load balancer is given below in Algorithm 4.1.

4.4.4 Access Control Mechanism

A mechanism that governs the availability of data within different layers of our edge computing environment, is utilised to manage access control in this work. The aim is to restrict the accessibility of specific categories of tasks to certain groups of resources that are unsuitable for execution. In this work, the unsuitability is evaluated based on the security requirements of an end-user. If the arriving task is restricted, it would be risky (in terms of security) to allocate this task on a shared resource as the communication channel will be accessible to a lot of other users as well. Therefore, a better option would be to limit allocation of all restricted tasks on private resources only whereas all the public tasks can be allocated to shared resources for execution. For experimentation, I have utilised three versions of access control mechanisms. They are as follows:
(1)- Secure Random Placement: When a task arrives for execution, it is randomly allocated to any of the resource nodes where the security tag matches the task requirements. For example, when a task with a restricted tag arrives, it will be allocated randomly to one of the available private resource nodes. Similarly, a task labelled with a public security tag will be randomly assigned to any available shared resource (and not to any private resource). *(2)- Secure Round Robin:* Instead of randomly selecting a node for task execution, this approach allocates each arriving task to resource nodes in a cyclic order. However, it is crucial that security tag of the task matches the requirements of the resource. All restricted tasks will be scheduled on private resources in a cyclic order. Similarly, all public tasks will be allocated on shared resources in a similar cyclic order. *(3)- Secure Least Loaded:* This approach allocates tasks to the resource node which has been least utilised among the total available pool of resources. However, I make sure that if the task is restricted, it is allocated to the least loaded resource node in our private pool

Algorithm 4.1 SHIELD

Input: set of tasks, set of resource nodes, security tags
Output: two best nodes for task deployment

```
1: procedure LOAD-BALANCER()
2:     select tasks (one-at-a-time) in the arrival order
3:     for selected task t_i do
4:         calculate waiting time
5:         calculate failure rate
6:         calculate b − score
7:         identify k − value
8:         procedure HF1()
9:             select k best resource nodes
10:        end procedure
11:        procedure HF2()
12:            identify two best nodes - r_1 & r_2
13:        end procedure
14:    end for
15:    for task t_i and selected node r_1 & r_2 do
16:        if t_i → private then
17:            if selected node → secure then
18:                perform execution
19:            else if selected node → shared then
20:                encrypt task
21:                perform execution
22:            end if
23:        end if
24:        if t_i → public then
25:            if selected node → secure or shared then
26:                perform execution
27:            end if
28:        end if
29:    end for
30: end procedure
```

of resources. Similarly, if the public task arrives, it will be assigned to the shared resource node which has been least utilised. Whenever a new task arrives, the allocation decision is made by evaluating the utilisation of all the available nodes.

4.5 Performance Comparison: Parsl vs OpenWhisk

Table 4.2 below provides evaluation metrics for Parsl and OpenWhisk platforms, describing their performance when executing an FL application on a Raspberry Pi 4. Performance benchmarks are measured across a range of tasks: pre-processing, model tuning, averaging models, model validation, comparing accuracy, inference, and decoding. A comparative analysis of each task with its corresponding average execution time is conducted, revealing the following findings: *(1)- Pre-processing*: Parsl displayed high efficiency in pre-processing operations, completing the task in approximately 0.33s, while OpenWhisk

Functions	***Parsl Time (seconds)***	***OW Time (seconds)***	***OW Memory (MB)***
Pre-processing	0.33	1.80	128
Model Tuning	178.21	161.54	1024
Averaging Models	22.33	17.63	2048
Model Validation	37.16	47.77	1024
Compare Accuracy	0.10	0.66	128
Inference	5.36	30.85	1024
Decoding	0.01	0.98	128

Table 4.2: Performance readings for Parsl and OpenWhisk functions.

required significantly more time, averaging around 1.79s. *(2)- Model Tuning*: OpenWhisk outperformed Parsl in model tuning, completing the task in approximately 161.54s compared to Parsl's 178.21s. *(3)- Averaging Models*: OpenWhisk also demonstrated better efficiency in this task, averaging around 17.63s, while Parsl took approximately 22.33s. *(4)- Model Validation*: Parsl performed better in this task, completing it in approximately 37.16s compared to OpenWhisk's 47.77s. *(5)- Comparing Accuracy*: Parsl demonstrated significant efficiency in this task, requiring only about 0.10s, a small difference to OpenWhisk's 0.66s. *(6)- Inference*: Parsl was far more efficient in this task, averaging around 5.36s as compared to OpenWhisk's 30.85s. *(7)- Decoding*: Parsl completed the task very quickly in 0.0067s, compared to OpenWhisk's substantially longer duration of 0.98s.

Running an FL application on Raspberry Pi highlights the strategic difference between Parsl and OpenWhisk in resource management and their consequential impact on performance. Moreover, a task-specific difference between performance on two platforms is observed. For tasks such as pre-processing, model validation, comparing accuracy, inference, and decoding, Parsl demonstrated higher efficiency. However, OpenWhisk performed better in model tuning and model averaging tasks. It can be clearly seen that Parsl exhibits superior performance in the majority of tasks because it operates more like a native script execution, where the operating system manages and distributes resources among processes. If the system runs out of memory, the OS can start paging whenever the function is called, allowing tasks to be completed, though with some added time. This accounts for the longer duration observed in the model tuning function executed using Parsl. On the other hand, OpenWhisk uses containerisation, allocating a specific amount of memory to each function. If this allocated memory runs out, the function might not run or more likely will fail, triggering a need for task reallocation. Also, each function execution in OpenWhisk requires the instantiation of a new container (cold activation). This process contributes to additional time overhead while executing the functions. Although this overhead appears to be minimal for longer tasks, it can significantly impact shorter tasks. For example: in the model tuning function, the 6s cold

activation contributes just 3.84% to the total 155.69s execution time. However, for the decoding task, the 0.95s cold activation accounts for roughly 99.17% of the total execution time, considerably reducing OpenWhisk's performance for tasks with smaller execution cycles. Therefore, efficient resource allocation becomes even more crucial in a multi-user infrastructure. Defining precise memory requirements for each function, such as the 1GB (or 25% of Raspberry Pi's memory) needed for model tuning, can be challenging. In such cases, a choice can be made to prioritise faster execution over efficient memory utilisation. Analysing these behaviours, it is possible to optimise performance by dynamically selecting either Parsl or OpenWhisk for execution, based on the task and available resources. Parsl could be preferred for tasks such as model tuning when sufficient memory is available to avoid OS-controlled paging. On the other hand, when memory is insufficient, OpenWhisk offers a solution by acquiring all the memory needed for a task beforehand, thus preventing the act of paging. In all scenarios where available memory is limited, it would be better to utilise the OpenWhisk framework (because of better control over memory). Such strategic selection could improve the overall performance and efficiency of FL applications. This selection process can be represented as an optimisation problem in itself and has not been considered in the current approach. Exploring this later could provide a promising direction for future work.

4.6 Experimentation Setup and Design

This subchapter presents a detailed description of the experimental setup devised for the implementation of our proposed security-aware load balancing framework. It includes setting up Parsl, OpenWhisk platforms, and designing a Python-based simulation that can simulate real-world conditions.

4.6.1 Testbed Setup for Parsl

The experimental configuration for evaluating the proposed framework utilises an edge computing environment that consists of an edge node and a controller node connected with each other. The edge node is a Raspberry Pi 4, Quad-core Cortex-A72 Processor, 64-bit SoC, 4-GB RAM, and Nvidia Jetson Nano, Quad-core ARM Cortex-A57 MPCore Processor, 4-GB RAM, located at Indian Institute of Technology (IIT) Ropar. The controller node is a Dell Latitude 5420 laptop equipped with an 11th Gen Intel Core i7-1185G7 processor operating at 3.00GHz with 8 cores, 16GB RAM, and 512GB storage. This node is also located in IIT Ropar and the connection between controller and edge node is established using the Parsl framework. Both the laptop and edge node are equipped with a 64-bit Ubuntu 22.04.1 LTS operating system for experimentation. Virtual environments were set up on both machines, and all prerequisites were installed before initiating the task execution process. To facilitate the communication, a Parsl executor (HighThroughputExecutor) and an Ad-Hoc cluster configuration are utilised in this work. A graphical description of the Parsl setup that has been utilised for orchestrating the

workflows is provided in Figure 4.7.

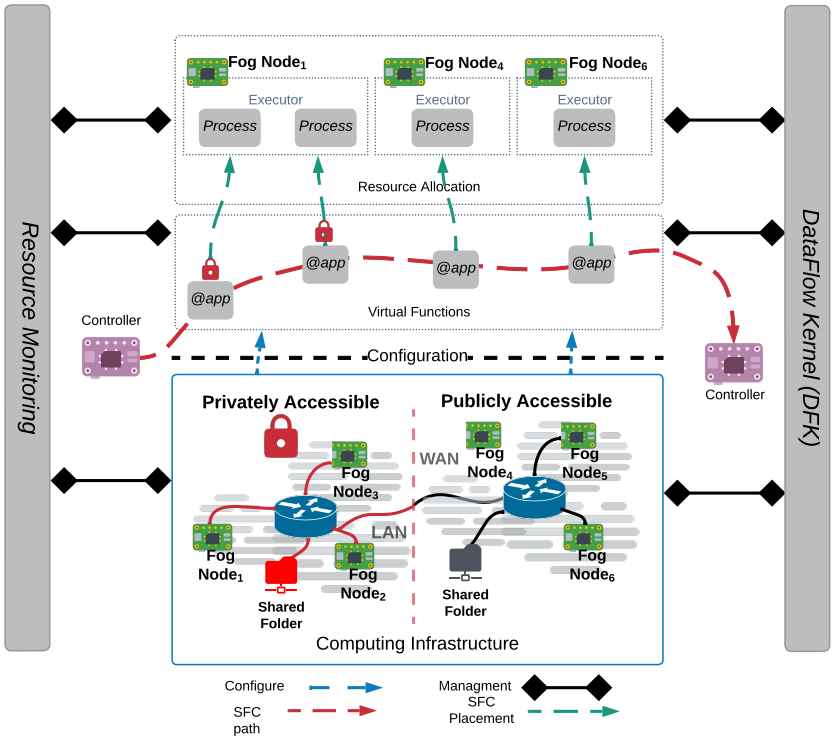

Figure 4.7: Utilising Parsl for pipelining and orchestrating the execution of workflow.

4.6.2 Testbed Setup for OpenWhisk

In order to test our framework on OpenWhisk platform, I have utilised six Raspberry Pi 4 Model B computers, operating on Raspberry Pi OS Lite (32-bit) Debian Bullseye. Every Raspberry Pi is powered by a 1.5GHz 64-bit quad-core CPU (ARM-V8 processor) with 4GB of RAM. A streamlined version of Apache OpenWhisk (Lean OpenWhisk) is integrated with edge devices, which eliminates the need for separate Kafka and Invokers. Currently, the intrinsic compatibility of Lean OpenWhisk is confined to x64 and x86 architectures only (and not ARM). To bridge this gap, I have customised Docker images, facilitating the installation of Lean OpenWhisk on Raspberry Pi devices equipped with ARM architectural setups. For a heterogeneous setup, an Nvidia Jetson Nano, Quad-core ARM Cortex-A57 MPCore Processor, with 4-GB RAM, is also utilised for experimentation with RPis.

4.6.3 Simulation Setup

I have utilised a two-step simulation process incorporating a queuing model and a failure model in this chapter.

Queuing Model: This model is designed to determine the waiting times for all tasks in the queue. It uses a function that considers the arrival time and the current state of

the queue to perform this calculation. Another key function identifies the status of a task (within the queue) and computes its completion time by considering both waiting and execution times. I utilise a dynamic simulation model for emulating a system with single server per node, operating under a First-Come-First-Serve policy.

In Algorithm 4.2, the process for handling task arrivals involves calculating the waiting time (refer to the procedure in lines 3-9) to determine the total completion time, which is the sum of waiting and execution times. As the task arrives, the system compares its arrival time with the time when the last task finished waiting (both are timestamps). This comparison helps to determine if all tasks in the queue have been processed or if there are still tasks waiting/being executed (refer to line 7). In the event of task allocation, the completion time is calculated using the procedure described in lines 10-18. If there are no tasks in the queue (refer to line 13), the system assigns task's execution time as the total completion time (line 14). However, if the tasks are waiting, system adds the waiting time to the total execution time and determines the final completion time (line 16) of that task.

Algorithm 4.2 Queue Simulator

Input: arrival time, execution time
Output: waiting time, completion time

```
1:  procedure QUEUE-SIMULATOR()
2:      Initialize: wait_end ← 0, wait_time ← 0
3:      procedure CALC_WAIT_TIME(arr_time)
4:          if wait_end < arr_time then
5:              wait_end ← arr_time
6:          end if
7:          wait_time ← max(0, wait_end − arr_time)
8:          return wait_time
9:      end procedure
10:     procedure CALC_COMPL_TIME(arr_time, exec_time)
11:         wait_time ← CALC_WAIT_TIME(arr_time)
12:         wait_end ← max(wait_end, arr_time + exec_time)
13:         if wait_time = 0 then
14:             compl_time ← exec_time
15:         else if wait_time > 0 then
16:             compl_time ← wait_time + exec_time
17:         end if
18:         return compl_time
19:     end procedure
20: end procedure
```

Failure Model: The model incorporates a Mean Time Between Failures (MTBF) clock as shown in Figure 4.8, to manage the task executions. If the task execution time surpasses its MTBF, the task will go through multiple cycles (as depicted in Figure 4.9). This model is specifically designed to simulate a system where MTBF is a significant factor. Upon initialisation, the model is provided with two parameters: the Mean Time To Failure (MTTF) and the Mean Time To Recovery (MTTR). A key function within this model evaluates the status and expected completion time of a task considering the system's

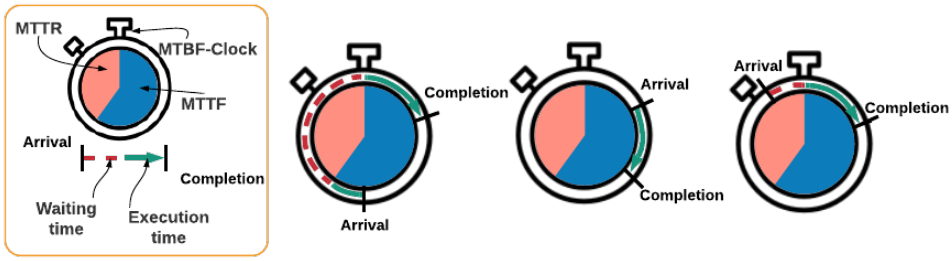

Figure 4.8: The execution of a service function at a process node, with completion times taking account of MTTF and MTTR.

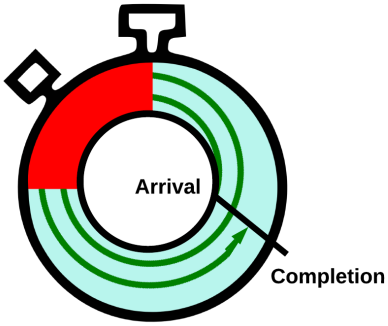

Figure 4.9: When the task execution time exceeds its MTBF, it will cycle repeatedly. At each MTTR interval, the operation halts and then resumes until MTTF is reached.

MTBF. It calculates the expected number of MTBF cycles that a task might undergo and adjusts its completion time accordingly, considering the possibility of a system failure during the task's execution process.

This failure model simulation considers the impact of system reliability and repair time on all the operations. The MTBF clock model provides an even more realistic framework for understanding, planning, and optimising system reliability, maintaining schedule, task execution, and completion timelines. Our custom-built clock operates on an MTBF cycle, which is divided into two phases: the MTTF and the MTTR. If the execution time is less than or equal to the MTTF, the task can be completed within the system's expected operational time. However, if the execution time exceeds beyond MTTF, the task enters the MTTR period, during which progress is paused until the system is restored.

In this model, a unique situation occurs when a task arrives during the MTTR period. In such cases, only the remaining time until system recovery, which is a fraction of the full MTTR period, is added to the task's completion time. When a task is interrupted by the MTTR period or begins in between this period, I have also added an appropriate MTTR time to the total completion time. If the task is interrupted n times by MTTR periods, then the completion time is calculated as: $Completion\ Time = Execution\ Time + n \times MTTR$. For a task initially expected to be completed within one MTBF period but gets interrupted by an MTTR period, the new completion time would be $Completion\ Time =$

Parameter	***Quantity***
MTTF	(250-500s)
MTTR	(20s-100s)
total requests	5,000
local requests	1,500
global requests	500
predictions requests	3,000
fog layers	2
restricted tags	50%
public tags	50%
controllers in field	10
simulation setups	2

Table 4.3: Summary of the simulation parameters.

MTBF + MTTR. If interrupted twice, the completion time extends to *Completion Time = MTBF + 2 × MTTR*, and so on. Please note that – the more frequently a task is interrupted by downtime, the longer it will take that task to complete in a real-world setting.

The two models are linked in a sequential manner. The queue model is initiated first, and its output, indicating the task's waiting and completion times, is used directly as an input for the failure model. The failure model processes this information, determining whether the task can be executed before a system failure occurs and calculates the task's completion time.

4.6.4 Simulation Parameters

Table 4.3 highlights the configuration details of the simulation parameters utilised in this study. I have evaluated the performance of four algorithms: secure random placement, secure round robin, secure least loaded, and SHIELD through a simulation having uniform and normal distribution configuration. The experimentation is carried out on a two-layered fog infrastructure having a restricted and a public layer, utilising RPi benchmarks. Separate tests are conducted on two different platforms: Parsl and OpenWhisk. Reliability metrics MTTF and MTTR range from 250s-500s and 20s-100s respectively in our evaluation. I utilised 200 RPis, distributed across both layers. It also includes three workflows: local model training (1,500 requests), global model aggregation (500 requests), and model predictions (3,000 requests). Each request and RPi is associated with a restricted or public security tag based on its characteristics. In the field, 10 RPis connected to robots act as network controllers. A dynamic k-value is also utilised for selecting the best k nodes in this work.

4.7 Results and Evaluations

This subchapter presents the experimental findings and evaluations conducted on our security-aware load balancing framework. I utilised three key metrics to assess the

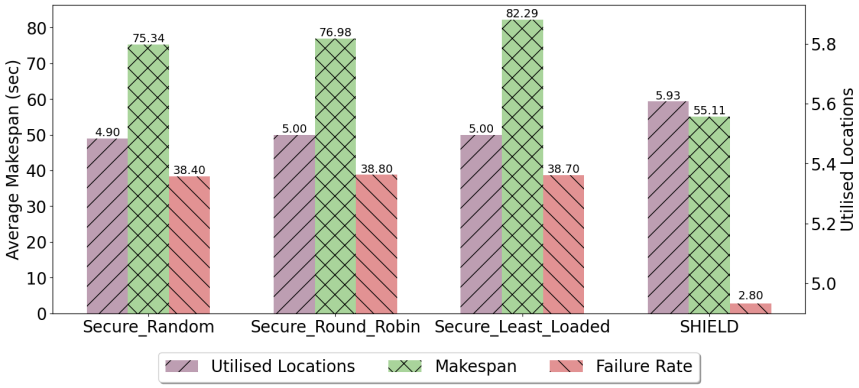

Figure 4.10: Global workflow evaluation on Parsl platform.

performance of proposed system. The first is *failure rate* which measures the percentage of tasks that do not meet their deadlines, showing actual cases of task incompletion. Another important metric is the *makespan*, which tracks the total time it takes for a workflow to execute across different systems, with an emphasis on reducing this duration. I also explored the *utilised location* (or utilised resources) metric, indicating the number of resources involved in a workflow. The goal of this metric is to ensure that resources are evenly distributed. It is worth noticing that basic load balancers such as round robin, might link utilised location to the task size in the workflow. The results compare our proposed SHIELD approach to the random placement method, round robin, and least loaded load balancing strategies. All these methods are secure, adhering to the access control procedures described in subchapter 4.4.4.

