

This is an Open Access document downloaded from ORCA, Cardiff University's institutional repository:<https://orca.cardiff.ac.uk/id/eprint/170821/>

This is the author's version of a work that was submitted to / accepted for publication.

Citation for final published version:

Kaushal, Ashish, Almurshed, Osama, Almoghamis, Osama, Alabbas, Areej, Auluck, Nitin, Veeravalli, Bharadwaj and Rana, Omer 2024. SHIELD: A secure heuristic integrated environment for load distribution in rural-AI. Future Generation Computer Systems: The International Journal of eScience 161 , pp. 286-301. 10.1016/j.future.2024.07.026

Publishers page: <https://doi.org/10.1016/j.future.2024.07.026>

Please note:

Changes made as a result of publishing processes such as copy-editing, formatting and page numbers may not be reflected in this version. For the definitive version of this publication, please refer to the published source. You are advised to consult the publisher's version if you wish to cite this paper.

This version is being made available in accordance with publisher policies. See <http://orca.cf.ac.uk/policies.html> for usage policies. Copyright and moral rights for publications made available in ORCA are retained by the copyright holders.



SHIELD: A Secure Heuristic Integrated Environment for Load Distribution in Rural-AI

Ashish Kaushal^{a,1}, Osama Almurshed^{b,d,1}, Osama Almoghamis^c, Areej Alabbas^b, Nitin Auluck^a, Bharadwaj Veeravalli^e, Omer Rana^b

^aIndian Institute of Technology Ropar, India

^bCardiff University, United Kingdom

^cKing Saud University, Saudi Arabia

^dPrince Sattam Bin Abdulaziz University, Saudi Arabia

^eNational University of Singapore, Singapore

Abstract

The increasing adoption of edge computing in rural areas is leading to a substantial rise in data generation, necessitating the need for development of advanced load balancing algorithms. This is particularly important in applications that utilise existing, though limited, computational and data communication infrastructures. Furthermore, rural communities have growing concerns regarding the privacy, security, and ownership of the data produced within their agricultural fields. Load distribution in rural edge devices can enhance agricultural practices by improving resource usage, decision-making, and addressing network connectivity challenges. Managing resource utilisation in this way also improves economic investments made in managing and deploying edge devices in rural environments. In this work, we propose SHIELD, a security-aware load balancing framework, primarily designed for edge-based systems in rural areas. For handling environments with limited connectivity, SHIELD efficiently manages tasks and computational resources by categorising them into restricted, public and private, shared respectively. It also allocates tasks considering key performance factors such as completion time, resource utilisation, failure rate, and security. The framework is evaluated on a weed detection scenario in precision agriculture, using three federated learning (FL) variants (local model training, global model aggregation, and model prediction) with the ResNet-50 model trained on the DeepWeeds image classification dataset. The proposed framework also integrates encryption and task replication techniques for data confidentiality, integrity, and availability. Experimental results show that SHIELD demonstrates an average of 23% (using Parsl), 29% (using OpenWhisk) improvement in failure rate and 18s (Parsl), 13s (OpenWhisk) average improvement in makespan compared to other task allocation approaches, such as secure variants of random, round robin, and least loaded.

Keywords: Artificial Intelligence, Distributed Systems, Edge Computing, Internet of Things, Load Balancing, Precision Agriculture

1. Introduction

The use of technology to automate agriculture processes has revolutionised farming, leading to improved utilisation of resources and decision-making [1, 2]. 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 [3].

While technology offers several benefits in agriculture, rural communities have limited access to data communications and computational infrastructure [4], 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 [5]. 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 [6, 7] 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 [8]. This allows farmers to take immediate action on the fields, directly enhancing agricultural productivity and sustainability.

Unreliable connectivity and environmental variations are

¹Ashish Kaushal and Osama Almurshed contributed equally in this work.

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 [9]. Load balancing [10, 11] 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.

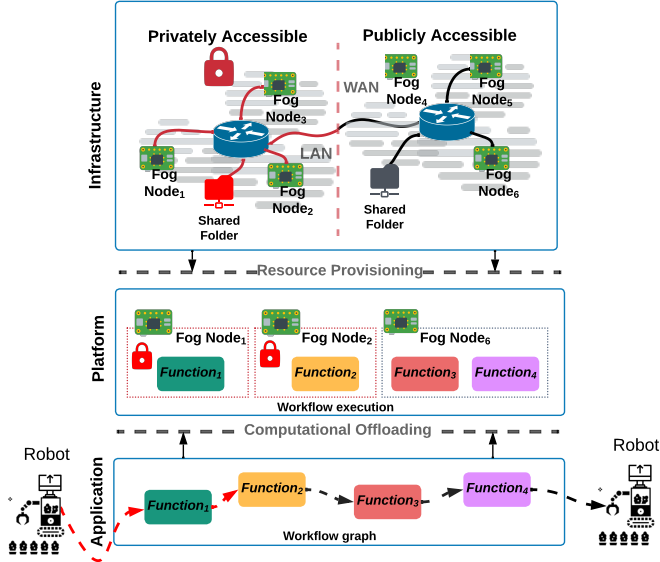


Figure 1. Architecture for the proposed load balancing framework. Private nodes (red links) available in high-security zone handle sensitive tasks; shared nodes handle less restricted tasks on the public network.

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 [12, 13, 14]. 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.

We 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. We divided the tasks into independent categories: restricted and public. These tasks are allocated to two different resources: private and shared, based on secu-

rity 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 1 of this section. The primary contributions of this work 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 (RPi) 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.

The remaining paper is organised as follows: Section 2 describes related work and provides an overview of edge-based load balancing frameworks. Section 3 outlines the key components used in SHIELD. A mathematical formulation of the problem and the associated objective functions are described in section 4. Section 5 includes a description of the system architecture and the proposed load balancing strategy. Section 6 compares the performance of Parsl and OW platforms for an FL application on a Raspberry Pi. Experiment setup for evaluation and generated results are described in sections 7 and 8. Section 9 provides the conclusion for our current work and proposes a direction for future research.

2. Related Work

This section describes research efforts toward designing agricultural applications using edge computing infrastructures. 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-farm distributed intelligent environments. This rapid growth in affordable IoT and edge-cloud systems has resulted in wider use of precision agriculture and rural-AI.

IoT infrastructures generally interconnect a diverse set of devices having sensors and actuators, that utilise communication protocols to exchange and collect critical data from end-users [15]. Edge computing [16] provides a task execution framework that critically handles processing and communication bottlenecks. It enables the processing of data at the network edge rather than relaying all data to a central cloud node [17]. This results in faster response times, higher quality of service, and improved security compared to centralised servers [18, 19].

2.1. Intelligent Applications using IoT and Edge

The use of robotics in agricultural applications has revolutionised farming practices with precision tasks, automated crop management, and data-driven decision-making processes [20]. A multi-robot system usually operates in a centralised or decentralised control setting [21]. A centralised strategy is proposed by Conesa-Munoz et al. [22], that involves administering the workflow of a designed system from primary base stations. 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, Tanner et al. [23] illustrated the interaction of two discrete groups of mobile agents, comprising a group of unmanned aerial and ground vehicles that adhere to time-invariant, nearest-neighbour rules for communication and interaction. Ground-based navigation is further adapted to align with the centroid of the ground group by utilising a Lyapunov-based optimisation approach.

Patros et al. [24] introduce an innovative solution to the rural agricultural environment having weak connectivity and high latency. Their framework utilises a serverless technology 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 the proof-of-concept for demonstration and evaluation in this work. Another work by Almurshed et al. [25] examined the ways in which edge-cloud computing can be utilised to address the unique challenges in rural areas. Recent surveys [26, 27, 28] have highlighted the potential of data collection in agricultural practices, enhancing productivity and sustainability on farms. These studies provide an overview of current methodologies, suggesting the increasing adoption and benefits of data-driven approaches in modern agriculture. These approaches yielded effective outcomes and improved the 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.