4.7.1 Performance Analysis of Workflows on Limited Resource Environment

For understanding the behaviour of framework with less number of resources, I carried out assessment with *k-value* set at 20 (equivalent to 10% of the total resources) rather than relying on the formulation mentioned in Equation 4.18 as depicted in Figure 4.5. Throughout this stage, I closely monitored the results and analysed the framework's effectiveness in load distribution with a smaller subset of resource nodes.

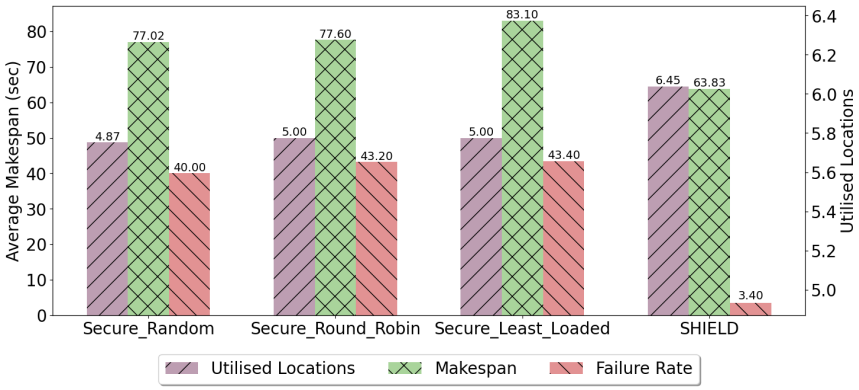

Figure 4.11: Global workflow evaluated on OpenWhisk platform.

Figure 4.10 and 4.11 display the performance of global workflow on Parsl and OpenWhisk execution platforms respectively. The analysis indicates that the SHIELD strategy shows a substantial reduction in makespan (55.11s, 63.83s), reflecting a more efficient execution process. Moreover, the failure rate is significantly reduced to just 2.8% and 3.4%, highlighting the robustness and reliability of our approach compared to other placement approaches. While round robin distributes tasks evenly across all resources in a cyclic manner, the Secure Least Loaded method first evaluates the current load on each node, aiming to allocate tasks to the node having a minimum number of tasks in the queue. This results in some additional time (around 5s) during the execution of workflows. Moreover, the average locations utilised (5.93, 6.45) for SHIELD are higher than the other access control mechanisms. This occurs due to our approach replicating tasks on two locations in a five-function chain workflow. However, instead of utilising all 10 locations, our framework selects locations based on performance factors mentioned previously. Selecting locations for task execution provides better results (in terms of makespan and failure rate) rather than uniformly distributing the load. When I increased the value of k to 100%, I noticed that SHIELD algorithm's performance varies slightly between the two configurations. It utilises more locations (5.93 vs 5.53) and has a marginally lower makespan (55.11s vs 55.97s) with 10% configuration compared to 100% k-value configuration. However, the failure rate is slightly higher when k value is 100% (3.0 vs 2.8).

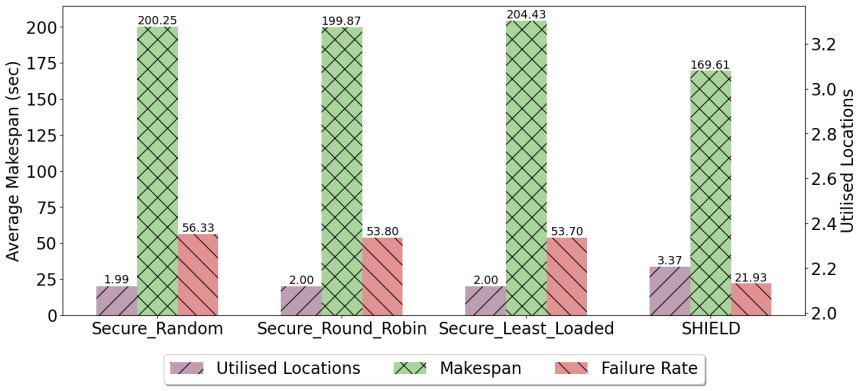

Figure 4.12: Local workflow evaluation on Parsl platform.

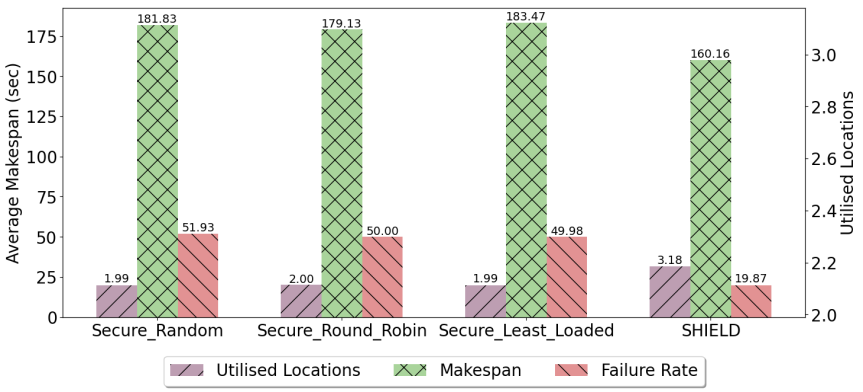

Figure 4.13: Local workflow evaluation on OpenWhisk.

The performance of local workflow on Parsl and OpenWhisk platforms are shown in Figure 4.12, 4.13 respectively. In this workflow, SHIELD also demonstrated better performance over the other strategies. The makespan observed under SHIELD was 169.61s and 160.16s, significantly lower than its counterparts. Furthermore, the failure rate is only 21.93% and 19.87%, far less than Secure Random, Secure Round Robin, and Secure Least Loaded. It can be observed that makespan is significantly higher for all strategies in comparison to the global workflow. This is because the "model tuning" function has a more substantial execution time in the local setting (which is not required in a global workflow). The high execution time not only results in a longer makespan but also increases the probability of failure. Therefore, the failure rate in the local workflow is also higher than that observed in the global workflow. The average utilised locations for SHIELD are 3.37 and 3.18, which is also higher than other three access control mechanisms. The 10% k-value configuration of the SHIELD algorithm utilises slightly more locations (3.37 vs 3.36) and has a slightly higher failure rate (21.93 vs 20.67) compared to the 100% configuration. However, the makespan is marginally higher in the 100% configuration (168.55s vs 169.61s).

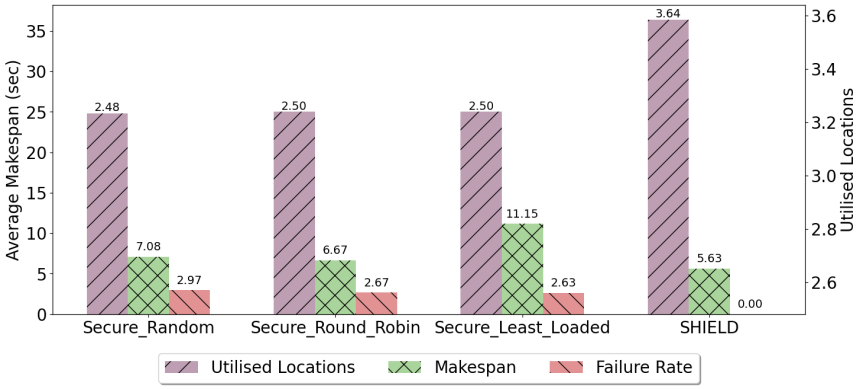

Figure 4.14: Prediction workflow evaluated on Parsl.

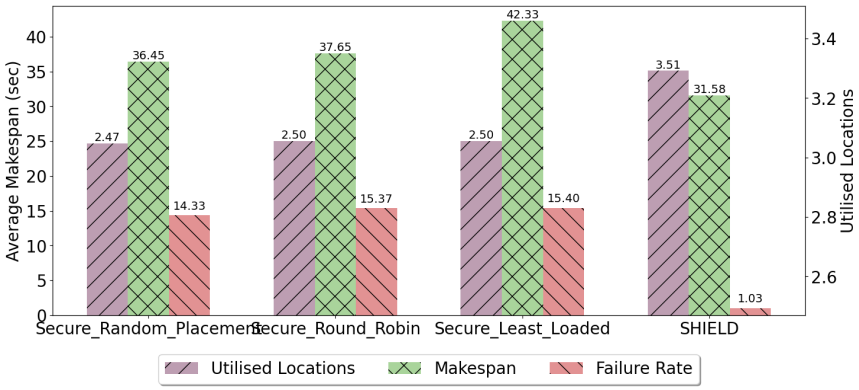

Figure 4.15: Prediction workflow evaluated on OpenWhisk.

For the prediction workflow, the performance on Parsl and OpenWhisk platforms are depicted in Figure 4.14 and 4.15 respectively. The proposed SHIELD strategy demonstrates the highest efficiency with the lowest makespan (5.63s, 31.58s) on both platforms. The access control mechanisms – Secure Random, Secure Round Robin, and

Secure Least Loaded have slightly higher makespan in comparison to our approach. The difference in makespan is attributed to the efficiency of SHIELD in allocating resources, thereby leading to quicker task completion. A slightly better makespan of round robin is observed because of its cyclic allocation of tasks on available resources. It is also observed that the average failure rate for SHIELD is 0.06% and 1.03% respectively. The prediction chain has a very low completion time and it is optimally placed on locations using SHIELD framework. This reduces the probability of execution failure and exhibits a high task execution rate. The performance of the SHIELD algorithm shows a noticeable difference between the two configurations in this workflow. The 10% configuration has a slightly higher number of utilised locations (3.64 vs 3.6) and a lower failure rate (0.0 vs 0.03). However, the makespan is almost identical, with a minor difference of 5.63s and 5.64s in 10% and 100% configuration respectively.

4.7.2 Interpreting the Additional Time Required for Different Execution Workflows

Another critical factor for analysing the performance of our framework is the additional time required whenever a task failure occurs and system goes into a recovery phase. The data presented in Table 4.4 shows the average additional time required by the system whenever a process node faces an interruption, along with the recovery time following such interruptions. This analysis focuses on understanding three available workflows on both available execution platforms.

Table 4.4: Average additional time on Parsl and OpenWhisk.

	Approach	***Global***	***Local***	***Prediction***
Parsl	*Random*	9.93s	10.21s	1.18s
	RR	16.95s	21.13s	4.45s
	SHIELD	0.77s	0.82s	0.45s
OW	*Random*	10.98s	11.57s	1.79s
	RR	18.63s	15.89s	2.6s
	SHIELD	3.39s	4.59s	0.64s

Random Approach: Both Parsl and OW recorded a longer execution time for the global and local workflows. Parsl is approximately 10s and OW is a bit closer to 11s. However, the prediction workflow in both platforms shows a much shorter duration.
Round Robin (RR) Approach: In Parsl, the RR approach showed a significant delay, especially in the local workflow, surpassing 21s. Though global and prediction workflows also showed an increase, it is relatively less than the local version. In contrast, RR method on OW showed a faster recovery for the local workflow in comparison to others.
SHIELD Approach: Our algorithm provides better results on both Parsl and OW platforms. On Parsl, the values range between 0.45s and 0.82s, which provides faster recovery for all three frameworks. SHIELD on OW is faster in comparison to other alternative methods but is marginally slower than Parsl. Across both Parsl and OW, the SHIELD approach succeeds in minimising the additional time. Conversely, the RR

strategy, in the Parsl local and OW global workflow, exhibits higher delays. The prediction workflow consistently recorded shorter additional times, irrespective of the platform. For applications prioritising the minimisation of additional time, SHIELD is a preferable choice. However, it is crucial to consider the intrinsic nature of each workflow, especially the local workflow, which inherently has a longer execution time in comparison to the prediction and global workflows.

4.7.3 Exploring Distribution of Load and its Trade-off with other Performance Critical Factors

To analyse the load distribution capability of our proposed framework, I have performed a comparison with the other three task allocation strategies having uniform load distribution. The uniform distribution strategy ensures a fair distribution of load across network resource nodes, independent of specific characteristics of the infrastructure. It focuses predominantly on maintaining a balance in load distribution, while also considering access control policies.

The random placement strategy employs a uniform distribution function for allocating tasks. Despite generating a balanced distribution, this approach still lacks deterministic predictability. In contrast, the round-robin distribution adopts a more deterministic approach by systematically allocating tasks among all processing nodes available in the infrastructure, thereby assuring fair distribution. The effectiveness of these strategies can be evaluated using the *task allocation ratio*. Both random placement and round-robin strategies typically operate with an allocation ratio of 0.5-0.53%, suggesting each node receives approximately 0.5-0.53% of the total tasks on average. On the other hand, SHIELD prioritises the reduction of waiting times and failure rates over balanced task distribution. The nodes chosen by SHIELD exhibit a higher allocation ratio, typically between 2.47-2.76%. This implies that nodes selected in SHIELD are less likely to miss deadlines, placing reliability over balancing task distribution. Despite this focus on reliability, SHIELD also takes account of task distribution among high-performing nodes. SHIELD increases the allocation ratio of reliable nodes but balances the load between them based on the currently queued tasks, considering waiting time and b-score.

4.7.4 Evaluating the Influence of Dynamic k-value on Overall Execution Time of Workflows

To analyse the behaviour of our framework for optimal resource utilisation and overall execution times, I have used a *softplus* function for dynamically nominating k number of resources that can be deployed for executing the selected task. The formula for determining the k-value (represented in equation 4.18), is essential for the effective distribution of resources. It dynamically adjusts resource allocation in response to the execution time of tasks in our given system. For a scenario where task execution times are equal to or less than 50s, the k-value remains constant. However, if the execution time surpasses this 50s threshold, there is a gradual increment in the number of nodes designated for

future resource allocation. This behaviour indicates a clear positive relationship between the k-value and execution time. However, in practical scenarios, I observed that it is less likely to have more than 20% of the nominees being considered for task allocation.
Our empirical analysis of the dynamic k-value also validates its efficiency. Tasks having longer execution time such as model tuning may experience interruptions from repeated recovery actions. As depicted in Figure 4.12 and 4.13, considering a larger group of nodes within the secondary heuristic function is advantageous in such scenarios.
Interestingly, fewer nominees for global and prediction workflows can lead to improved results (Parsl: 55.11s, OW: 62.71s and Parsl: 5.63s, OW: 31.03s respectively). This outcome is obtained by maintaining a balance between exploration and exploitation in the SHIELD heuristic approach. By applying a dynamic k-value, about 10% of nodes are nominated to the secondary heuristic functions, leading to enhanced outcomes.
Using a dynamic k-value for adjusting the node count across heuristic functions not only improves the optimisation process but also ensures efficient resource allocation. Providing all resources to the next heuristic function for shorter workflows, such as global (Parsl: 55.97s, OW: 63.83s) and prediction (Parsl: 5.64s, OW: 31.58s), can lead to results that are not optimal. This emphasises the significance of k-value in ensuring better system performance, reducing an average of 0.86s on Parsl, 1.12s on OW and 0.01s on Parsl, 0.55s on OW platform (for global and prediction workflows respectively).

4.7.5 Analysing Performance with different Data Distribution and Task Load

All the previous experimentation in the chapter has been performed with uniform data distribution. To evaluate the proposed framework for a different probability of task arrival, I analysed the performance of SHIELD over Gaussian distribution of data, as shown in Figure 4.16, 4.18, 4.20, 4.17, 4.19, and 4.21. I observed that on global workflow, the failure rate increased to 6.20% and 9.40%, for Parsl and OpenWhisk, respectively. Similarly, for local and prediction workflows, the failure rates also increased to 26.27%, 23.93%; 0.07%, 2.47%, respectively, on Parsl and OpenWhisk. I also noticed that the makespan for both distributions was similar, with a slight variation of 1s-3s, on average. Gaussian distribution is known to simulate a more realistic and varied workload scenario where some nodes might be more heavily loaded than others. The experimental results show that SHIELD still outperformed the other three strategies in successfully completing the tasks. However, a slight increase in failure rate can be observed due to the changing load pattern of the distribution, which can be more challenging than managing a consistent, evenly spread-out load.
Moreover, as I increase the task load to 5x and 20x of the initial load, I observe a notable trend in both Parsl and OpenWhisk platforms. In most cases, the makespan marginally increases or remains stable, indicating robust handling of larger workloads by SHIELD. The utilised locations also vary, reflecting adaptive resource allocation strategies to accommodate the increased load. Interestingly, the failure rate increases (around 1-10%,

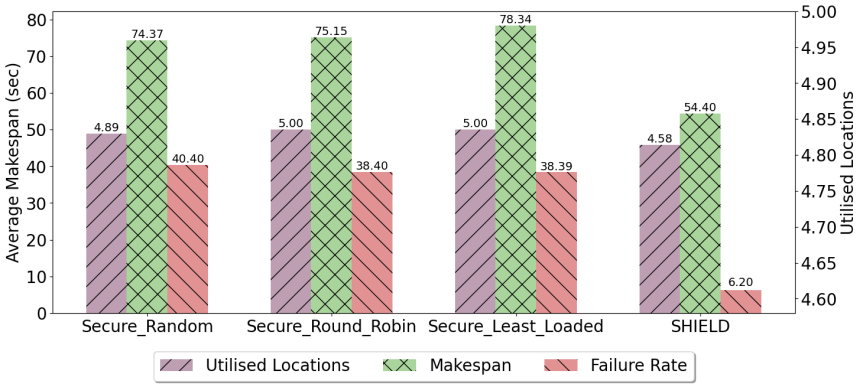

Figure 4.16: Global workflow evaluation on Parsl platform (with Gaussian Distribution).

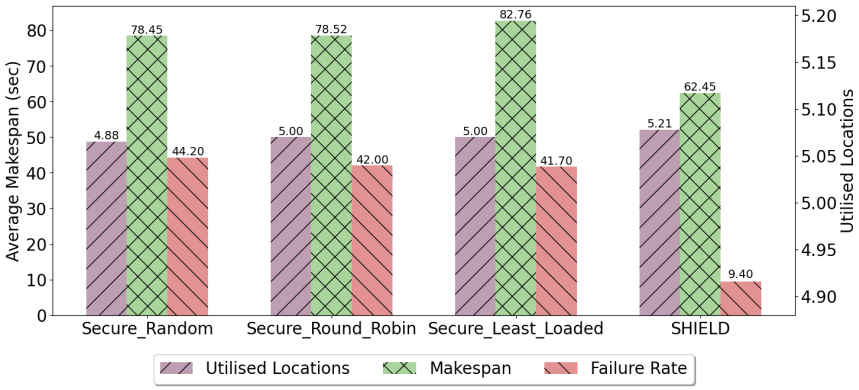

Figure 4.17: Global workflow evaluation on OpenWhisk platform (with Gaussian Distribution).

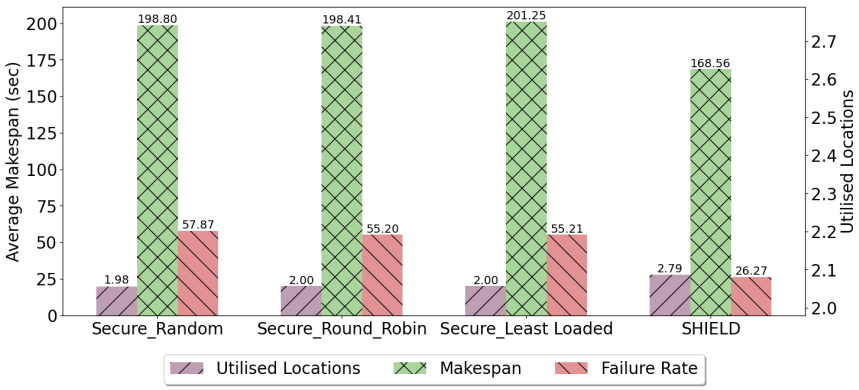

Figure 4.18: Local workflow evaluation on Parsl platform (with Gaussian Distribution).

depending on the type of workflow) with a higher task load, suggesting a correlation between increased workload and the likelihood of task failures. Detailed results about task load can be seen in Table 4.5 shown below.