2.2. Load Balancing and Distribution

Naas et al. [29] 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 [30] describe a dynamic task offloading mechanism for a resource node where the 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. Maia et al. [31] 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. These proposed approaches outperform state-of-the-art algorithms, however, there are other factors such as security and failure risk that can still impact the performance of load balancing and distribution systems.

2.3. Security and Privacy

Over recent years, farm management has started to use Machine Learning (ML), AI, and IoT to improve the efficiency and sustainability of farming practices. However, these mechanisms also introduce new security risks and vulnerabilities. Vangala et al. [32] and Zanella et al. [33] 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. [34] also analyses the reluctance of farmers in sharing their agri-data and how this is affecting smart agriculture. A comparative analysis of recent research focusing on ML/AI, load distribution, application use cases, and workload management is provided in Table 1 below. Several recent studies have addressed the challenges of mobile-edge computing (MEC) in resource-constrained environments. Zhang et al. [35] proposed a joint load balancing and computation offloading technique with an additional security layer using encryption reinforced by ECG signals. Their approach demonstrated significant energy savings compared to local execution. Elgendy et al. [36] presented a multiuser resource allocation and computation offloading model that incorporates data security measures. Their integrated model aimed to minimise time and energy consumption while protecting sensitive information. In another work, Elgendy et al. [37] extended this work by introducing an efficient and secure multi-user multi-task computation offloading model. This model not only considered offloading strategies and resource allocation but also incorporated data compression techniques and security measures in their approach. However, these works focus on confidentiality in mobile-edge computing but overlook many aspects of data integrity and service availability in security.

Recent developments in load balancing [31, 38, 39, 40] and security [41, 42] identify approaches for managing intelligent agricultural applications in rural areas. However, existing research lacks focus on security-aware load balancing infrastructure which considers completion time, task security, load distribution, and failure risks, highly relevant in a rural environment

Work	ML Tasks	ML Pipeline	Security Evaluation	Load Sharing	Use-Case	Failure Examined	Serverless Evaluation	Workload Management
[15]	inference	centralised	no	yes	smart city	no	no	nil
[17]	training	distributed	no	no	industry	no	no	nil
[22]	none	none	no	yes	agriculture	no	no	nil
[23]	none	none	no	yes	UAVs	no	no	lyapunov analysis
[24]	training, aggregation	distributed	no	no	agriculture	no	yes	nil
[26]	none	none	no	yes	smart city	no	no	heuristic/graph modelling
[27]	none	none	no	yes	nil	no	no	integer linear program
[37]	none	distributed	availability	no	smart traffic	yes	no	greedy heuristic
[38]	none	none	confidentiality, integrity	yes	nil	no	no	nil
[34]	none	none	no	yes	nil	yes	no	formal modeling
SHIELD	inference, training, aggregation	distributed	confidentiality, integrity, availability	yes	agriculture	yes	yes	meta heuristics

Table 1. Comparative analysis of various studies focusing on ML, security, load distribution, and workload management in edge-based applications.

which is more prone to faults and has limited resilience. Therefore, there is need for a framework that distributes the load across resource nodes considering security, function chaining, failure rate, and resource utilisation.

Our proposed security infrastructure enhances existing precision agriculture systems, such as the tomato crop monitoring system by Devaraj et al. [43]. The three-tier architecture – comprising local sensors, farm-level gateways and regional edge servers – aligns well with our framework. Their system lacks security measures, whereas our approach supports encryption for sensor data transmission and access control for farm information.

3. System Model

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

3.1. 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. A graphical depiction of the three-tier architecture considered for designing and experimenting in our proposed SHIELD framework is provided in Figure 1 of this paper.

3.2. Task and Resource Classification

In order to limit the access of tasks on different layers of resources, we 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 farm-

ing 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, 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.