4.7.6 Analysing Performance in Heterogeneous Resource Environment

In order to analyse the performance in a heterogeneous resource environment, I evaluated the SHIELD framework with two types of hardware resources (RPi and Jetson Nano). I observed that SHIELD outperforms other approaches in utilising locations (4.42, 4.63)

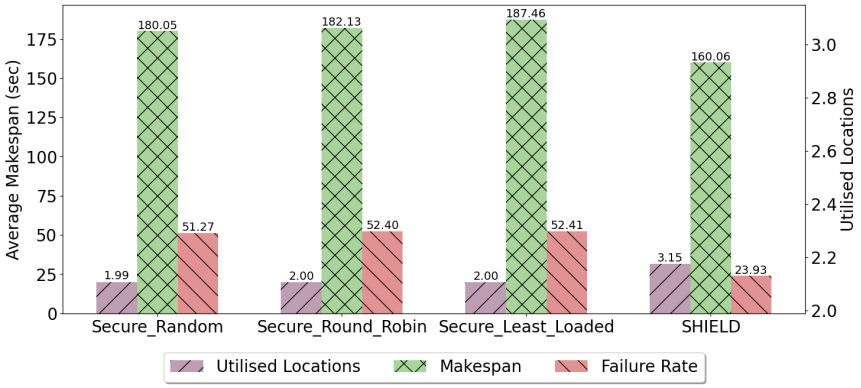

Figure 4.19: Local workflow evaluation on OpenWhisk platform (with Gaussian Distribution).

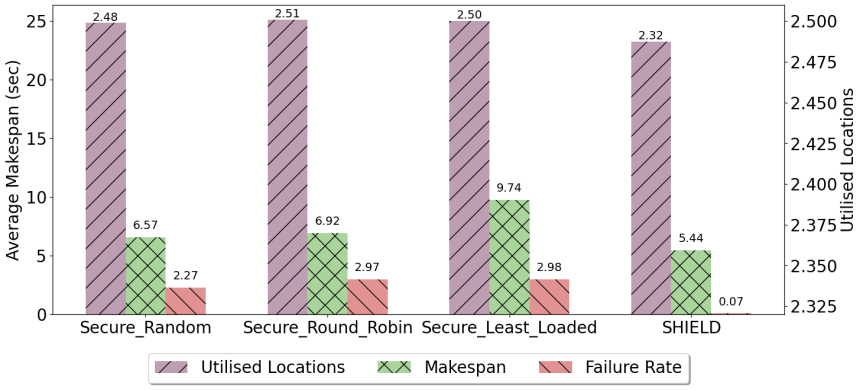

Figure 4.20: Prediction workflow evaluation on Parsl platform (with Gaussian Distribution).

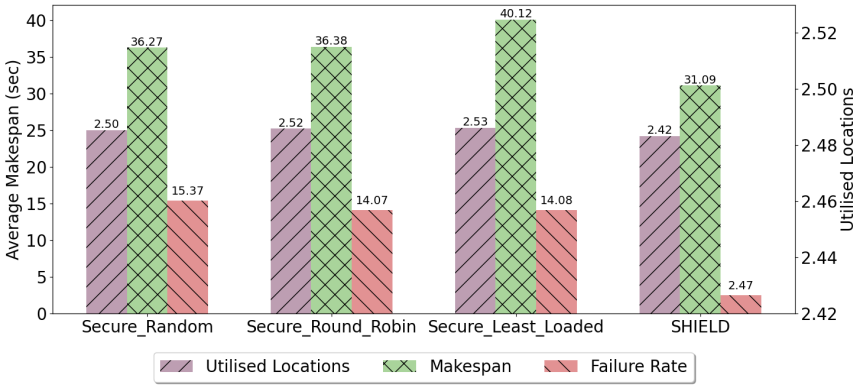

Figure 4.21: Prediction workflow evaluation on OpenWhisk platform (with Gaussian Distribution).

more effectively, achieving lower makespan (81.98s, 85.48s), and ensuring a lower failure rate (9.73%, 8.75%) across both Parsl and OpenWhisk platforms, respectively. Figure 4.22, 4.24, 4.26, 4.23, 4.25, and 4.27 shows detailed results of workflows on Parsl and OpenWhisk with heterogeneous resource nodes. The experimentation demonstrates that Parsl is a better choice for global and prediction workflows, as it offers faster task completion (6.37s, 26.27s), and high reliability (2%, 1.6%). Its architecture and execution model are

Platform	Workflow	Requests Load	Makespan	Failure Rate	Utilised Locations
Parsl	local	5x	169.16	21.52	2.58
Parsl	local	20x	170.39	26.16	2.45
Parsl	global	5x	54.24	6.68	4.91
Parsl	global	20x	54.90	6.87	4.65
Parsl	prediction	5x	5.44	0.01	2.26
Parsl	prediction	20x	5.44	0.07	2.32
OpenWhisk	local	5x	160.11	22.61	2.92
OpenWhisk	local	20x	159.57	23.05	2.84
OpenWhisk	global	5x	63.71	10.88	4.68
OpenWhisk	global	20x	63.95	9.16	4.71
OpenWhisk	prediction	5x	30.98	2.22	2.34
OpenWhisk	prediction	20x	31.02	2.23	2.31

Table 4.5: Evaluating SHIELD framework with 5x and 20x task load.

well-suited for complex computations and scenarios where faster execution and success rates are critical for performance. However, OpenWhisk offers better performance than Parsl in the local workflow, where pre-allocating the resources before task execution can significantly improve the makespan (22.14s), and reduce the failure rate (4.28%).

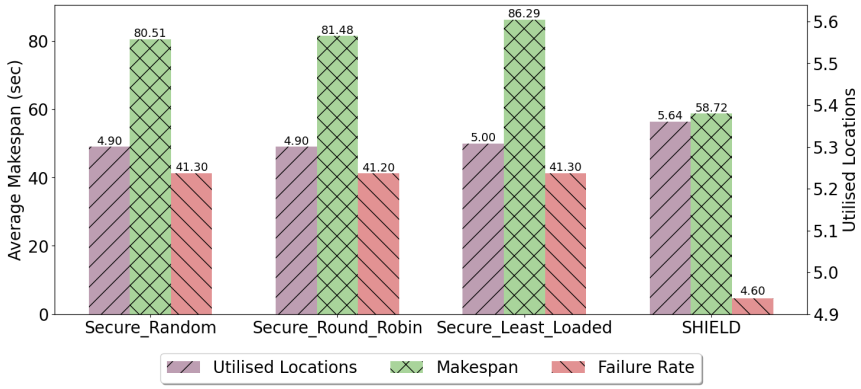

Figure 4.22: Global workflow evaluation on Parsl with heterogeneous nodes.

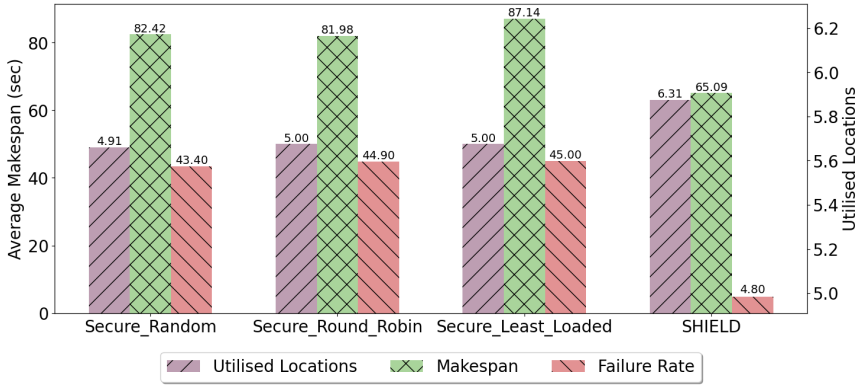

Figure 4.23: Global workflow evaluation on OpenWhisk having heterogeneous nodes.

Comparing the performance of SHIELD, when evaluated on a uniform mix of resources, I observed similar trends across all workflows, for both Parsl and OpenWhisk. The results

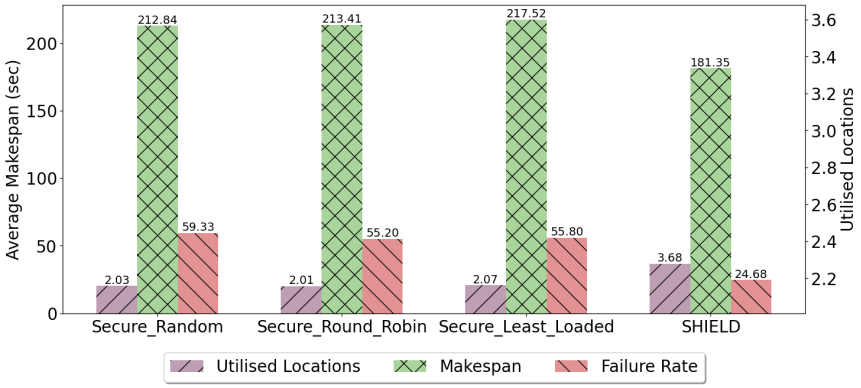

Figure 4.24: Local workflow evaluation on Parsl platform having heterogeneous nodes.

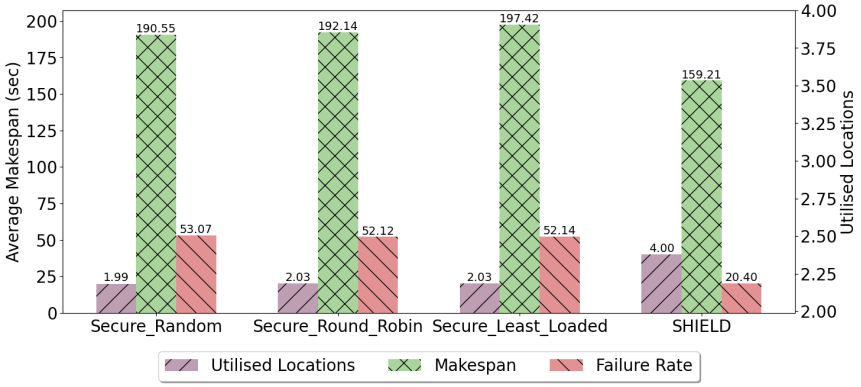

Figure 4.25: Local workflow evaluation on OpenWhisk with heterogeneous nodes.

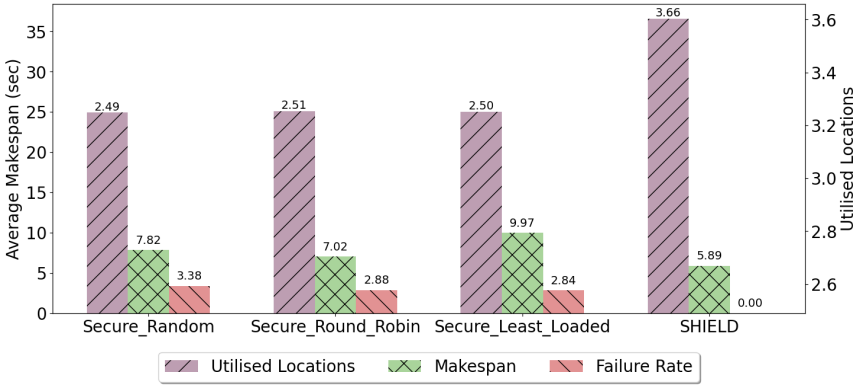

Figure 4.26: Prediction workflow evaluation on Parsl platform with heterogeneous nodes.

show that the resource utilisation in both scenarios is almost similar, whereas there is a slight increase in makespan and failure rate when execution is performed in a heterogeneous environment. This increase can be attributed to the overheads incurred from the serverless architecture and GPU utilisation in the edge layer. In the serverless deployments, as seen with both Parsl and OpenWhisk, there are significant initialisation overheads, more dominantly seen in Jetson Nano, due to its complexity and higher memory requirements. Moreover, Jetson Nano's do not have the advantage of spreading out initialisation costs over longer periods, since functions are temporary, and do not run continuously. This results in notable delays every time the node is reloaded, thus resulting in higher execution

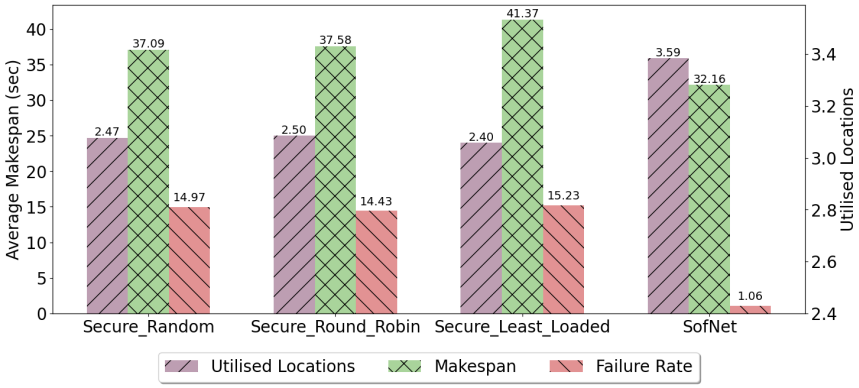

Figure 4.27: Prediction workflow evaluation on OpenWhisk having heterogeneous nodes.

costs during task execution. Furthermore, their performance is also affected by memory hierarchy challenges, as frequent data transfers between different memory layers lead to bottlenecks, particularly in devices that contain limited GPU memory, such as the Jetson Nanos used in edge computing infrastructures.

4.8 Summary

This chapter [133] demonstrates a framework for managing computational tasks within a rural edge infrastructure, establishing a balance between effective load distribution and maintaining data privacy, security of end-users. A heuristic-based two-function algorithm is used to assign tasks on private or shared resources, taking into account completion time, waiting time, failure rate, additional time, overheads, and resource utilisation. SHIELD (Secure Heuristic Integrated Environment for Load Distribution), utilises three task scenarios: local model training, global model aggregation, and prediction model for evaluation of the framework. Our results show that SHIELD not only provides an improvement in completion time (makespan) and failure rate but also reduces the use of risky nodes that have a high chance of failure. The designed framework can also be utilised in other rural applications where load balancing and security are key performance factors. However, another approach to enhance the performance of framework is to optimise the ML and AI operations of IoT applications. The next chapter explores this aspect of improving the ML execution and deployment for an agricultural based use case scenario.

Chapter 5

Optimising AI Operations in IoT-based Applications

5.1 Integrating ML with IoT

In recent years, the advancements in IoT have offered significant opportunities to achieve Sustainable Development Goals (SDGs) by providing enhanced connectivity and real-time data analysis across various sectors. By integrating sensors, actuators, and end-user devices, IoT networks facilitate the collection of vast amounts of data and enable more informed decision-making with optimised utilisation of resources [134]. Integrating ML and AI models in IoT can provide a direction for significant advancement in the domain of computational technology. These models have also seen exponential growth across numerous real-world tasks such as image classification [114], object detection [135], and video analysis [136] in the past few years. In order to achieve higher accuracy and performance, researchers have focused on designing architectures that are both deeper and broader, like VGG, Inception, ResNet, YOLO etc.

Deploying these complex, deep models on resource constrained nodes, such as mobile robots, field side units, unmanned aerial vehicles, and end-user IoT devices, presents significant challenges [137]. Performing convolution operations on these models demands substantial computational power and energy. Additionally, the extensive number of network parameters results in high storage requirements, causing further challenges in resource limited environments [138]. If I consider InceptionV3 and VGG16 models as an example; they have more than 138 million and 23 million parameters respectively. Moreover, to process a single 224x224 image, these models require around 6 billion and 30 billion floating-point operations. In order to implement such large-scale deep learning models on resource constrained infrastructures, it is imperative to tackle the issue of their computational intensity and memory demands.

Numerous methods have been developed to handle the rising demand of memory and resources by deep learning models. Model compression techniques like pruning, quantization, and knowledge distillation are crucial for reducing the size and improving the speed of these models. Pruning [139] involves removing unnecessary weights from a neural network, effectively decreasing its complexity and size. Quantization [139] converts parameter weight values from floating type to integer type, which decreases memory usage and can speed up inference. Knowledge distillation [140] transfers the knowledge

from a large, complex model to a smaller, faster one without significant loss in accuracy. Additionally, creating compact neural architectures and optimising for specific hardware are a few other strategies used to further accelerate the model performance.

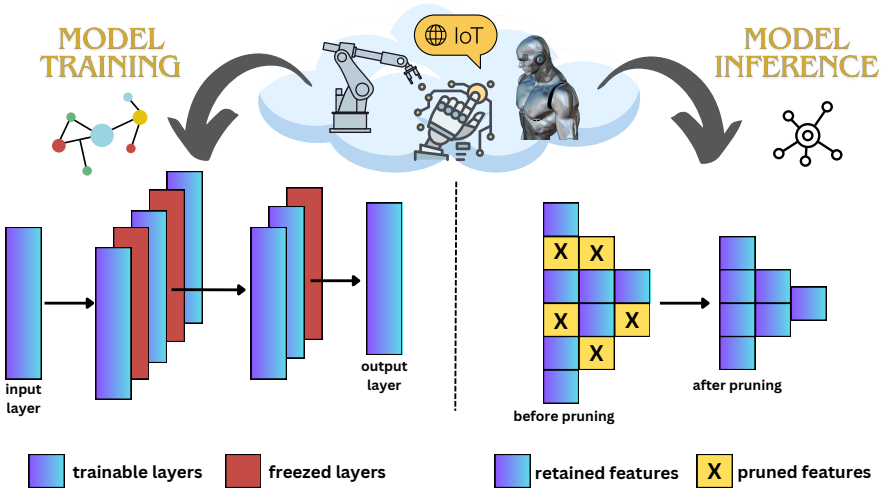

Figure 5.1: An overview of our proposed methodology.

Optimising ML pipelines and system life cycles often presents significant challenges and is a very complex task. Training larger neural networks with sparse activations can enhance model scalability and performance. However, this approach may lead to increased carbon emissions and energy utilisation due to higher demand of system resources [141]. Offloading model training and inference tasks to data centres powered by carbon-neutral energy sources offers a potential reduction in emissions but might not be practical for all application use cases. This is because the development of carbon-neutral infrastructure is often constrained by geographical and material availability limitations [142]. Moreover, with the rising trend of on-device learning to enhance data privacy, more computational tasks are being offloaded from centralised servers to low-level edge and IoT devices [143, 144]. Therefore, there is a critical need for a sustainable training and inference infrastructure that reduces resource utilisation – both in terms of memory and computation – without compromising the accuracy and performance of the models.

In this chapter, I am evaluating two distinct methodologies that enhance the efficiency of ML operations – (1) The first method aims to reduce the computational demands associated with backpropagation by systematically freezing certain layers of the neural network. Specifically, this involves fixing the parameters of selected layers to prevent them from updating during training. The primary objective is to cut down on the time required for both the current training session and any subsequent adjustments to model in the future. (2) Another method focuses on refining the architecture of an ML model by adjusting its parameter count. This strategy uses an automated process to identify and prioritise the most significant parameters for a specific dataset on which it is trained. Subsequently, parameters that are found less critical are eliminated from the model through a pruning procedure. This approach enhances the overall efficiency of ML

operations for both the training and inference phases, by reducing the model's complexity. A brief overview of our proposed approach is shown in Figure 5.1. Implementing these strategies results in an optimised configuration of models such that models require fewer resources for training and inference thus reducing the energy, computation, and storage requirements of the frameworks.

5.2 Agricultural Use-Case

I have considered an agricultural scenario for experimentation with our approach. Effective weed management in precision agriculture is a critical task that can help farmers in maximising crop yield, reducing cost, and minimising the use of herbicides in agricultural fields. AI models trained on large datasets of agricultural imagery are capable of distinguishing between crops and weeds with remarkable accuracy. These models are deployed through advanced sensors and imaging technologies mounted on drones, robots, agricultural machinery, scanning fields in real time to identify weed infestations. For evaluation, I have utilised a weed identification task as the use case in this chapter. Two ML models namely InceptionV3 and VGG16 have been used for training and inference procedures with DeepWeeds dataset in the approach.

5.3 Proposed Methodology

This subchapter provides a detailed description about our approach for layer selection and pruning. The first part describes the genetic algorithm based selection mechanism for layers and the second part highlights the heatmap visualisation technique for channel pruning.

5.3.1 Optimising Layer Selection with Genetic Algorithm

In optimisation, the solution encoding consists of a binary array that defines each layer's selection during the training process. A genetic algorithm is used to manipulate this encoding by modifying the array to discover an optimal mix of active and inactive layers for enhancing model's performance. Selecting the optimal layers for training while keeping the remaining layers frozen to preserve their weights during training, constitutes a complex computational optimisation challenge. This task is crucial for improving the model's functionality, especially considering the complexity and scale of neural networks which make manual selection non-feasible. Therefore, sophisticated optimisation algorithms are necessary to efficiently navigate through potential configurations and identify the most effective ones for deployment.

Genetic algorithms are particularly suited for this task due to their good capability in handling complex optimisation problems. Inspired by the genetic process of combining attributes, these algorithms utilise mechanisms such as mutation, crossover, and selection to refine solutions progressively. For the layer selection approach, GA enables a systematic

exploration of layer combinations, identifying those layers that can significantly reduce the computational demand and enhance the performance. I will first describe the formulation using mathematical notations and then provide details about the selection mechanism.

Problem Formulation

I have formulated our selection approach as an optimisation problem to maximise the accuracy of a neural network model by strategically selecting layers for training. This selection process is controlled by binary decision variables for each layer, denoted as:

$$x_i = \begin{cases} 1, & \text{if layer } i \text{ is selected for training,} \\ 0, & \text{otherwise layer } i \text{ is } NOT \text{ trained.,} \end{cases} \tag{5.1}$$

for $i = 1, \ldots, n$, where n is the total number of layers in the model.
The primary objective is to maximise the model's accuracy, which can be represented by the objective function $f(x)$ described as:

$$f(x) = \text{model\_accuracy}(M(x)), \tag{5.2}$$

where $M(x)$ represents the neural network model, configured according to the trainable status of each layer i, as determined by the binary decision variable x_i. This function directly maps the selection of trainable layers to the overall performance of our model.
The solution's feasibility is constrained by a limit on the total computational resources allocated for training the selected layers. This constraint ensures that the sum of the computational weights of the selected layers does not exceed a predefined limit L. It can be specified as:

$$\sum_{i=1}^{n} W_i \cdot x_i \leq L, \tag{5.3}$$

where W_i denotes the computational cost or weight associated with training layer i.
The aim of this formulation is to find an optimal set of layers (x_i) that maximises the neural network's accuracy within the given computational cost L. By identifying the ideal set of layers which can be used to establish a balance between performance accuracy, resource utilisation, and computational cost; we can design an approach that performs the ML model training based on the execution requirements of end-user applications for which the solution is begin designed.

Genetic Algorithm Based Selection

For the framework, the solution encoding is represented by a binary array which determines the training status of each layer. A genetic algorithm modifies this array to find the optimal configuration of layers to improve model performance. Figure 5.2 illustrates our solution encoding mechanism and its manipulation by the genetic algorithm. The selection algorithm chooses solutions in each iteration, followed by the crossover and mutation

operations. Mutation involves flipping bits ('0' to '1' or vice versa) by targeting specific indices based on a random distribution. These operations are repeated multiple times to generate new probable solutions.

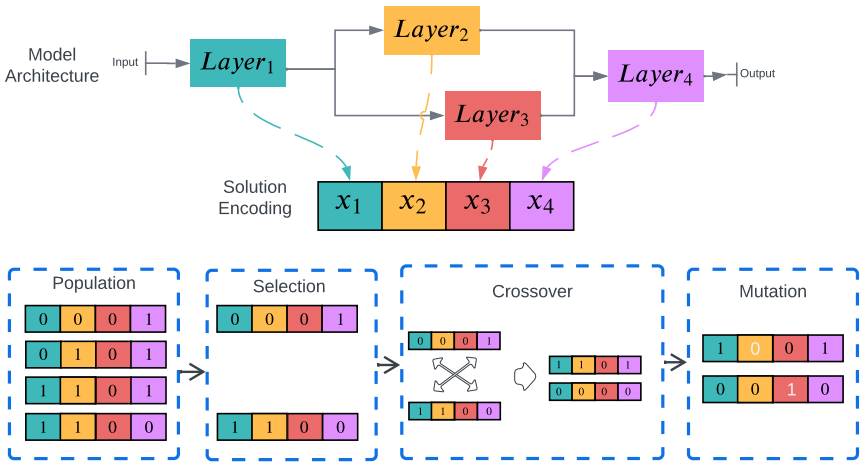

Figure 5.2: Solution encoding for layer selection.

Individuals with higher fitness are preferably selected through *Binary Tournament Selection* mechanism, ensuring those more suited to the problem are more likely to reproduce. A 90% chance of *Single Point Crossover* promotes genetic diversity by mixing genes from pairs of parents, while a 20% *Bit Flip Mutation* rate introduces new genetic variations to explore the solution space effectively. The algorithm runs until a specific number of fitness evaluations are obtained, maintaining a constant population size. This setup provides a balance between exploring new possibilities and exploiting already-known good solutions, aiming for an efficient search of optimal solution within a limited number of computational steps.

5.3.2 Efficient Feature Mapping for Pruning

Modern deep neural networks consist of various types of convolutional layers, and the execution runtime during an inference phase is dominated by evaluation of those layers. In order to speed up the inference process, our strategy involves pruning complete feature maps by utilising the analytical capability of GradCAM and performing heatmap visualisation to identify the impactful features.

Introduction to GradCAM

GradCAM – stands for Gradient-weighted Class Activation Mapping [145], is a technique that can enhance the interpretability of convolutional neural networks (CNNs) by analysing gradients in the final layer and quantifying the significance of each neuron in the decision-making process. This technique visualises the importance of features using heatmaps, where color intensity represents values between 0 and 1 to show the scale of significance in the data. Figure 5.3 below shows a step-by-step visualisation of the active areas in an image that was highlighted during the feature identification process.

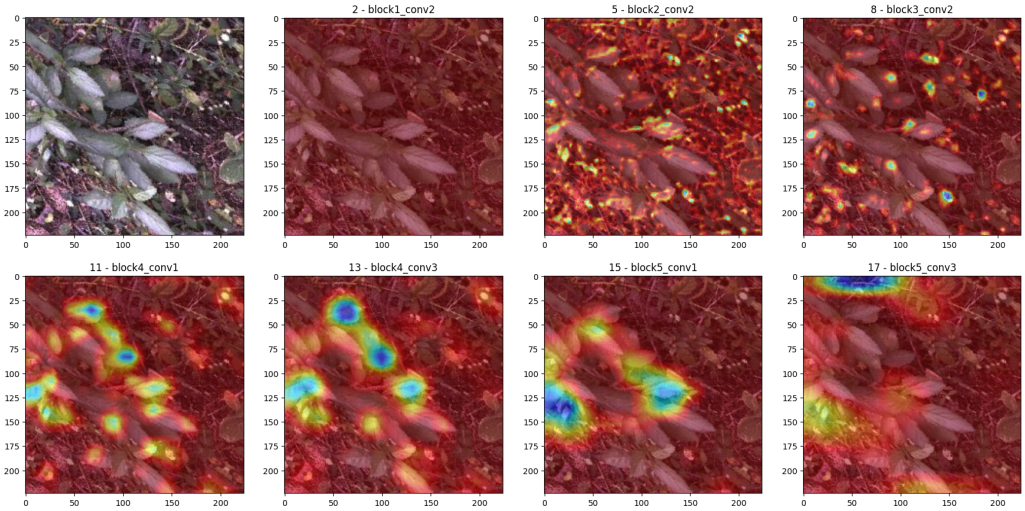

Figure 5.3: Visualisation of the data and its corresponding class activation heatmap for identified labels.

The method is utilised to identify the crucial areas and bring out the relation between model parameters and these identified regions. By highlighting the association between parameters and weights with important features or regions, the technique can be used for the interpretation of AI models as well as improving their performance. GradCAM quantifies each weight's importance in values ranging between 0 to 1 and uses it to measure the parameter's contribution in class identification decision. I have considered a threshold variable for establishing the separation between the importance value of features and executing the pruning process to eliminate less critical connections, thus optimising model performance and efficiency.

To represent it mathematically, let us consider that C is a set of all channels in the convolutional neural network's target layer. $I(c)$ is the importance score of channel c, derived from the GradCAM heatmap values, where $c \in C$. Moreover, T is the threshold value for pruning, a predetermined parameter that distinguishes between essential and non-essential channels.

For a channel c to be retained, its importance score must be greater than or equal to the threshold otherwise the channel is pruned, such that:

$$retainCh(c) = \begin{cases} yes, & I(c) \geq T \\ no, & I(c) < T \end{cases} \tag{5.4}$$

The aim is to identify the optimal value of *retainCh(c)* such that majority of the necessary features are retained whereas all the other features that do not significantly contribute to the decision making process are pruned. By optimally selecting limited number of features, we plan to reduce the overall size of ML model and retain all the necessary features which contributes in the delivery of output decision obtained during model inference process.

Pruning Workflow

I have designed the Algorithm 5.1 that shows the step-by-step approach of our pruning workflow. The process begins with the identification and extraction of the target CNN layer from the neural network model. Following this, the weights of the designated layer are extracted before further analysis and manipulation are performed. Next, the channel importance is calculated by summing up heatmap values per channel, indicating their contribution to the model's output and assistance in pruning decisions.

The selection of channels to be retained is based on a pre-specified importance threshold value. Channels with cumulative heatmap values surpassing this threshold are selected for retention, while those with values below it are considered for removal. This process ensures the retention of only those channels that significantly contribute to the network's effectiveness. The pruning phase involves the targeted deletion of weights and biases associated with channels that are deemed non-essential (below the threshold). This crucial step aims to decrease the model's computational demands while preserving its true predictive characteristics.

Following the pruning process, a new CNN layer is created. This layer mirrors the original layer in terms of its architectural parameters (e.g., kernel size, strides, padding, activation function) but differs in the number of filters, which are adjusted to match the count of channels retained. The pruned weights and biases are then allocated to the newly instantiated CNN layer, ensuring that the layer is properly equipped to perform its function within the network. The final step replaces the original convolutional layer with its pruned version, ensuring the model's integrity and connectivity.

Algorithm 5.1 Channel Pruning using Heatmaps

```
1: Input: model, layerName, impThresh
2: Output: prunedModel
3: targetLayer ← GetLayer(model, layerName)
4: w, b ← GetWeights(targetLayer)
5: imp ← empty map
6: for each ch in targetLayer do
7:     imp[ch] ← Sum(heatmap[ch])
8: end for
9: retainCh ← empty list
10: for each ch, i in imp do
11:     if i > impThresh then
12:         Append(retainCh, ch)
13:     end if
14: end for
15: prunedW, prunedB ← PruneWB(w, b, retainCh)
16: newLayer ← CreateLayer(len(retainCh), targetLayer.kSize, targetLayer.stride,
    targetLayer.pad, targetLayer.act)
17: SetWeights(newLayer, prunedW, prunedB)
18: model ← ReplaceLayer(model, layerName, newLayer)
19: return model
```

5.4 Experimental Design and Setup

The subchapter describes the experimental tools and detailed configurations that have been utilised for evaluation of our approach.

5.4.1 Dataset and Hardware Configuration

For both layer selection and pruning, I have used TensorFlow to build and modify state-of-the-art models including VGG16 and InceptionV3. I have also used JMetalPy[1] for genetic algorithm operations in layer selection. For pruning, I leveraged TensorFlow[2] and Keras built-in capabilities to implement a GradCAM solution. The heatmap generation in the approach uses TensorFlow's Keras API[3], where a function builds a model to observe a specific convolutional layer's output and model's predictions for an image array. This process generates heatmap values highlighting key image regions influencing the model's decisions, showcasing API's ability to analyse and interpret model behavior.

I performed our AI operations on a virtual instance of dual-core Intel(R) Xeon(R) CPU at 2.00GHz, 12 GB of RAM, and an NVIDIA Tesla T4 GPU with 16 GB of memory. The system has 79GB of total storage, with 27GB used and 52GB available. Datasets were pre-loaded from Google Drive as virtual instances to minimise data transfer overhead and optimise the performance.

Our agricultural use-case considers collecting data from IoT devices at the network's edge, processing it on a FSU, using the DeepWeeds [32] image classification plant-based dataset. The DeepWeeds dataset is the first large, public dataset for Australian range-land weeds, having 17,509 images of eight common species of weeds from eight regions spread across northern Australia. This dataset is designed to support the development of classification methods, enabling the use of robotic weed control in challenging environments and highlighting the role of machine learning in improving weed management and agricultural tasks.

5.4.2 Estimation of Power Consumption

To estimate the power consumption of Nvidia T4 GPU, based on its running time, I have used the following approach:

Let P_{load} be the power consumption in watts when the resource node (GPU) is running and let T_{load} be the total time for which our node is executing the tasks.

Given that $P_{load} = 74$ watts. The total power consumption (P_{total}) in watt-hours (Wh) can be calculated as follows:

$$P_{total} = (P_{load} \times T_{load}) \tag{5.5}$$

[1] https://github.com/jMetal/jMetalPy
[2] https://www.tensorflow.org
[3] https://www.tensorflow.org/guide/keras

Similarly, in kilowatt-hours (kWh), the power can be calculated as:

$$P_{total} = (P_{load} \times T_{load}) / 1000 \tag{5.6}$$

This model allows us to estimate the T4 GPU's power consumption over the duration for which GPU was consuming power and executing the ML/AI tasks utilised for evaluation in our experimentation.

5.4.3 Experimental Configuration

This subchapter describes the configuration details for performing the layer selection and feature pruning mechanism in this work.

Layer Selection Setup

The first part of our configuration focuses on setting up the layer selection mechanism for the approach. The steps are as follows:
Data Splitting: The dataset is divided into three segments: 60% for training, 20% for validation, and 20% for testing. This division is used to ensure effective model training, fine-tuning, and performance assessment of our approach.
Data Preparation: The process begins with resizing images to standardised input dimensions, followed by data augmentation techniques such as flipping and rotating. Additionally, these images are normalised by converting pixel values from integers to floats and scaling them to the range of $[0, 1]$ and then batched for efficient processing during the training process.
Model Customisation: This phase involves updating pre-trained models, specifically InceptionV3 and VGG16, which were initially trained with the ImageNet dataset. The models are customised for DeepWeeds dataset by appending new layers and freezing the original layers to preserve learned/retained features. This customisation aims to leverage the pre-existing knowledge of the models while making them relevant for the new tasks. During this process, accuracy metrics are tracked to evaluate the objective function's performance.
Training Setup: The models are trained with hyperparameters including 10 epochs and batch sizes of 20. Additionally, the 0/1 knapsack algorithm is utilised to fine-tune the non-trainable (frozen) layers of the model, enhancing its customisation and improving performance on the new dataset.
Genetic Algorithm Based Selection: A similar model undergoes brief training to learn the changes and improve model adaptability. The generated models are then evaluated with a focus on accuracy and other key performance metrics. The obtained results are used for further genetic algorithm driven refinements, continuously improving model outcomes. The genetic algorithm population considered in our experimentation is 10.
Evaluation and Analysis: After training, the models are evaluated on the test dataset to determine their generalisation capabilities. This evaluation helps in assessing the

effectiveness of the training and the customisation process. The performance metrics, specifically training, validation accuracy and loss, are plotted over the epochs to analyse the model's learning behavior and fit to the data, providing insights into their strengths and remaining challenges.

Feature Map Pruning Setup

The second part of our approach is designed to use a heatmap visualisation mechanism for feature pruning. The steps are as follows:
Data Preparation: The process begins by dividing the complete dataset into two separate parts (referred to as Dataset A and Dataset B) to simplify the training and evaluation process.
Initial Training: During this phase, the Dataset A is utilised as a base for the model's initial training process. This phase establishes the benchmark for the model's performance and provides insights about its learning capabilities.
Pruning Phase: Following the initial training, the model undergoes a pruning process using the following methodology – I selectively removed some of the model components, such as weights, biases, or neurons, that have minimal impact on the output. The aim is to simplify the model's structure and enhance its operational efficiency. I have used the approach outlined in subchapter 5.3.2 to selectively remove the model components and reduce its size.
Fine-Tuning with Dataset B: After pruning, Dataset B is used to fine-tune the selected model. This phase aims to perform minor adjustments to our model's parameters to refine its performance, focusing on recovering or improving aspects, generalisations that potentially got diminished or removed during our earlier pruning phase.
I have used both Dataset A and B as training sets within this experimental approach. The only key difference is in their usage such that Dataset A is allocated for initial model training whereas Dataset B is designated for post-pruning fine-tuning.
Evaluation and Analysis: The final step involves an evaluation of the model after its fine-tuning. This analysis focuses on various performance metrics such as execution time and accuracy, to verify the rate of improvements I was able to achieve during the pruning and fine-tuning phases.

5.5 Results and Evaluation

This subchapter highlights the experimental results and their findings after evaluating the proposed approach. Figure 5.4 compares the training times (in seconds) for one epoch of the InceptionV3 and VGG16 models across different percentages of trainable layers, ranging from 10% to 100% of the total layers in each model. The training time increases for both models as the percentage of trainable layers increases, which is expected since more parameters require more computational effort to update the model. Moreover, VGG16 has consistently higher training time across all percentages of trainable layers

than InceptionV3. This can be attributed to the architectural complexities and the overall number of parameters in VGG16, which is known for its simplicity in structure but high computational cost due to the depth and size of its fully connected layers.

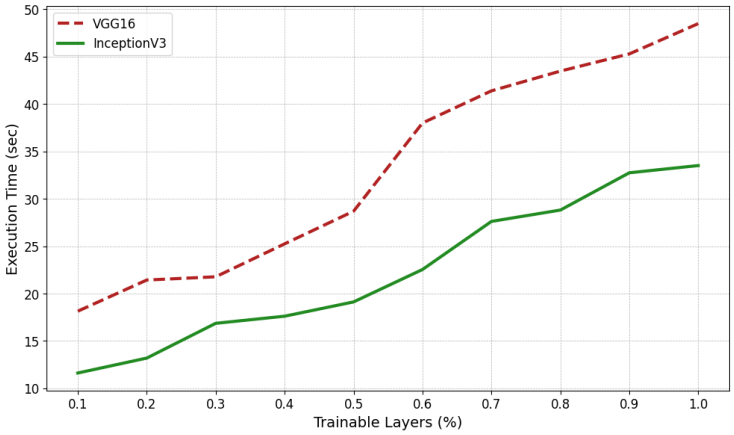

Figure 5.4: Total execution time with different percentages of trainable layers.

The Table 5.1 below summarises the power consumption, measured in watts, of two deep learning models (per epoch) during the training sessions with different percentages of their layers trainable. Specifically, the table reports measurements for three scenarios: when 10%, 50%, and 100% of the layers are trainable. For both the models, the power consumption increase as the percentage of frozen layers decreases.