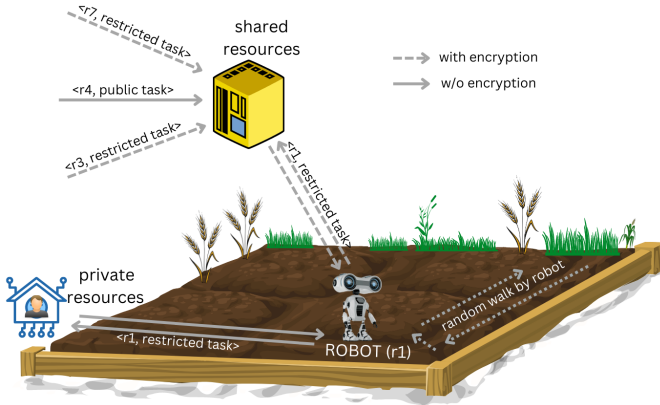


Figure 2. Load distribution of tasks on the agricultural field.

3.3. Real-World Use Case

For evaluation of our proposed framework, the edge-based architecture has been mapped to a real-life agricultural use case. We 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 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, we 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 2.

3.4. 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. We utilised encryption, hash-based message authentication code (HMAC), and access control mechanisms for ensuring confidentiality, availability, and integrity of data.

3.5. ML Model Description

In order to perform weed detection on the proposed framework, we have leveraged the *ResNet-50* [44] convolutional neural network model to classify weed species using the DeepWeeds [45] image classification dataset. This dataset includes

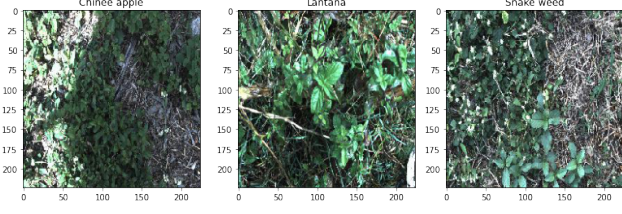


Figure 3. Three sample images from DeepWeeds dataset.

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 3. The *ResNet-50* model, known for its residual learning framework, was specifically selected for its efficacy in handling complex, high-dimensional data. We 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:

(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 describes the service function chain (SFC) for task variants utilised in this work. We 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 ImageNet [46] dataset.
- *Global model aggregation*: This task collects local mod-

els 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 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.

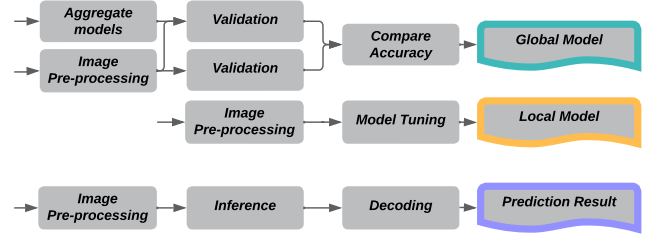


Figure 4. SFC for global, local, and prediction workflow.

3.6. 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 work.

Parsl [47] 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 [48] 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. We 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 Sections 6 and 8, we present detailed results from our experiments with these platforms, validating our approach's applicability in managing distributed serverless computing tasks.

3.7. The CIA Triad

For ensuring the protection of sensitive agricultural data, the fundamental concept of CIA triad is utilised in this work.

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, we 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 work to ensure CIA are based on well-known methods utilised for data protection [49]. We 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.