Trainable (%)	***InceptionV3***	***VGG16***
0.1	11.61	18.13
0.5	19.11	28.68
1.0	33.50	48.48

Table 5.1: Power consumption (watts) of models during training (one epoch) with different percentages of layers trainable.

This progressive increase in power can be attributed to the computational demand associated with updating a greater number of parameters. With more layers participating in the training process, the models require more computational resources, leading to higher energy usage. The comparison between InceptionV3 and VGG16 also reveals the inherent differences in their architectural efficiency. InceptionV3 consistently requires less power than VGG16 across all scenarios, indicating it is a more energy-efficient model for training tasks. This can also be due to its more optimised architecture, which might involve fewer parameters or more efficient operations compared to the deeper and more parameter-intensive architecture of VGG16.

The Figure 5.5 outlines the validation accuracies of two distinct convolutional neural network models considered for optimisation in this work. I performed the model training utilising the layer selection mechanism described above in previous subchapters. The accuracies are measured across a span of 10 epochs, providing insight into the learning efficiency and performance of each model over time. InceptionV3 demonstrates a starting

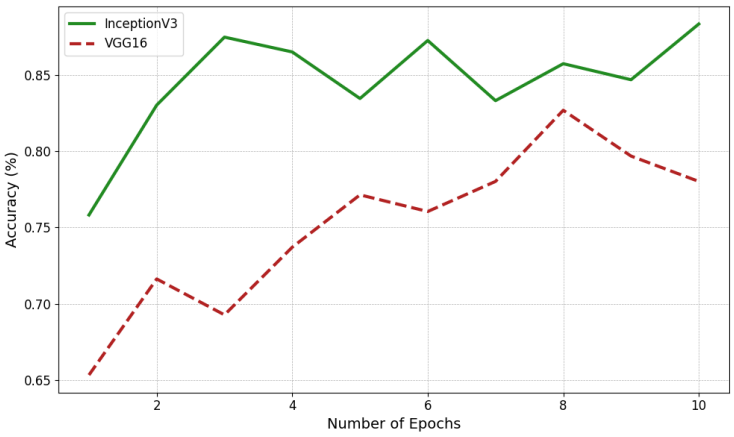

Figure 5.5: Accuracy benchmarks on DeepWeeds dataset for model training.

accuracy of 75.81% at epoch 1, which exhibits a generally upward trend, peaking at 88.32%. The overall trend indicates improvement in model performance with continued training and shows the strong ability of InceptionV3 to generalise from training data to unseen validation data. On the other hand, VGG16 starts with a bit lower accuracy of 65.33% at epoch 1 and follows an upward trend by peaking at 82.67%. The observed differences in accuracies between InceptionV3 and VGG16 can be attributed to factors inherent to their architectures and complexity of the task. InceptionV3 consists of an inception module, which is designed to efficiently manage computational resources while capturing complex features at various scales. This design contributes to its higher accuracy and ability to better generalise during the validation phase. VGG16, characterised by its simplicity and depth with repetitive convolutional blocks, demonstrates a considerable capacity for feature extraction. However, its architecture sometimes has a higher susceptibility to overfitting and requires more data or regularisation to achieve optimal generalisation, which explains its lower performance compared to InceptionV3 in this scenario.

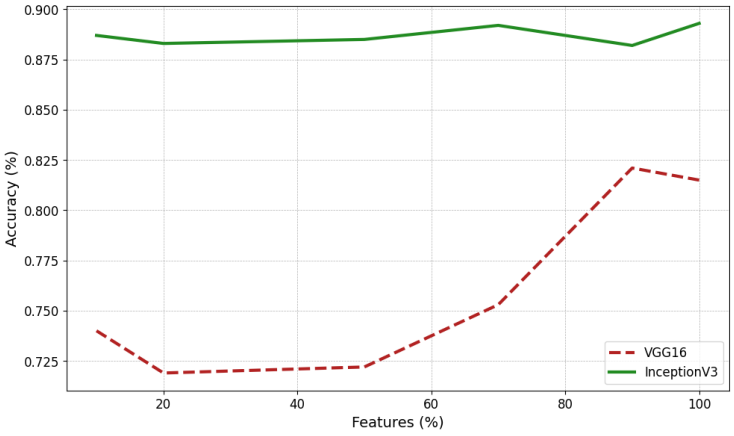

Figure 5.6: Change in model accuracy with features retained above the threshold.

The proposed heatmap based pruning approach is also evaluated on two available CNN models – VGG16 and InceptionV3. Figure 5.6 shows that across both the models, there is a general trend where accuracy increases when a higher percentage of features are

retained. This pattern is observed as retaining more features likely preserves more relevant information necessary for accurate predictions. Models with deeper or more complex architectures, like the VGG, show a more significant impact from feature pruning, requiring a higher percentage of features to be retained for optimal accuracy. InceptionV3's design, incorporating modules with parallel convolutions of different kernel sizes, allows it to capture information at various scales effectively thus ensuring high accuracies across all percentages.

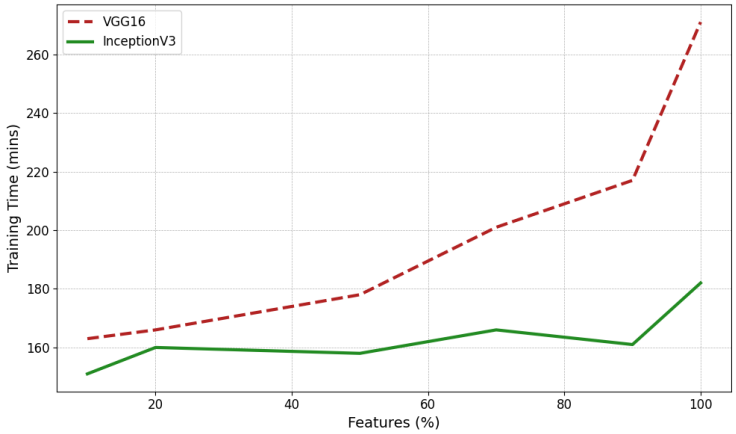

Figure 5.7: Effect on model training time with features retained above the threshold.

The pruning process aims to reduce the computational complexity of the models by selectively removing less important channels based on their contributions, as determined by heatmaps in this work. This selective reduction can significantly affect the training time of the models, as observed in Figure 5.7. The VGG16 model shows an increasing trend in training time as the percentage of features retained increases. Over the range of feature percentages, the training time increases from 163 minutes to 271 minutes gradually. The significant reduction in training time indicates that the pruning process effectively removes redundant or less important features without compromising the model's predictive power significantly. For the InceptionV3 model, the training time initially decreases when the percentage of retained features goes from 10% to 50%; and then increases slightly when feature retension percentage is increased to 100%. This suggests that InceptionV3 may achieve optimal performance with a moderate level of feature retention, where the balance between model complexity and computational load is ideal. Beyond this point, the slight increase in training time as more features are retained could be due to the added computational burden outweighing the benefits of additional features.

The size of a model is directly related to its complexity and the amount of information it can process and store, as shown in Figure 5.8. The increase in model size with an increase in the percentage of features retained indicates a nearly linear relationship between the two factors. More features mean more channels in the convolutional layers, which in turn increases the number of weights and biases that need to be stored, thereby increasing the model size. The InceptionV3 model shows a more complex pattern. The size increases from 174.405 MBs at 10% feature retention to 217.229 MBs at 100% feature retention. It

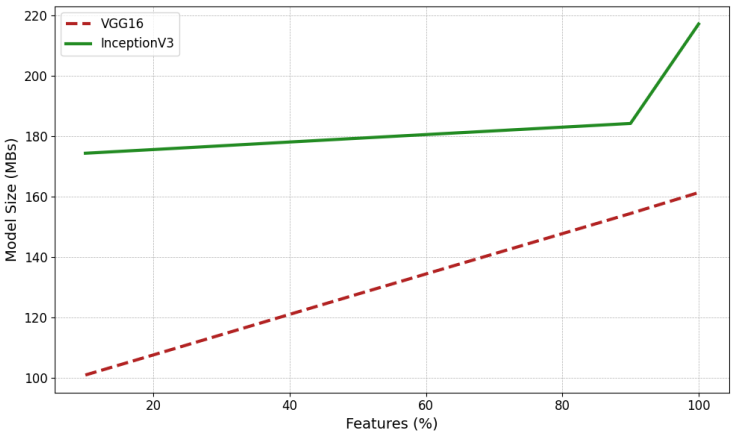

Figure 5.8: Change in model size with features retained above the threshold.

is observed that the increase in size is not as linear as with VGG16, especially in the higher percentages of feature retention. The larger initial size of InceptionV3 compared to VGG16 at lower feature retention levels suggests that even with fewer features, InceptionV3's complex architecture requires more parameters to be stored. This complexity contributes to the overall increase in model size as more number of features are retained.

Features (%)	***InceptionV3***	***VGG16***
0.1	164.03	219.53
0.5	194.86	238.03
1.0	224.46	334.23

Table 5.2: Power consumption (watts) of models after pruning with different percentages of retained features.

The given Table 5.2 illustrates the effect of pruning process on power consumption of both the models at specific levels (10%, 50%, and 100%) of retained features. For both models, the power consumption increases as the percentage of retained features increases. The data indicates that VGG16 generally consumes more power than InceptionV3 across the same levels of feature retention, suggesting that VGG16's architecture may be inherently more power-intensive or less efficient at these high levels of feature reduction. Additionally, the increasing trend in power consumption with higher feature retention percentages highlights the relationship between model complexity and power needs. This suggests that even slight increases in retained features can significantly impact power consumption, likely due to the increased computational workload associated with processing a higher number of features. Analysing the process of structured training and fine-tuning process, which involves initial training with Dataset A, followed by model pruning and subsequent fine-tuning with Dataset B, provides a comprehensive understanding of the approach and its implications on the model performance. This approach utilises the distinct characteristics of two datasets to enhance model performance progressively. The initial training on Dataset A establishes a foundation, allowing the model to learn general features and patterns. The pruning process then refines the model by eliminating less significant features, reducing

complexity and computational demand. Finally, fine-tuning with Dataset B adjusts the model to perform well in specific, possibly more challenging, real-world scenarios. This method highlights the adaptability of the models to new information and their capacity for incremental learning. The maximum accuracy achieved after re-training by VGG16 and InceptionV3 models is 86.8% and 93.1% respectively. Comparing these results with the accuracies recorded after the pruning process, both models exhibit significant improvements in accuracy after fine-tuning. The VGG16 model observed an average of 6.3% improvement over the initial results whereas InceptionV3 achieved improvements of 4.3%.

5.6 Analysis of Results

The experimental results highlight the differences in performance dynamics of VGG16 and InceptionV3 models under varying configurations of trainable layers, with particular emphasis on training time, power consumption, and accuracy benchmarks. I observed that VGG16 has longer training times and higher power consumption across all degrees of trainable layers. On the other hand, the InceptionV3 has higher accuracy and model size over the different percentages of retained features. The InceptionV3 model performs about 32.63% better than the VGG16 model in terms of execution time with different values of trainable layers. The average accuracy of InceptionV3 model is around 6%-7% higher than VGG16 across the variable degree of retained features in our approach. As the percentage of retained features decreased, the model size for InceptionV3 and VGG16 decreased by 20.27% and 37.5% when the ratio of features was reduced from 100% to 10%. Moreover, I also observed that the average power consumption of InceptionV3 model is 26.33% more efficient than the VGG16 model. The VGG16 model showed a 66.26% decrease in training time when the number of features are reduced to 10%. In contrast, the InceptionV3 model exhibits a 27.27% decrease over the same measure. The results also show that the InceptionV3 model is approximately 18.90% more efficient in training time compared to the VGG16 model. Comparing these two architectures highlights that the model structure, number of features, and trainable layers are some of the critical parameters that directly affect the performance of AI models and can be a deciding factor for the efficient implementation of ML optimisation approaches on limited resource nodes. Moreover, based on the time, accuracy, and space requirements of limited resource nodes; a decision can be made using our approach to select the optimal values of parameters that will generate the best possible results for the end-users and applications for which the ML models are designed.

5.7 Aspects for Further Optimisation

I observed that our proposed approach achieves desirable results in reducing the training time and model size, however, in order to contribute towards more sustainable practices

for IoT applications, it does present certain limitations that can provide options for future exploration and optimisation.

Our current approach for layer selection is managed as a single-objective optimisation problem, where the number of layers are predetermined by the user. For improvement, I can consider the number of layers as an additional objective function along with model accuracy, using a multi-objective optimisation method. This will allow for a balanced trade-off between maximising accuracy and minimising the number of layers, addressing this problem as a connected optimisation challenge.

Moreover, our pruning approach is currently considered as a constraint satisfaction problem with an aim to meet specific conditions defined by a predetermined threshold. For improvements, I can consider the pruning approach as an optimisation challenge, rather than simply satisfying the set constraints. This adjustment can decrease the variability and transform the pruning threshold value into an adjustable variable. Objective functions can then be used to evaluate the most effective areas for pruning, identifying the optimal threshold that aligns best with a particular model and dataset. Furthermore, these objective functions can prioritise two main factors – accuracy and inference time, as both of these are important for developing an efficient model suitable for real-world IoT applications.

From our experiments, I also observed that each neural network has distinct characteristics that can significantly influence the model optimisation framework. These characteristics include layer size, total number of parameters, model branching (which allows for the concurrent execution of two or more layers), and the parameter count within individual layers. I can explore the possibility of establishing a method that examines these specific aspects of each network to support ongoing testing and improvements. By understanding and addressing these elements, I aim to restructure models to promote energy-efficient and sustainable machine learning operations, thereby improving the performance of ML based tasks in IoT infrastructures.

5.8 Summary

The chapter introduces a sustainable optimisation approach for enhancing the efficiency of ML tasks in the IoT based infrastructures. Our approach leverages the ability of genetic algorithm for layer selection and heatmap visualisation technique for pruning mechanism. Through systematic experimentation with VGG16 and InceptionV3 models using the DeepWeeds dataset, I have demonstrated that our proposed methodology significantly reduces model size and training time, without compromising on accuracy. Experimental results show that the strategy of selectively freezing neural network layers and pruning less significant parameters addresses the critical challenge of limited-resource constraints of IoT devices. The approach also highlights the potential procedure for deployment of ML and AI capabilities in environments where computational, storage, and energy resources are in limited capacity. The work can be extended to other AI models and

different real-world IoT applications to understand domain-specific challenges and adjust the approach accordingly. However, this approach considers that the training, evaluation data, and ML models are available in IoT applications all the time. In a real-world scenarios, it is highly possible that there can be a form of node crash or failure which will limit the availability of data in IoT applications. Therefore, the next chapter explores the data management of applications by ensuring proper access, distribution, and availability of data in case any unexpected failure occurs.

Chapter 6

Data Management in Edge-Cloud Environment

This chapter explores the aspect of data management in IoT applications that utilise serverless edge-cloud environments. As the available data is prone to failure and loss due to system crash or any node fault; reliable storage, access, and retrieval of data is a critical factor for ensuring high performance of the system.

6.1 Data Management in IoT

The rise of IoT represents one of the most significant technological transformations in the modern era. As a network of interconnected devices, IoT integrates physical objects with embedded sensors, software, and other tools to collect and exchange data, facilitating a new level of automation and integration in everyday systems. This evolution has been adopted in almost every sector, from healthcare and agriculture to manufacturing and urban development, providing its services [146]. This adoption of IoT across diverse applications can be attributed to its ability to enhance data communication, streamline processes, and improve the efficiency and accuracy of data-driven decisions.
However, the rapid expansion of IoT also brings an enormous amount of data with itself. Each device connected to the IoT network generates a continuous stream of information. This large amount of data presents not only opportunities but also significant challenges in terms of storage, management, and analysis. Cloud computing, edge computing, and advancements in AI are increasingly crucial for managing big data, ensuring that the information is processed and utilised efficiently. Utilising the serverless approach of computation with the IoT and ML is a popular option to handle data and execute tasks in modern computing infrastructures. I can utilise its strengths, creating a scenario that enhances efficiency, scalability, and cost-effectiveness in handling large amounts of data and complex computations, particularly within edge-cloud environments [147].
Serverless computing is a paradigm in which the allocation of resources is managed by service providers without the need for users to configure underlying servers. It offers several distinct characteristics that can be beneficial in ML-based IoT applications [20]. Firstly, it can provide automatic scaling of the infrastructure. IoT applications can have variable workloads depending on the traffic, and serverless platforms can dynamically adjust computing resources to meet the fluctuating demands of millions of connected IoT devices. Another key benefit of serverless computing is its event-driven nature.

IoT devices can generate events in real time, from sensor data indicating a change in environmental conditions to alerts triggered by user interactions. Serverless architectures are designed to respond to such events instantaneously, initiating specific functions or computations without any delay. This is highly effective in edge-cloud environments, where latency-sensitive applications require faster decision-making close to the data source. Cost efficiency is another significant advantage of serverless platforms. It allows organisations to only pay for the computation time they consume, with no cost associated with idle server capacity. This can lead to substantial cost reductions in large-scale deployments of IoT networks, where devices and end-users have to execute jobs at unspecified intervals. [33]
IoT data management for ML-based tasks deployed on serverless platforms has many challenges related to storage and resilient execution that need serious consideration. The following subchapters describe in detail the data handling mechanisms associated with serverless platforms and ML operations on edge-cloud frameworks. I explore the implications of data loss during task execution and data storage phase. An efficient and optimised approach for data and model management is proposed in this chapter to address these challenges.

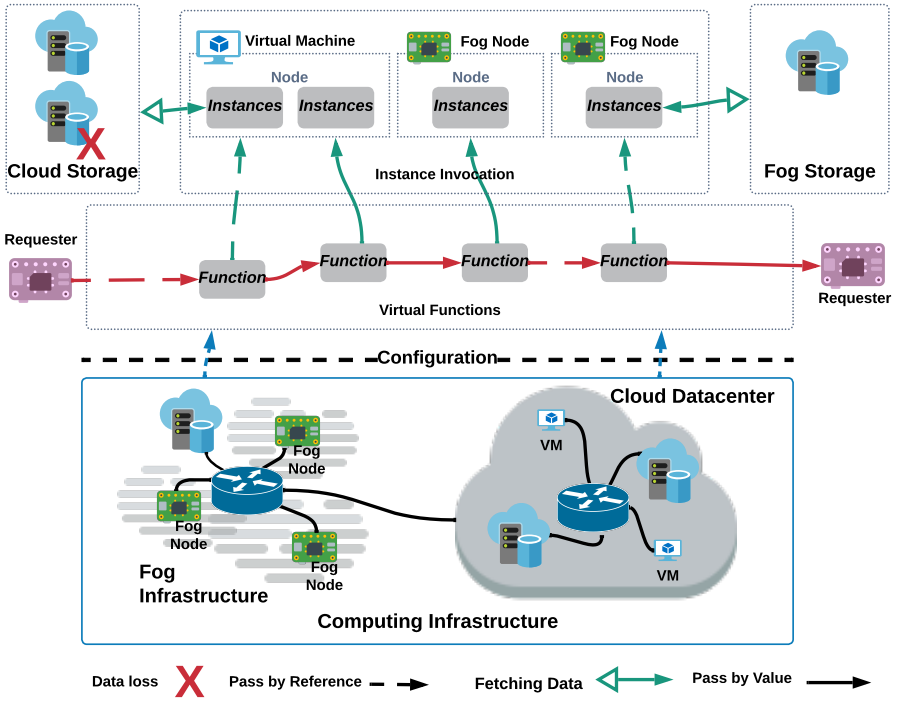

Figure 6.1: An architecture for data handling in serverless platforms.