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
$t_{st}, t_{exec}, t_{comp}$	start, execution, completion time
$dl_{trans}, dl_{prop}, dl_{proc}$	transmission, propagation, processing delay
$dl_{queue}, dl_{lat}, dl_{comm}$	queuing, network, communication delay
t_{add}	additional time
len, bw	data packet length, network bandwidth
dis, vl	distance, velocity
n_{jq}, 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

Table 2. Symbol Table for the Problem Formulation

4. Problem Formulation

In this section, we now present a quantitative objective function for our proposed architecture which schedules IoT jobs in a load balanced manner. We have interchangeably used the term ‘task’ or ‘job’ in this work. Both these terms refer to the execution of a weed detection function. We have a set of J jobs and R resources such that a job can be sent on any available resource for execution. 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:

$$t_{cmp} = t_{st} + t_{exec} + dl_{lat} \quad (1)$$

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:

$$dl_{lat} = dl_{trans} + dl_{prop} + dl_{proc} + dl_{queu} \quad (2)$$

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, we consider that $dl_{proc} \approx 0$ in this work. Thus, the network latency can be written as:

$$dl_{lat} = dl_{trans} + dl_{prop} + dl_{queu} \quad (3)$$

$$dl_{lat} = \frac{len}{bw} + \frac{dis}{vl} + (njq \times awt) \quad (4)$$

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 work.

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, we have also considered additional time on resource nodes in this framework. It can be mathematically given as:

$$t_{add} = t_{exec} \times failure\ rate \quad (5)$$

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 be given by j_p respectively. The total number of tasks to be executed are given by N such that:

$$N = \sum j_r + \sum j_p \quad (6)$$

Furthermore, a restricted and public job that failed execution on the node is denoted by j_r' and j_p' respectively. The total number of jobs (out of N) that failed their execution are given by N' such that:

$$N' = \sum j_r' + \sum j_p' \quad (7)$$

To analyse the performance of task execution, we 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:

$$FR = \frac{\sum j_r' + \sum j_p'}{\sum j_r + \sum j_p} = \frac{N'}{N} \quad (8)$$

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

$$\underset{N'}{\operatorname{argmin}} FR = \{FR | FR = f(N')\} \quad (9)$$

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:

$$SC = c_{set} + c_{exec} + c_{comm} \quad (10)$$

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:

$$\underset{\tau}{\operatorname{argmin}} SC = \{SC | SC = f(\tau)\} \quad (11)$$

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:

$$RU = \frac{\sum t_{exec}}{cap} \quad (12)$$

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

$$\underset{t_{exec}}{\operatorname{argmax}} RU = \{RU | RU = f(t_{exec})\} \quad (13)$$

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:

$$PR = \max \left(\underset{N'}{\operatorname{argmin}} FR, \underset{\tau}{\operatorname{argmin}} SC, \underset{t_{exec}}{\operatorname{argmax}} RU \right) \quad (14)$$

Note that by maximising resource utilisation, we aim to optimally utilise resources for task execution. Similarly, by minimising the failure rate and total cost, we aim to avoid failure of tasks and also remove the expenditures which do not provide added value to our task execution framework.

5. The SHIELD Framework

This section describes the details of our load balancing framework proposed in this study. It is assumed that we have 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:

$$b - score(i, r) = utz_r - utz_{avg} \quad (15)$$

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, we observed that different functions performed better with different sets of k -values. For determining the dynamic k -value, we 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:

$$\delta_t = t_{exec} - t_{emax} \quad (16)$$

The δ_t undergoes the softplus transformation and is normalised by:

$$\frac{\text{softplus}(\delta_t)}{\text{softplus}(t_{emax})} \quad (17)$$

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:

$$k = \left\lceil R \times \left(0.1 + c_l \times \frac{\text{softplus}(\delta_t)}{\text{softplus}(t_{emax})} \right) \right\rceil \quad (18)$$

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 18. Figure 5 shows the relationship between the dynamic k -value and the average execution time of the function.

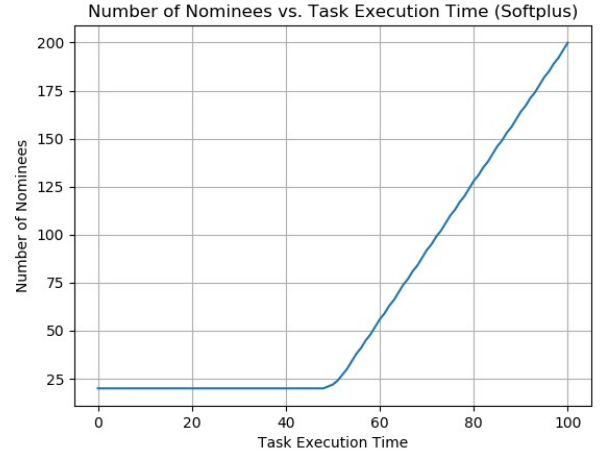


Figure 5. Correlation between dynamic k -value and average execution time of the function.