6.2 Data Handling in Serverless Environment

Serverless computing involves executing functions in a stateless manner, which complicates data management. Figure 6.1 illustrates the two main types of data handling in a serverless environment. The approaches are as follows:

6.2.1 Direct Data Transfer

This method involves directly transferring data between functions, where the serverless platform actively manages data duplication. If one function generates data as its output, and this output is forwarded to multiple subsequent functions, the serverless platform will handle the replication of this output data to each recipient function. This approach ensures that each function receives the data it requires to operate. However, such duplication can impose an additional load on the serverless platform, potentially increasing the time it takes for data to be transferred from one function to another as the platform manages the copying process. Moreover, this approach introduces the issue of data volatility. Data is streamed through input-output buffers and network channels, making it susceptible to loss in case there is an issue with the server or there are any disruptions in the network. Therefore, while direct data transfer can be efficient, it also has significant risks due to the volatile nature of data handling.

6.2.2 Use of Intermediate Storage

This approach involves storing data in an intermediate system such as a broker or file system, which acts as a central repository. Functions pass references to the data, rather than the data itself, minimising direct data transfer. The intermediate storage system manages access and replication among functions. However, challenges arise related to the physical locations of function execution and data storage. For instance, if functions are executed on servers in Europe but the data is stored in India, significant delays can occur due to the time required to transfer data across these distances. This latency issue, arising from geographical disparities between servers and storage, can impact performance due to extended upload and download times, and poses a challenge in maintaining efficient connectivity and replication.

6.3 Methods for Ensuring Data Availability

For IoT applications, the traditional data availability techniques focus on ensuring that data is always accessible for processing and analysis while maintaining integrity and performance. These techniques can be broadly categorised into three categories, such that each of them handles availability in a different manner.

Data replication involves creating multiple copies of data across different nodes or locations to ensure high availability and fault tolerance. This approach is particularly advantageous for IoT environments where device or connection failures are common. By replicating data, systems can ensure that even if one node fails, others can take over without data loss or service interruption. However, replication increases the space cost significantly as multiple copies of data need to be maintained. The execution cost can vary as read operations are generally faster due to multiple data points but write operations become more costly and complex for ensuring integrity across all copies.

Data partitioning distributes different parts of a dataset across various database systems or storage locations. This technique is often used in big data scenarios where handling large datasets as a single unit is inefficient. By partitioning data, tasks can be executed in parallel, significantly improving performance and reducing execution time. However, the space cost does not necessarily increase unless redundancy is also introduced as part of the partitioning strategy. In this approach, the main execution cost is for handling the complexity of managing data integrity and consistency across partitions, which can be a significant overhead for many AI tasks.

Data compression reduces the size of the data through various algorithms, which is crucial when dealing with the large volumes of data generated in ML and big data tasks. Compressed data requires less storage space, effectively reducing space costs. It also potentially lowers data transfer times, which is useful in distributed systems designed for scalable IoT environments. However, the execution cost mainly includes the computational overhead required to compress and decompress data, which can be very high, especially with complex algorithms utilised in current IoT systems.

6.4 Challenges with Data Handling

Serverless frameworks when handling big-data and ML applications, encounter significant challenges related to data management and locality. A detailed description of these challenges is as follows:

Data storage and transfer: In ML, data is often stored for further training which requires a substantial amount of storage capacity. Transferring this data across different locations, such as from edge devices to cloud servers or between storage servers, can be an issue because of its size. This can significantly slow down the transfer process and increase communication costs. [20]

Resilience mechanisms: To address issues of data integrity and availability, resilience mechanisms such as erasure coding can be used [148]. Erasure coding helps ensure data durability and recoverability by utilising data fragments that can reconstruct the original data in case of partial loss. However, this technique introduces extra latency in data operations, as it requires additional time to generate and manage the encoded fragments.

Data locality and recovery: Optimising data locality (placing data close to the computation source) can enhance data access speeds and reduce latency, which is crucial for efficient processing and analysis in ML tasks. In scenarios where data is damaged or lost, having multiple copies or encoded fragments distributed across different locations enables effective data recovery. The process of encoding/decoding these fragments can protect data but also introduces additional delays and computational overhead.

Efficiency in data handling: Another challenge is to design an infrastructure that not only manages a large amount of data but also prepares it efficiently for immediate execution in ML operations. This requires new approaches that minimise data size and ensure availability for analysis, thereby reducing the time and resources needed for data

preparation.

6.5 Challenges in Deploying ML Workflow on Serverless Platforms

The implementation of ML tasks on serverless environments encounters data management challenges that can be mainly classified into two categories:

1. **Data Management:** It includes the steps associated with collection, preparation, and processing of datasets. A crucial aspect is to consider data availability and its distribution across various locations to facilitate access. Effective data management allows data to be strategically distributed, thereby enhancing both availability and reliability.

2. **Model Management:** This involves the iterative fine-tuning and periodic updates of ML models at edge of the network. Model management requires reducing the size of dynamically changing elements (such as parameters) and ensuring the availability of replicas to maintain data consistency. Considering that a model size can vary from small to extremely large, it is essential to design strategic ways to manage these variations effectively.

The integration of serverless architectures with ML frameworks necessitates new approaches for the management of data and models. Such approaches can ensure that despite the stateless nature of serverless platforms, essential data can still remain accessible and recoverable during ML operations, enabling a seamless and efficient approach for task execution.

6.6 Proposed Approach for Data Handling

This subchapter describes the proposed method to address challenges in data storage and processing within the context of ML and edge-cloud environments. The following aspects are considered in this chapter:

Data reduction: The approach begins by minimising the volume of data at the edge of the network, preparing it for subsequent processing. This reduction is aimed at optimising storage and processing efficiencies before the data is transferred to servers.

Standardised data format: Data is standardised into a format that is compatible with cloud-based machine learning operations. This standardisation is part of the preprocessing stage and improves the efficiency of subsequent processing steps.

Data compression and replication: To effectively manage data without excessively increasing storage requirements, the approach includes compressing the data before replication. This ensures that, even with the redundancy techniques in erasure coding, the data footprint is limited and controlled.

Data recovery and decompression: In case of any issues, the compressed data can be recovered and decompressed. The restored data can retain its pre-processed format, facilitating efficient transmission over networks without significant bandwidth consumption.

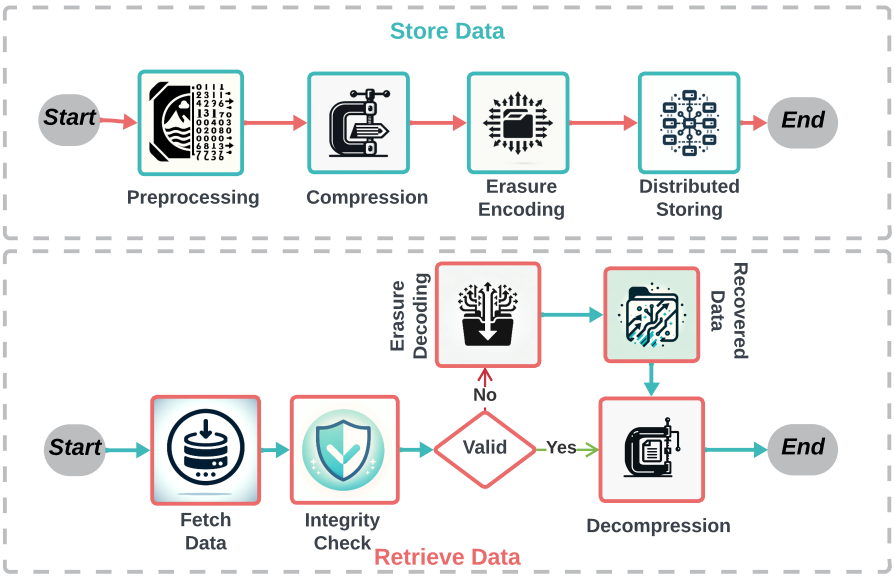

Figure 6.2: Proposed workflows for data handling.

A depiction of our data handling approach is shown in Figure 6.2. Our methods for optimising data management in serverless environments focus on balancing storage efficiency with network and processing demands. It enhances efficiency by storing the static structure of ML model separately from their dynamic parameters. A single instance of the model's static structure is maintained, and Marita coding is applied to the frequently updated parameters. This approach minimises redundancy and storage requirements, ensuring reliable protection of dynamic data elements against data loss.

6.7 Workflow for Data Handling

This subchapter describes the workflow that is utilised in preparing the data used for training and inference in ML models. The steps are as follows:

1. **Preprocessing:** The raw data undergoes preprocessing to optimise its utility for ML models. This step involves cleansing, normalising, and transforming the data, thereby enhancing processing speed, reducing its size, and preparing it for effective training sessions.

2. **Compression:** After preprocessing, the data is compressed using the Brotli algorithm. This process reduces the data size further, facilitating a more efficient storage and transmission.

3. **Erasure Coding:** Once the data is compressed, it is encoded using Reed-Solomon (RS) coding scheme [148]. This erasure coding technique introduces redundancy,

thus enhancing data resilience by enabling recovery in cases of corruption, damage, or connectivity issues.

4. **Storage:** The prepared data is then stored. At this stage, it has been preprocessed, compressed, and protected against potential data loss, making it ready for retrieval and use in our IoT application.

5. **Data Recovery and Decompression:**

   (a) *Integrity Check:* Upon retrieval, the integrity of the compressed data is first checked to ensure it has not been damaged or tampered with.

   (b) *Decoding:* If any corruption is detected, RS decoding is used to reconstruct the original data from the encoded fragments.

   (c) *Decompression:* Once the data is confirmed to be intact or successfully recovered, it is decompressed and restored to a state which is suitable for ML training sessions.

6.8 Workflow for Model Storage

This subchapter describes the workflow for storing and managing the versions of ML models, as the architecture of the considered model is fixed. The steps are as follows:

1. **Parameter Extraction:** Initially the parameters or weights of the model which encapsulate the learned information are extracted. This step is crucial as these parameters need to be preserved properly for model execution.

2. **RS Encoding:** Once the parameters are extracted, they are immediately encoded using the RS coding scheme. This process distributes the parameter data and adds redundancy, ensuring that in the event of data corruption or loss, recovery can be efficiently achieved.

3. **Storage and Distribution:** The encoded parameters are then stored and distributed across multiple storage locations to further protect against data loss and enhance its accessibility.

4. **Recovery Process:**

   (a) *Damage Check:* Upon retrieval, a check is performed to determine if the parameter data is damaged or corrupted.

   (b) *RS Decoding:* If damage is detected, the RS decoding process is initiated to recover the corrupted parameters. This step is critical for restoring the data to its original state.

   (c) *Model Integration:* After decoding, the recovered parameters are re-integrated with the model. This process involves importing parameters back into the model's architecture and preparing it for further usage or continued training.

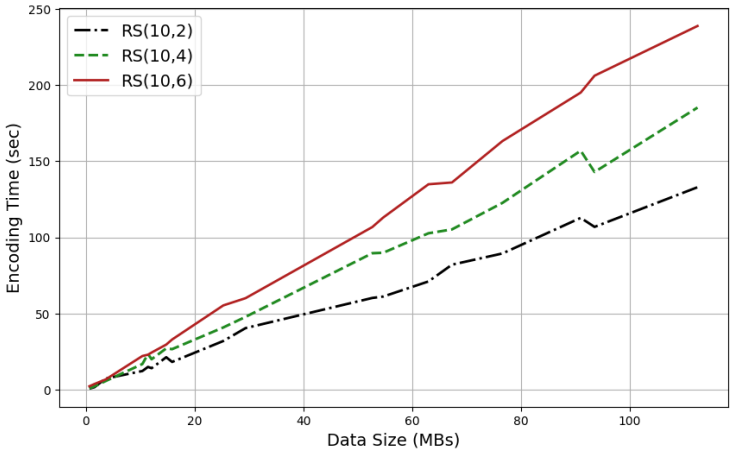

Figure 6.3: Encoding time for different RS configurations.

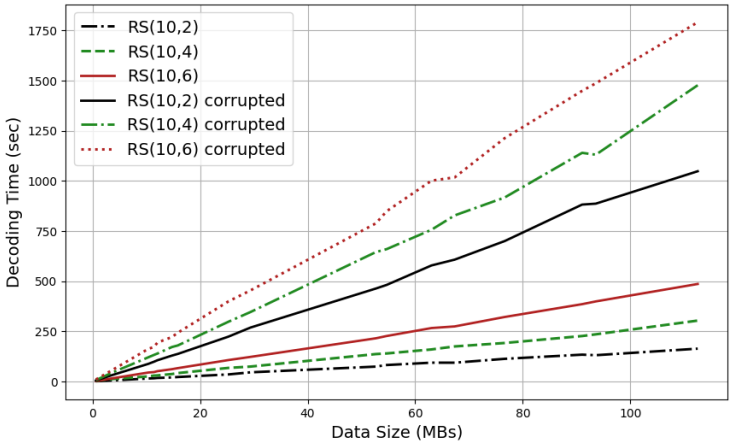

Figure 6.4: Decoding time for different RS configurations.

The Figure 6.3 and 6.4 show the performance of various versions of RS scheme without any compression or optimisation. It depicts the relation between the original data size (in MBs) and the encoding, decoding time (in seconds) for different configurations. I have considered three different configurations for analysis: RS(10,2), RS(10,4), and RS(10,6), where the numbers indicate the code parameters, with the first number representing the total symbols and the second the parity symbols. As the data size increases, both the encoding and decoding time for all configurations increases, which is expected since larger data volumes require more computational effort to encode or decode.

RS(10,2) shows the shortest encoding and decoding times across all data sizes. This configuration has the least number of parity symbols, resulting in less redundancy and hence faster encoding and decoding. On the other hand, RS(10,6) has the longest encoding and decoding times among the configurations suggesting the impact of parity symbols on additional computational overhead in the system. The second part of Figure 6.3 also shows the decoding time of scenarios where one data fragment is corrupted. It can be seen that for all configurations, the decoding times increase significantly when a fragment is corrupted.

6.9 Experimentation and Results

In order to evaluate the proposed approach, I have performed our experiment on an Intel Xeon CPU@2.30 GHz, supported by 13 GBs of RAM, providing a suitable platform for performing data processing tasks. I have used the Multitype Cloud Storage Dataset (MCSD-100) dataset for evaluating the performance of storage systems [149]. It has been specifically designed to assess the system's erasure coding policy and its ability to effectively handle data fragmentation. I utilised MCSD-100 due to its comprehensive capability in handling and testing storage aspects of the data, making it highly relevant for practical assessments.

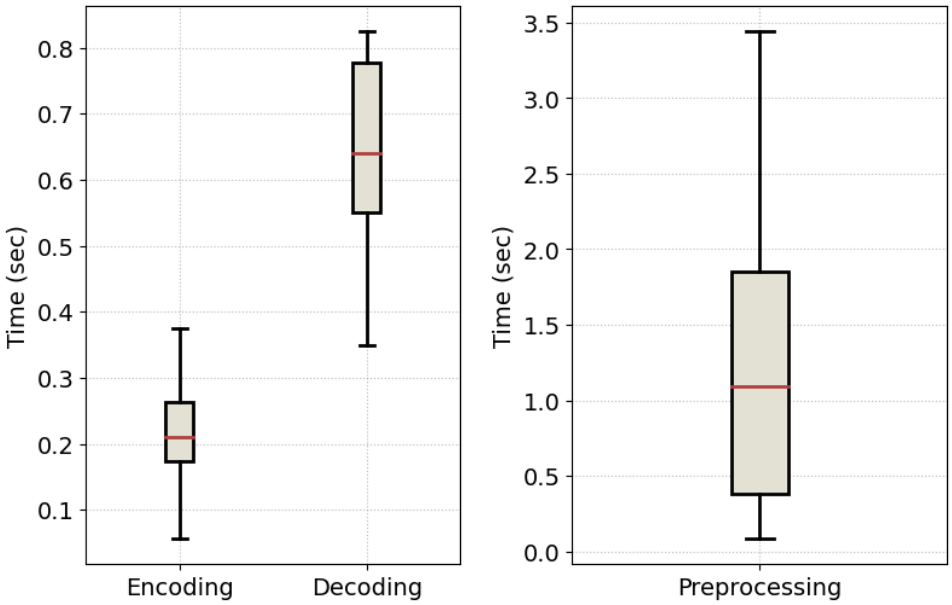

Figure 6.5: Encoding, Decoding, and Preprocessing time.

The box plots provided in Figure 6.5 represent the distribution of preprocessing, encoding, and decoding times for our proposed data storage approach. I observed that the median encoding time for the approach is lower than that for decoding, suggesting that the proposed approach is optimised for faster encoding. Moreover, the smaller inter-quartile range compared to preprocessing indicates a more consistent performance during the encoding process. The results show that our approach showed an average improvement of 96.79%, 97.67%, and 98.72% over the standard RS(10,2), RS(10,4), and RS(10,6) schemes respectively. I also noticed that the execution time for decoding is higher in comparison to encoding times, This is typical in systems where more computational effort is required to reconstruct data from encoded formats. The decoding times for our approach show 92.03%, 96.08%, and 97.38% improvements over standard RS approaches.

Figure 6.6 shows how preprocessing time and compressed data size vary with increasing original data size in a data processing pipeline. The preprocessing time demonstrates a rising trend as the size of the original data increases. It suggests that larger datasets require more time to pre-process, likely due to the increased computational demands of handling

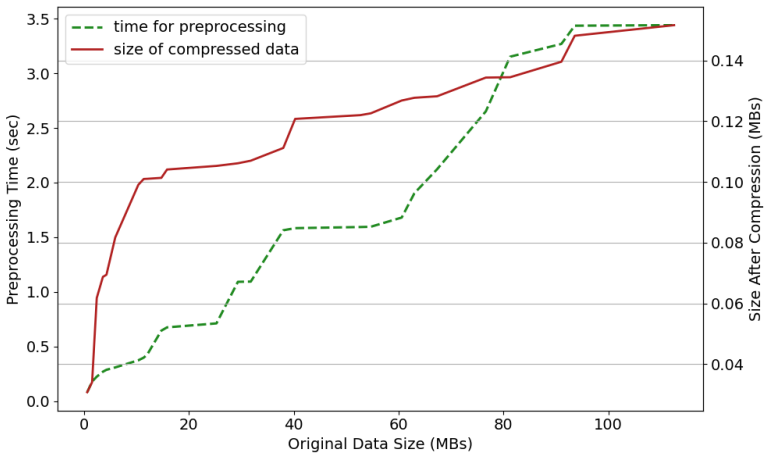

Figure 6.6: Effect on preprocessing time and data size after applying our data storage mechanism.

larger volumes of data. The time required for preprocessing is not exactly linear, showing some ups and downs that may indicate variations in the complexity or type of data being processed. Due to the careful reduction in size during the preprocessing phase, I have also achieved an average reduction of 98.44% size over the standard Brotli compression technique applied on our dataset. On the other side, the compressed data size also shows a correlation with the original data size. As the original data size increases, the compressed size also increases but remains significantly reduced compared to the original, highlighting the effectiveness of the compression technique utilised in the system. This relation describes the capacity of compression method to maintain a relatively constant reduction ratio regardless of the original data size, which is crucial for applications requiring efficient data storage and processing.

Another experimentation is performed to understand the encoding and decoding times for VGG16 model files using various configurations of RS coding scheme. For RS(10, 2), the encoding time is 526.31 seconds and the decoding time is 659.50 seconds, offering the fastest processing among the schemes. In contrast, RS(6, 3) shows longer times with 790.79 seconds for encoding and 1226.20 seconds for decoding, suggesting more complex error correction due to higher redundancy. The RS(14, 4) setup also reflects longer processing times – 769.78 seconds for encoding and 1144.71 seconds for decoding – due to a greater total number of symbols, enhancing error protection. The most data-intensive configuration, RS(10, 6), records the longest times of 1044.19 seconds for encoding and 2133.71 seconds for decoding, indicating the most extensive error correction phase among all the configurations tested.