5.1. 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 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. We have calculated the additional time using the formula: $\text{additional time} = t_{exec} \times \text{failure rate}$. 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.

5.2. Adaptive Cryptographic Measures for Public Networks

The security solution we implemented uses the file system to transfer data between different resource nodes. In instances where the 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, demonstrated by 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 6 depicts the model's tuning procedure. This involves adding a security layer to nodes that are accessible through public networks. It can be seen that Fog Node₆ is reachable via such a network. As a result, a cryptographic approach is required to ensure its integrity and privacy. In this setup, the robot handles encryption and HMAC generation, while Fog Node₆ manages decryption and verification.

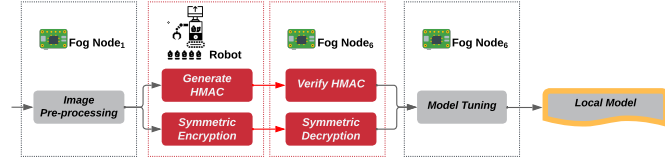


Figure 6. Model tuning on a public network. Robot encrypts, generates HMAC, and Fog Node₆ manages decryption, verification.

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

(1)- *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. (2)- *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. (3)- *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. (4)- *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 mini-

mal 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.

Algorithm 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  $t_i \rightarrow$  private then
17:     if selected node  $\rightarrow$  secure then
18:       perform execution
19:     else if selected node  $\rightarrow$  shared then
20:       encrypt task
21:       perform execution
22:     end if
23:   end if
24:   if  $t_i \rightarrow$  public then
25:     if selected node  $\rightarrow$  secure or shared then
26:       perform execution
27:     end if
28:   end if
29: end for
30: end procedure
  
```

5.3. 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, we 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, we 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 1.

5.4. Access Control

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, we have utilised two 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.

6. Performance Comparison: Parsl vs OpenWhisk

Table 3 below provides evaluation metrics for Parsl and OpenWhisk platforms, describing their performance when executing an FL application on a Raspberry Pi. 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 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.

Functions	Parisl 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 3. Performance readings for Parsl and OpenWhisk functions

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. There-

fore, 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.

7. Experimentation and Design

This section presents a detailed description of the experimental setup devised for 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.

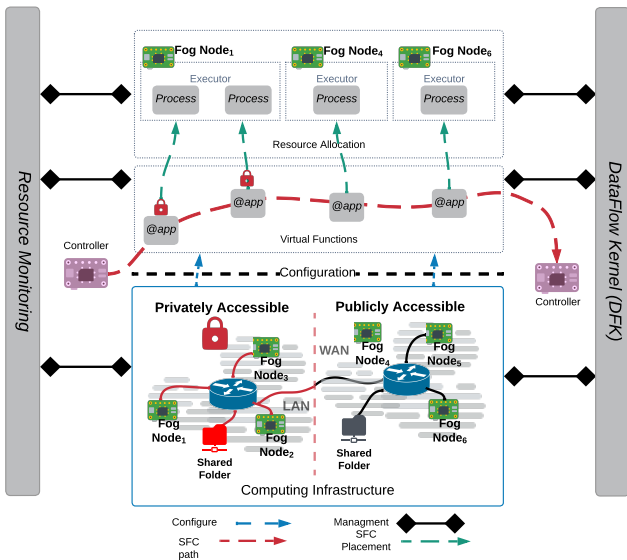


Figure 7. Utilising Parsl for orchestrating the execution of a workflow.

7.1. 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 (High-ThroughputExecutor) 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 7.

7.2. Testbed Setup for OpenWhisk

In order to test our framework on OpenWhisk platform, we 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, we 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.

7.3. Simulation Setup

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

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. We utilise a dynamic simulation model for emulating a system with single server per node, operating under a First-Come-First-Serve policy.

In Algorithm 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 2 Queue Simulator

Input: arrival time, execution time

Output: waiting time, completion time