6.10 Summary

The increasing demand for digital services in the IoT has resulted in a continuous surge of application usage, leading to a huge number of users and connected devices. This growth intensifies the challenges associated with managing and storing large data

volumes, presenting new challenges in data handling. This chapter presents an approach for managing the substantial amount of data generated by IoT devices within ML and edge-cloud environments. Data management in ML-based IoT applications is a crucial factor as it directly affects the performance of deployed applications. Resilience in managing stored data and ML model provides a promising direction for ensuring higher availability of the information. Our work shows that utilising data standardisation, compression, and error correction techniques can provide significant benefits for handling applications deployed on serverless edge-cloud infrastructures. A promising future direction for this work is to design a fault tolerant framework that can handle the failure at data, task, and resource nodes in the IoT infrastructures.

Chapter 7

Conclusion

In this final chapter of the thesis, I summarise the various aspects of edge-cloud frameworks and describe the research questions considered regarding the performance of intelligent IoT applications in the edge-cloud continuum. As the rapid increase of IoT devices continues, ensuring the efficient operation and effective management of tasks, functions, and data is a highly crucial aspect.

7.1 Summary

The research highlights the shortcomings of conventional architectures in managing distributed, heterogeneous IoT deployments across edge and cloud platforms. I first presented an approach that optimally allocates tasks on edge-cloud resource nodes when the network connectivity is limited and the connection is unreliable. A deadline-based heuristic approximation algorithm is proposed to decide where to allocate the task. The algorithm assigns the tasks to the remote Field Side Unit (FSU) or executes them locally based on the quality of connection available at that time. Moreover, as these devices generate and process large volumes of data, effective distribution of load and resource allocation become critical for avoiding any degradation in performance. To address this, I introduced a security-aware load distribution mechanism that ensures fair distribution of tasks based on factors such as failure rate, resource utilisation, completion time, waiting time, and execution overhead. I designed a two-function heuristic pipeline that initially selects the best k nodes (out of all available nodes) and then selects the best two nodes for offloading the tasks. To ensure confidentiality, integrity, and availability, I have utilised encryption, HMAC, and replication mechanisms respectively for all the restricted tasks that were allocated on shared public resources. I also proposed another approach to optimise the ML-based operations that can enhance the execution and deployment of IoT tasks on edge and cloud platforms. This approach uses a genetic algorithm based selection method for optimally selecting the layers during the model training process. To improve the inference framework, I utilise a heatmap visualisation technique that prunes the irrelevant features and only retains the features that directly affect the decision making process. Finally, a data management approach is designed to address the storage, access, and retrieval of data in IoT-based systems in case any failure or data loss occurs. A Reed-Solomen (RS) based encoding method is used to store the data on distributed nodes and the Brotli compression technique is used to reduce the size of data that is stored on the nodes.

7.2 Future Directions

There are numerous opportunities for future research in the deployment of IoT applications within edge-cloud environments. Considering the findings of this thesis, there is a scope to enhance the effectiveness, flexibility, and robustness of our proposed strategies. This subchapter outlines some potential directions for future research with an aim to further improve the performance of frameworks and enhance their efficiency and reliability. These future directions are as follows:

- **Enhancing ML Classification Results:** Future research should focus on enhancing the classification results of our current approach by incorporating optimisation techniques such as multi-objective genetic algorithms and particle swarm optimisation. These methods can be used to fine-tune the parameters and structure of our models, aiming to identify optimal configurations that significantly improve accuracy and efficiency.

- **Dynamic Selection of Models:** I aim to develop a real-time dynamic approach that intelligently selects the most appropriate algorithm for inference from a pool of available options. This selection process can be guided by historical system performance data and predefined user thresholds for making that decision. Moreover, the decision for selection of an algorithm can also be made considering other performance critical factors such as accuracy, task deadline deadline, space available etc.

- **Creation of Local Dataset:** A promising direction for extending this thesis is the development and integration of local datasets. This would strengthen the applicability of our frameworks in diverse real-world scenarios but also enhance its relevance to specific societal needs. By creating datasets that reflect local conditions and challenges, I can refine our solutions to be more precise and ensure that they are robust across various environments.

- **Utilising Reinforcement Learning:** Further research can investigate the application of reinforcement learning, temporal difference learning, and the Markov decision process as optimisation techniques within our system. By integrating these methodologies, I can expect significant enhancements in system adaptability and performance. Additionally, applying reinforcement learning for risk management can enable improved assessment and response strategies, further enhancing our system's capability to handle and mitigate risks dynamically.

- **Real-time Selection of Platform:** I also plan to design an infrastructure that utilises the strengths of both Parsl and OpenWhisk platforms. This approach aims to determine specific scenarios where one framework may outperform the other, thereby selecting that for executing specific federated learning applications. It can ensure that the cost for performing execution in the infrastructure can be minimised and

also provide failure resistance in case one of the platform's performance decreases with time.

- **Self Healing Mechanism for Handling Failures:** I aim to develop a fault-tolerant framework specifically designed for IoT-based infrastructures, capable of managing failures at the data, task, and resource node levels. This framework can be enhanced further by creating a self-healing infrastructure designed to autonomously detect and resolve those failures. The implementation of this robust system will significantly increase resilience, ensuring continuous operation and reliability across diverse IoT applications.

- **Reducing Data Footprint without Compromising Performance:**: Another direction is to explore the integration of advanced data compression algorithms to minimise the data footprint in IoT systems without compromising data quality. I aim to investigate adaptive compression techniques that dynamically adjust according to the type of data and prevailing network conditions. This development is expected to optimise data handling efficiencies, thereby enhancing the performance and scalability of IoT applications in variable network environments.

- **Sustainable Infrastructure for Managing IoT Tasks:** Design a sustainable infrastructure that not only improves the performance efficiency but is also optimised for resource utilisation. This will include improvements in power consumption, resource usage, communication, latency, and execution cost. By focusing on these critical factors, I intend to design an infrastructure that enhances operational effectiveness while minimising environmental and economic impacts, ensuring a balanced approach for function execution in IoT systems.

References

[1] Karen Rose, Scott Eldridge, and Lyman Chapin. The internet of things: An overview. *The internet society (ISOC)*, 80(15):1–53, 2015.

[2] Ting Li, Wei Liu, Tian Wang, Zhao Ming, Xiong Li, and Ming Ma. Trust data collections via vehicles joint with unmanned aerial vehicles in the smart internet of things. *Transactions on Emerging Telecommunications Technologies*, 33(5):e3956, 2022.

[3] Xiaoming Li, Hao Liu, Weixi Wang, Ye Zheng, Haibin Lv, and Zhihan Lv. Big data analysis of the internet of things in the digital twins of smart city based on deep learning. *Future Generation Computer Systems*, 128:167–177, 2022.

[4] Mouzhi Ge, Hind Bangui, and Barbora Buhnova. Big data for internet of things: a survey. *Future generation computer systems*, 87:601–614, 2018.

[5] William Voorsluys, James Broberg, and Rajkumar Buyya. Introduction to cloud computing. *Cloud computing: Principles and paradigms*, pages 1–41, 2011.

[6] Yi Wei and M Brian Blake. Service-oriented computing and cloud computing: Challenges and opportunities. *IEEE Internet Computing*, 14(6):72–75, 2010.

[7] Weisong Shi and Schahram Dustdar. The promise of edge computing. *Computer*, 49(5):78–81, 2016.

[8] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[9] Luciano Baresi, Danilo Filgueira Mendonça, Martin Garriga, Sam Guinea, and Giovanni Quattrocchi. A unified model for the mobile-edge-cloud continuum. *ACM Transactions on Internet Technology (TOIT)*, 19(2):1–21, 2019.

[10] Amin Banitalebi-Dehkordi, Naveen Vedula, Jian Pei, Fei Xia, Lanjun Wang, and Yong Zhang. Auto-split: A general framework of collaborative edge-cloud ai. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2543–2553, 2021.

[11] Fan Liang, Wei Yu, Xing Liu, David Griffith, and Nada Golmie. Toward edge-based deep learning in industrial internet of things. *IEEE Internet of Things Journal*, 7 (5):4329–4341, 2020.

[12] Yangguang Cui, Kun Cao, Junlong Zhou, and Tongquan Wei. Optimizing training efficiency and cost of hierarchical federated learning in heterogeneous mobile-edge

cloud computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[13] Morghan Hartmann, Umair Sajid Hashmi, and Ali Imran. Edge computing in smart health care systems: Review, challenges, and research directions. *Transactions on Emerging Telecommunications Technologies*, 33(3):e3710, 2022.

[14] Latif U Khan, Ibrar Yaqoob, Nguyen H Tran, SM Ahsan Kazmi, Tri Nguyen Dang, and Choong Seon Hong. Edge-computing-enabled smart cities: A comprehensive survey. *IEEE Internet of Things Journal*, 7(10):10200–10232, 2020.

[15] Gabriele Penzotti, Michele Amoretti, and Stefano Caselli. Enabling precision irrigation through a hierarchical edge-to-cloud system. In *International Conference on Advanced Information Networking and Applications*, pages 277–286. Springer, 2024.

[16] Pasquale Pace, Gianluca Aloi, Raffaele Gravina, Giuseppe Caliciuri, Giancarlo Fortino, and Antonio Liotta. An edge-based architecture to support efficient applications for healthcare industry 4.0. *IEEE Transactions on Industrial Informatics*, 15(1):481–489, 2018.

[17] Siyuan Liang, Hao Wu, Li Zhen, Qiaozhi Hua, Sahil Garg, Georges Kaddoum, Mohammad Mehedi Hassan, and Keping Yu. Edge yolo: Real-time intelligent object detection system based on edge-cloud cooperation in autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(12):25345–25360, 2022.

[18] Osama Almurshed, Omer Rana, Yinhao Li, Rajiv Ranjan, Devki Nandan Jha, Pankesh Patel, Prem Prakash Jayaraman, and Schahram Dustdar. A fault tolerant workflow composition and deployment automation IoT framework in a multi cloud edge environment. *IEEE Internet Computing*, 2021.

[19] Xin Wu, Lan You, Ruofan Wu, Qi Zhang, and Kaixin Liang. Management and control of load clusters for ancillary services using internet of electric loads based on cloud–edge–end distributed computing. *IEEE Internet of Things Journal*, 9(19): 18267–18279, 2022.

[20] Panos Patros, Melanie Ooi, Victoria Huang, Michael Mayo, Chris Anderson, Stephen Burroughs, Matt Baughman, Osama Almurshed, Omer Rana, Ryan Chard, et al. Rural AI: Serverless-powered federated learning for remote applications. *IEEE Internet Computing*, 2022.

[21] Shaojie Zhuo, Hongyu Chen, Ramchalam Kinattinkara Ramakrishnan, Tommy Chen, Chen Feng, Yicheng Lin, Parker Zhang, and Liang Shen. An empirical study of low precision quantization for tinyml. *arXiv preprint arXiv:2203.05492*, 2022.

[22] Fred Hohman, Mary Beth Kery, Donghao Ren, and Dominik Moritz. Model compression in practice: Lessons learned from practitioners creating on-device

machine learning experiences. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–18, 2024.

[23] Yibo Yang, Stephan Mandt, Lucas Theis, et al. An introduction to neural data compression. *Foundations and Trends® in Computer Graphics and Vision*, 15(2): 113–200, 2023.

[24] Jianhua He, Jian Wei, Kai Chen, Zuoyin Tang, Yi Zhou, and Yan Zhang. Multitier fog computing with large-scale IoT data analytics for smart cities. *IEEE Internet of Things Journal*, 5(2):677–686, 2017.

[25] Siliang Lu, Jinfeng Lu, Kang An, Xiaoxian Wang, and Qingbo He. Edge computing on IoT for machine signal processing and fault diagnosis: A review. *IEEE Internet of Things Journal*, 2023.

[26] Xiaokang Wang, Lei Ren, Ruixue Yuan, Laurence T Yang, and M Jamal Deen. Qtt-dlstm: A cloud-edge-aided distributed lstm for cyber-physical-social big data. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[27] Qiang Duan, Shangguang Wang, and Nirwan Ansari. Convergence of networking and cloud/edge computing: Status, challenges, and opportunities. *IEEE Network*, 34(6):148–155, 2020.

[28] Tian Wang, Yuzhu Liang, Xuewei Shen, Xi Zheng, Adnan Mahmood, and Quan Z Sheng. Edge Computing and Sensor-Cloud: Overview, Solutions, and Directions. *ACM Computing Surveys*, 2023.

[29] Chunlin Li, Hezhi Sun, Hengliang Tang, and Youlong Luo. Adaptive resource allocation based on the billing granularity in edge-cloud architecture. *Computer Communications*, 145:29–42, 2019.

[30] Mohamed A Kamel, Xiang Yu, and Youmin Zhang. Formation control and coordination of multiple unmanned ground vehicles in normal and faulty situations: A review. *Annual reviews in control*, 49:128–144, 2020.

[31] Osama Almurshed, Omer Rana, and Kyle Chard. Greedy nominator heuristic: Virtual function placement on fog resources. *Concurrency and Computation: Practice and Experience*, 34(6):e6765, 2022.

[32] Alex Olsen, Dmitry A Konovalov, Bronson Philippa, Peter Ridd, Jake C Wood, Jamie Johns, Wesley Banks, Benjamin Girgenti, Owen Kenny, James Whinney, et al. Deepweeds: A multiclass weed species image dataset for deep learning. *Scientific reports*, 9(1):1–12, 2019.

[33] Osama Almurshed, Panos Patros, Victoria Huang, Michael Mayo, Melanie Ooi, Ryan Chard, Kyle Chard, Omer Rana, Harshaan Nagra, Matt Baughman, et al. Adaptive

edge-cloud environments for rural AI. In *2022 IEEE International Conference on Services Computing (SCC)*, pages 74–83. IEEE, 2022.

[34] Ramyad Hadidi, Jiashen Cao, Michael S Ryoo, and Hyesoon Kim. Toward collaborative inferencing of deep neural networks on internet-of-things devices. *IEEE Internet of Things Journal*, 7(6):4950–4960, 2020.

[35] Xing Chen, Jianshan Zhang, Bing Lin, Zheyi Chen, Katinka Wolter, and Geyong Min. Energy-efficient offloading for DNN-based smart iot systems in cloud-edge environments. *IEEE Transactions on Parallel and Distributed Systems*, 33(3): 683–697, 2021.

[36] Saeed Javanmardi, Mohammad Shojafar, Valerio Persico, and Antonio Pescapè. FPFTS: a joint fuzzy particle swarm optimization mobility-aware approach to fog task scheduling algorithm for internet of things devices. *Software: Practice and Experience*, 51(12):2519–2539, 2021.

[37] Shihong Hu and Guanghui Li. Dynamic request scheduling optimization in mobile edge computing for iot applications. *IEEE Internet of Things Journal*, 7(2): 1426–1437, 2019.

[38] Firdose Saeik, Marios Avgeris, Dimitrios Spatharakis, Nina Santi, Dimitrios Dechouniotis, John Violos, Aris Leivadeas, Nikolaos Athanasopoulos, Nathalie Mitton, and Symeon Papavassiliou. Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions. *Computer Networks*, 195:108177, 2021.

[39] Michael Putzier, T Khakzad, M Dreischarf, Sylvia Thun, F Trautwein, and N Taheri. Implementation of cloud computing in the german healthcare system. *NPJ Digital Medicine*, 7(1):12, 2024.

[40] Su Hu and Yinhao Xiao. Design of cloud computing task offloading algorithm based on dynamic multi-objective evolution. *Future Generation Computer Systems*, 122: 144–148, 2021.

[41] Yuxuan Sun, Xueying Guo, Sheng Zhou, Zhiyuan Jiang, Xin Liu, and Zhisheng Niu. Learning-based task offloading for vehicular cloud computing systems. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2018.

[42] Min Chen and Yixue Hao. Task offloading for mobile edge computing in software defined ultra-dense network. *IEEE Journal on Selected Areas in Communications*, 36(3):587–597, 2018.

[43] Ragib Hasan, Mahmud Hossain, and Rasib Khan. Aura: An incentive-driven ad-hoc iot cloud framework for proximal mobile computation offloading. *Future Generation Computer Systems*, 86:821–835, 2018.

[44] Ben Langmead and Abhinav Nellore. Cloud computing for genomic data analysis and collaboration. *Nature Reviews Genetics*, 19(4):208–219, 2018.

[45] Belen Bermejo and Carlos Juiz. Improving cloud/edge sustainability through artificial intelligence: A systematic review. *Journal of Parallel and Distributed Computing*, 176:41–54, 2023.

[46] Abdenacer Naouri, Hangxing Wu, Nabil Abdelkader Nouri, Sahraoui Dhelim, and Huansheng Ning. A novel framework for mobile-edge computing by optimizing task offloading. *IEEE Internet of Things Journal*, 8(16):13065–13076, 2021.

[47] Changchun Long, Yang Cao, Tao Jiang, and Qian Zhang. Edge computing framework for cooperative video processing in multimedia iot systems. *IEEE Transactions on Multimedia*, 20(5):1126–1139, 2017.

[48] Tiankui Zhang, Yu Xu, Jonathan Loo, Dingcheng Yang, and Lin Xiao. Joint computation and communication design for uav-assisted mobile edge computing in iot. *IEEE Transactions on Industrial Informatics*, 16(8):5505–5516, 2019.

[49] Jielin Jiang, Zheng Li, Yuan Tian, and Najla Al-Nabhan. A review of techniques and methods for iot applications in collaborative cloud-fog environment. *Security and Communication Networks*, 2020(1):8849181, 2020.

[50] Caihong Kai, Hao Zhou, Yibo Yi, and Wei Huang. Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability. *IEEE Transactions on Cognitive Communications and Networking*, 7 (2):624–634, 2020.

[51] Jinke Ren, Guanding Yu, Yinghui He, and Geoffrey Ye Li. Collaborative cloud and edge computing for latency minimization. *IEEE Transactions on Vehicular Technology*, 68(5):5031–5044, 2019.

[52] Yixue Hao, Yingying Jiang, Tao Chen, Donggang Cao, and Min Chen. itaskoffloading: intelligent task offloading for a cloud-edge collaborative system. *IEEE Network*, 33(5):82–88, 2019.

[53] Xu Chen, Qian Shi, Lei Yang, and Jie Xu. Thriftyedge: Resource-efficient edge computing for intelligent iot applications. *IEEE network*, 32(1):61–65, 2018.

[54] Robert Sparrow and Mark Howard. Robots in agriculture: prospects, impacts, ethics, and policy. *precision agriculture*, 22:818–833, 2021.

[55] G Pantelimon, Kemal Tepe, Rupp Carriveau, and Sabbir Ahmed. Survey of multi-agent communication strategies for information exchange and mission control of drone deployments. *Journal of Intelligent & Robotic Systems*, 95:779–788, 2019.

[56] Jesús Conesa-Muñoz, João Valente, Jaime Del Cerro, Antonio Barrientos, and Angela Ribeiro. A multi-robot sense-act approach to lead to a proper acting in environmental incidents. *Sensors*, 16(8):1269, 2016.

[57] Angelos Dimakos, Daniel Woodhall, and Seemal Asif. A study on centralised and decentralised swarm robotics architecture for part delivery system. *arXiv preprint arXiv:2403.07635*, 2024.

[58] Albert Rego, Pedro Luis González Ramírez, Jose M Jimenez, and Jaime Lloret. Artificial intelligent system for multimedia services in smart home environments. *Cluster Computing*, 25(3):2085–2105, 2022.

[59] Panos Patros, Melanie Ooi, Victoria Huang, Michael Mayo, Chris Anderson, Stephen Burroughs, Matt Baughman, Osama Almurshed, Omer Rana, Ryan Chard, et al. Rural-AI: Serverless-powered federated learning for remote applications. *IEEE Internet Computing*, 27(2):28–34, 2022.

[60] Osama Almurshed, Panos Patros, Victoria Huang, Michael Mayo, Melanie Ooi, Ryan Chard, Kyle Chard, Omer Rana, Harshaan Nagra, Matt Baughman, et al. Adaptive edge-cloud environments for rural AI. In *IEEE International Conference on Services Computing (SCC)*, pages 74–83, 2022.

[61] Mohammed Islam Naas, Laurent Lemarchand, Jalil Boukhobza, and Philippe Raipin. A graph partitioning-based heuristic for runtime IoT data placement strategies in a fog infrastructure. In *Proceedings of the 33rd annual ACM symposium on applied computing*, pages 767–774, 2018.

[62] Sudip Misra and Niloy Saha. Detour: Dynamic task offloading in software-defined fog for IoT applications. *IEEE Journal on Selected Areas in Communications*, 37 (5):1159–1166, 2019.

[63] Jessica Oueis, Emilio Calvanese Strinati, and Sergio Barbarossa. The fog balancing: Load distribution for small cell cloud computing. In *2015 IEEE 81st vehicular technology conference (VTC spring)*, pages 1–6. IEEE, 2015.

[64] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. Presto: Edge-based load balancing for fast datacenter networks. *ACM SIGCOMM Computer Communication Review*, 45(4):465–478, 2015.

[65] Kai Peng, Hualong Huang, Wenjie Pan, and Jiabin Wang. Joint optimisation for time consumption and energy consumption of multi-application and load balancing of cloudlets in mobile edge computing. *IET Cyber-Physical Systems: Theory & Applications*, 5(2):196–206, 2020.

[66] Adyson M Maia, Yacine Ghamri-Doudane, Dario Vieira, and Miguel Franklin de Castro. An improved multi-objective genetic algorithm with heuristic

initialization for service placement and load distribution in edge computing. *Computer networks*, 194:108146, 2021.

[67] Herbert G Tanner and Dimitrios K Christodoulakis. Decentralized cooperative control of heterogeneous vehicle groups. *Robotics and autonomous systems*, 55(11): 811–823, 2007.

[68] Brett Koonce. ResNet 50. *Convolutional Neural Networks with Swift for Tensorflow: Image Recognition and Dataset Categorization*, pages 63–72, 2021.

[69] Alex Olsen, Dmitry A Konovalov, Bronson Philippa, Peter Ridd, Jake C Wood, Jamie Johns, Wesley Banks, Benjamin Girgenti, Owen Kenny, James Whinney, et al. DeepWeeds: A multiclass weed species image dataset for deep learning. *Scientific reports*, 9(1):2058, 2019.

[70] Kaustabha Ray and Ansuman Banerjee. Prioritized fault recovery strategies for multi-access edge computing using probabilistic model checking. *IEEE Transactions on Dependable and Secure Computing*, 20(1):797–812, 2023. doi: 10.1109/TDSC.2022.3143877.

[71] Anusha Vangala, Ashok Kumar Das, Vinay Chamola, Valery Korotaev, and Joel JPC Rodrigues. Security in IoT-enabled smart agriculture: Architecture, security solutions and challenges. *Cluster Computing*, 26(2):879–902, 2023.

[72] Angelita Rettore de Araujo Zanella, Eduardo da Silva, and Luiz Carlos Pessoa Albini. Security challenges to smart agriculture: Current state, key issues, and future directions. *Array*, 8:100048, 2020.

[73] Leanne Wiseman, Jay Sanderson, Airong Zhang, and Emma Jakku. Farmers and their data: An examination of farmers' reluctance to share their data through the lens of the laws impacting smart farming. *NJAS-Wageningen Journal of Life Sciences*, 90:100301, 2019.

[74] Jitendra Kumar Samriya, Chinmay Chakraborty, Aditi Sharma, Mohit Kumar, et al. Adversarial ml-based secured cloud architecture for consumer internet of things of smart healthcare. *IEEE Transactions on Consumer Electronics*, 2023.

[75] Zecheng He, Tianwei Zhang, and Ruby B Lee. Attacking and protecting data privacy in edge–cloud collaborative inference systems. *IEEE Internet of Things Journal*, 8 (12):9706–9716, 2020.

[76] Tian Wang, Yaxin Mei, Weijia Jia, Xi Zheng, Guojun Wang, and Mande Xie. Edge-based differential privacy computing for sensor–cloud systems. *Journal of Parallel and Distributed computing*, 136:75–85, 2020.

[77] Dingde Jiang, Peng Zhang, Zhihan Lv, and Houbing Song. Energy-efficient multi-constraint routing algorithm with load balancing for smart city applications. *IEEE Internet of Things Journal*, 3(6):1437–1447, 2016.

[78] Roberto Beraldi, Claudia Canali, Riccardo Lancellotti, and Gabriele Proietti Mattia. Distributed load balancing for heterogeneous fog computing infrastructures in smart cities. *Pervasive and Mobile Computing*, 67:101221, 2020.

[79] Atakan Aral and Ivona Brandić. Learning spatiotemporal failure dependencies for resilient edge computing services. *IEEE Transactions on Parallel and Distributed Systems*, 32(7):1578–1590, 2021.

[80] Deepak Puthal, Rajiv Ranjan, Ashish Nanda, Priyadarsi Nanda, Prem Prakash Jayaraman, and Albert Y Zomaya. Secure authentication and load balancing of distributed edge datacenters. *Journal of Parallel and Distributed Computing*, 124: 60–69, 2019.

[81] Emna Baccour, Naram Mhaisen, Alaa Awad Abdellatif, Aiman Erbad, Amr Mohamed, Mounir Hamdi, and Mohsen Guizani. Pervasive ai for iot applications: A survey on resource-efficient distributed artificial intelligence. *IEEE Communications Surveys & Tutorials*, 24(4):2366–2418, 2022. doi: 10.1109/COMST.2022.3200740.

[82] Liang Xiao, Xiaoyue Wan, Xiaozhen Lu, Yanyong Zhang, and Di Wu. Iot security techniques based on machine learning: How do iot devices use ai to enhance security? *IEEE Signal Processing Magazine*, 35(5):41–49, 2018. doi: 10.1109/MSP.2018.2825478.

[83] Fanyu Bu and Xin Wang. A smart agriculture iot system based on deep reinforcement learning. *Future Generation Computer Systems*, 99:500–507, 2019.

[84] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

[85] Da Li, Xinbo Chen, Michela Becchi, and Ziliang Zong. Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus. In *2016 IEEE international conferences on big data and cloud computing (BDCloud), social computing and networking (SocialCom), sustainable computing and communications (SustainCom)(BDCloud-SocialCom-SustainCom)*, pages 477–484. IEEE, 2016.

[86] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International conference on machine learning*, pages 10096–10106. PMLR, 2021.

[87] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[88] Yiming Hu, Siyang Sun, Jianquan Li, Xingang Wang, and Qingyi Gu. A novel channel pruning method for deep neural network compression. *arXiv preprint arXiv:1805.11394*, 2018.

[89] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

[90] Zhen Chen, Zhibo Chen, Jianxin Lin, Sen Liu, and Weiping Li. Deep neural network acceleration based on low-rank approximated channel pruning. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(4):1232–1244, 2020.

[91] Zhuoran Song, Bangqi Fu, Feiyang Wu, Zhaoming Jiang, Li Jiang, Naifeng Jing, and Xiaoyao Liang. Drq: dynamic region-based quantization for deep neural network acceleration. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 1010–1021. IEEE, 2020.

[92] Chuhan Wu, Fangzhao Wu, Lingjuan Lyu, Yongfeng Huang, and Xing Xie. Communication-efficient federated learning via knowledge distillation. *Nature communications*, 13(1):2032, 2022.

[93] Eva García-Martín, Crefeda Faviola Rodrigues, Graham Riley, and Håkan Grahn. Estimation of energy consumption in machine learning. *Journal of Parallel and Distributed Computing*, 134:75–88, 2019.

[94] Seul-Ki Yeom, Philipp Seegerer, Sebastian Lapuschkin, Alexander Binder, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Pruning by explaining: A novel criterion for deep neural network pruning. *Pattern Recognition*, 115:107899, 2021.

[95] Yadu N Babuji, Kyle Chard, Aaron Gerow, and Eamon Duede. Cloud kotta: Enabling secure and scalable data analytics in the cloud. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 302–310. IEEE, 2016.

[96] Tianli Zhou and Chao Tian. Fast erasure coding for data storage: A comprehensive study of the acceleration techniques. *ACM Transactions on Storage (TOS)*, 16(1): 1–24, 2020.

[97] Qianqiu Wang, Xiaoping Ye, Xianlu Luo, Lunjie Li, and Hainan Chen. A distributed data storage strategy based on LOPs. *Arabian Journal for Science and Engineering*, 47(8):9767–9779, 2022.

[98] Justice Opara-Martins, Reza Sahandi, and Feng Tian. Critical review of vendor lock-in and its impact on adoption of cloud computing. In *International conference on information society (i-Society 2014)*, pages 92–97. IEEE, 2014.

[99] Ying Song, Qiang Zhang, and Bo Wang. FACHS: Adaptive hybrid storage strategy based on file access characteristics. *IEEE Access*, 11:16855–16862, 2023.

[100] Jad Darrous and Shadi Ibrahim. Understanding the performance of erasure codes in hadoop distributed file system. In *Proceedings of the Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems*, pages 24–32, 2022.

[101] Michael Abebe, Khuzaima Daudjee, Brad Glasbergen, and Yuanfeng Tian. Ec-store: Bridging the gap between storage and latency in distributed erasure coded systems. In *2018 IEEE 38th international conference on distributed computing systems (ICDCS)*, pages 255–266. IEEE, 2018.

[102] Zhinan Cheng, Lu Tang, Qun Huang, and Patrick PC Lee. Enabling low-redundancy proactive fault tolerance for stream machine learning via erasure coding. In *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*, pages 99–108. IEEE, 2021.

[103] Rekha Nachiappan, Rodrigo N Calheiros, Kenan M Matawie, and Bahman Javadi. Optimized proactive recovery in erasure-coded cloud storage systems. *IEEE Access*, 2023.

[104] *The State of Food Security and Nutrition in the World 2022 [Online]. Available: https://www.fao.org/3/cc0639en/online/sofi-2022/ introduction.html.*

[105] Chengsong Hu, J Alex Thomasson, and Muthukumar V Bagavathiannan. A powerful image synthesis and semi-supervised learning pipeline for site-specific weed detection. *Computers and Electronics in Agriculture*, 190:106423, 2021.

[106] Sukhpal Singh Gill. Quantum and blockchain based serverless edge computing: A vision, model, new trends and future directions. *Internet Technology Letters*, page e275, 2021.

[107] Amit Samanta, Flavio Esposito, and Tri Gia Nguyen. Fault-tolerant mechanism for edge-based IoT networks with demand uncertainty. *IEEE Internet of Things Journal*, 8(23):16963–16971, 2021. doi: 10.1109/JIOT.2021.3075681.

[108] Heidi Saxby, Menelaos Gkartzios, and Karen Scott. 'farming on the edge': wellbeing and participation in agri-environmental schemes. *Sociologia Ruralis*, 58(2):392–411, 2018.

[109] N Brisson, C Gary, E Justes, R Roche, B Mary, D Ripoche, D Zimmer, J Sierra, P Bertuzzi, P Burger, F Bussière, Y.M Cabidoche, P Cellier, P Debaeke, J.P Gaudillère, C Hénault, F Maraux, B Seguin, and H Sinoquet. An overview of the crop model stics. *European Journal of Agronomy*, 18(3):309–332, 2003. Modelling Cropping Systems: Science, Software and Applications.

[110] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[111] Yadu Babuji, Anna Woodard, Zhuozhao Li, Daniel S. Katz, Ben Clifford, Rohan Kumar, Lukasz Lacinski, Ryan Chard, Justin Wozniak, Ian Foster, Mike Wilde,

and Kyle Chard. *parsl*: Pervasive parallel programming in python. In *28th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2019.

[112] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[113] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[114] Ashish Kaushal, Osama Almurshed, Areej Alabbas, Nitin Auluck, and Omer Rana. An edge-cloud infrastructure for weed detection in precision agriculture. In *IEEE Intl Conf on Pervasive Intelligence and Computing (PiCom)*, pages 0269–0276, 2023.

[115] Marcelo José Carrer, Hildo Meirelles de Souza Filho, Marcela de Mello Brandao Vinholis, and Carlos Ivan Mozambani. Precision agriculture adoption and technical efficiency: An analysis of sugarcane farms in brazil. *Technological Forecasting and Social Change*, 177:121510, 2022.

[116] Peeyush Kumar, Andrew Nelson, Zerina Kapetanovic, and Ranveer Chandra. Affordable artificial intelligence–augmenting farmer knowledge with ai. *arXiv preprint arXiv:2303.06049*, 2023.

[117] *The State of Food Security and Nutrition in the World 2022 [Online]. Available: https://www.fao.org/3/cc0639en/online/sofi-2022/ introduction.html.*

[118] René Dumont. *Types of Rural Economy: Studies in World Agriculture*, volume 8. Taylor & Francis, 2023.

[119] Hui Yuan and Hong Nie. Edge computing driven sustainable development: A case study on professional farmer cultivation mechanism. *Expert Systems*, 2023.

[120] Andrew D. Balmos, Fabio A. Castiblanco, Aaron J. Neustedter, James V. Krogmeier, and Dennis R. Buckmaster. Isoblue avena: A framework for agricultural edge computing and data sovereignty. *IEEE Micro*, 42(1):78–86, 2022. doi: 10.1109/MM.2021.3134830.

[121] Yogeswaranathan Kalyani and Rem Collier. A systematic survey on the role of cloud, fog, and edge computing combination in smart agriculture. *Sensors*, (17): 5922, 2021.

[122] Ahmad Ali Alzubi and Kalda Galyna. Artificial intelligence and internet of things for sustainable farming and smart agriculture. *IEEE Access*, 2023.

[123] M Trendov, Samuel Varas, Meng Zeng, et al. Digital technologies in agriculture and rural areas: status report. *Digital technologies in agriculture and rural areas: status report.*, 2019.

[124] Sumarga Kumar Sah Tyagi, Amrit Mukherjee, Shiva Raj Pokhrel, and Kamal Kant Hiran. An intelligent and optimal resource allocation approach in sensor networks for smart agri-IoT. *IEEE Sensors Journal*, 21(16):17439–17446, 2020.

[125] Anandarup Mukherjee, Sudip Misra, Anumandala Sukrutha, and Narendra Singh Raghuwanshi. Distributed aerial processing for IoT-based edge UAV swarms in smart farming. *Computer Networks*, 167:107038, 2020.

[126] Othmane Friha, Mohamed Amine Ferrag, Lei Shu, Leandros Maglaras, Kim-Kwang Raymond Choo, and Mehdi Nafaa. FELIDS: Federated learning-based intrusion detection system for agricultural Internet of Things. *Journal of Parallel and Distributed Computing*, 165:17–31, 2022.

[127] Khalid Haseeb, Ikram Ud Din, Ahmad Almogren, and Naveed Islam. An energy efficient and secure iot-based wsn framework: An application to smart agriculture. *Sensors*, 20(7):2081, 2020.

[128] Theyazn HH Aldhyani and Hasan Alkahtani. Cyber Security for Detecting Distributed Denial of Service Attacks in Agriculture 4.0: Deep Learning Model. *Mathematics*, 11(1):233, 2023.

[129] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. pages 248–255, 2009.

[130] Yadu Babuji, Anna Woodard, Zhuozhao Li, Daniel S Katz, Ben Clifford, Rohan Kumar, Lukasz Lacinski, Ryan Chard, Justin M Wozniak, Ian Foster, et al. Parsl: Pervasive parallel programming in python. pages 25–36, 2019.

[131] Areej Alabbas, Ashish Kaushal, Osama Almurshed, Omer Rana, Nitin Auluck, and Charith Perera. Performance analysis of apache openwhisk across the edge-cloud continuum.

[132] Rainer Diesch, Matthias Pfaff, and Helmut Krcmar. A comprehensive model of information security factors for decision-makers. *Computers & Security*, 92:101747, 2020.

[133] Ashish Kaushal, Osama Almurshed, Osama Almoghamis, Areej Alabbas, Nitin Auluck, Bharadwaj Veeravalli, and Omer Rana. Shield: A secure heuristic integrated environment for load distribution in rural-ai. *Future Generation Computer Systems*, 161:286–301, 2024.

[134] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga, Jinshi Huang, Charles Bai,

et al. Sustainable AI: Environmental implications, challenges and opportunities. *Proceedings of Machine Learning and Systems*, 4:795–813, 2022.

[135] Ling Hu and Qiang Ni. Iot-driven automated object detection algorithm for urban surveillance systems in smart cities. *IEEE Internet of Things Journal*, 5(2):747–754, 2017.

[136] Areej Alabbas, Ashish Kaushal, Osama Almurshed, Omer Rana, Nitin Auluck, and Charith Perera. Performance analysis of apache openwhisk across the edge-cloud continuum. In *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*, pages 401–407. IEEE, 2023.

[137] Radu Marculescu, Diana Marculescu, and Umit Ogras. Edge ai: Systems design and ml for iot data analytics. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3565–3566, 2020.

[138] Sudershan Boovaraghavan, Anurag Maravi, Prahaladha Mallela, and Yuvraj Agarwal. Mliot: An end-to-end machine learning system for the internet-of-things. In *Proceedings of the International Conference on Internet-of-Things Design and Implementation*, pages 169–181, 2021.

[139] Pavana Prakash, Jiahao Ding, Rui Chen, Xiaoqi Qin, Minglei Shu, Qimei Cui, Yuanxiong Guo, and Miao Pan. Iot device friendly and communication-efficient federated learning via joint model pruning and quantization. *IEEE Internet of Things Journal*, 9(15):13638–13650, 2022.

[140] Eun Som Jeon, Anirudh Som, Ankita Shukla, Kristina Hasanaj, Matthew P Buman, and Pavan Turaga. Role of data augmentation strategies in knowledge distillation for wearable sensor data. *IEEE internet of things journal*, 9(14):12848–12860, 2021.

[141] Joseph Azar, Abdallah Makhoul, Mahmoud Barhamgi, and Raphaël Couturier. An energy efficient iot data compression approach for edge machine learning. *Future Generation Computer Systems*, 96:168–175, 2019.

[142] Huirong Ma, Zhi Zhou, Xiaoxi Zhang, and Xu Chen. Towards carbon-neutral edge computing: Greening edge ai by harnessing spot and future carbon markets. *IEEE Internet of Things Journal*, 2023.

[143] Sha Zhu, Kaoru Ota, and Mianxiong Dong. Energy-efficient artificial intelligence of things with intelligent edge. *IEEE Internet of Things Journal*, 9(10):7525–7532, 2022.

[144] Sha Zhu, Kaoru Ota, and Mianxiong Dong. Green ai for iiot: Energy efficient intelligent edge computing for industrial internet of things. *IEEE Transactions on Green Communications and Networking*, 6(1):79–88, 2021.

[145] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. pages 618–626, 2017.

[146] Osama Almurshed, Souham Meshoul, Asmail Muftah, Ashish Kumar Kaushal, Osama Almoghamis, Ioan Petri, Nitin Auluck, and Omer Rana. A framework for performance optimization of internet of things applications. In *European Conference on Parallel Processing*, pages 165–176. Springer, 2023.

[147] Luiz Bittencourt, Roger Immich, Rizos Sakellariou, Nelson Fonseca, Edmundo Madeira, Marilia Curado, Leandro Villas, Luiz DaSilva, Craig Lee, and Omer Rana. The internet of things, fog and cloud continuum: Integration and challenges. *Internet of Things*, 3:134–155, 2018.

[148] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[149] Md Sadiqul Islam Sakif, Rezuana Imtiaz Upoma, and Jannatun Noor. Towards benchmarking erasure coding schemes in object storage system. *Available at SSRN 4713329*.