```

1: procedure QUEUE-SIMULATOR()
2:   Initialize:  $wait\_end \leftarrow 0$ ,  $wait\_time \leftarrow 0$ 
3:   procedure CALC_WAIT_TIME( $arr\_time$ )
4:     if  $wait\_end < arr\_time$  then
5:        $wait\_end \leftarrow arr\_time$ 
6:     end if
7:      $wait\_time \leftarrow \max(0, wait\_end - arr\_time)$ 
8:     return  $wait\_time$ 
9:   end procedure
10:  procedure CALC_COMPL_TIME( $arr\_time$ ,  $exec\_time$ )
11:     $wait\_time \leftarrow \text{CALC\_WAIT\_TIME}(arr\_time)$ 
12:     $wait\_end \leftarrow \max(wait\_end, arr\_time + exec\_time)$ 
13:    if  $wait\_time = 0$  then
14:       $compl\_time \leftarrow exec\_time$ 
15:    else if  $wait\_time > 0$  then
16:       $compl\_time \leftarrow wait\_time + exec\_time$ 
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 [50], as shown in Figure 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 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 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.

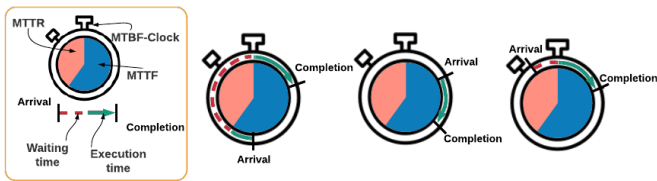


Figure 8. The execution of a service function at a process node, with completion times taking account of MTTF and MTTR.

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.

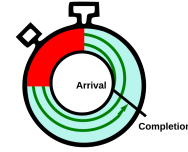


Figure 9. When the task execution time exceeds its MTBF, it will cycle repeatedly. At each MTTR interval, operation halts and then resumes until MTTF is reached.

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, we 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 $Completion\ Time = MTBF + MTTR$. If interrupted twice, the completion time extends to $Completion\ Time = MTBF + 2 \times MTTR$, 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.

7.4. Simulation Parameters

Table 4 highlights the configuration details of the simulation parameters utilised in this study. We evaluate 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

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. Summary of simulation parameters

MTTR range from 250s-500s and 20s-100s respectively in our evaluation. We 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.

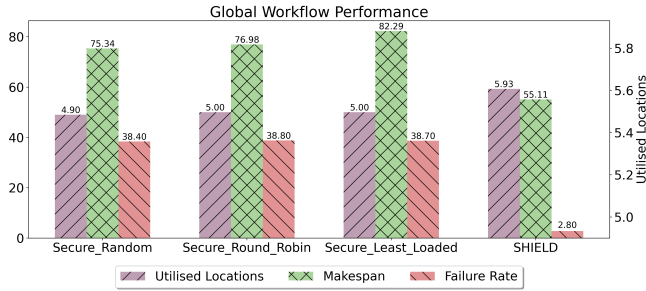


Figure 10. Global workflow evaluation on Parsl platform.

8. Results and Evaluation

This section presents the experimental findings and evaluations conducted on our security-aware load balancing framework. We utilised three key metrics to assess the 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 incompleteness. 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. We 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 section 5.4.

8.1. Performance Analysis of Workflows on Limited Resource Environment

For understanding the behaviour of framework with less number of resources, we 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 18 as depicted in Figure 5. Throughout this stage, we closely monitored the results and analysed the framework's effectiveness in load distribution with a smaller subset of resource nodes.

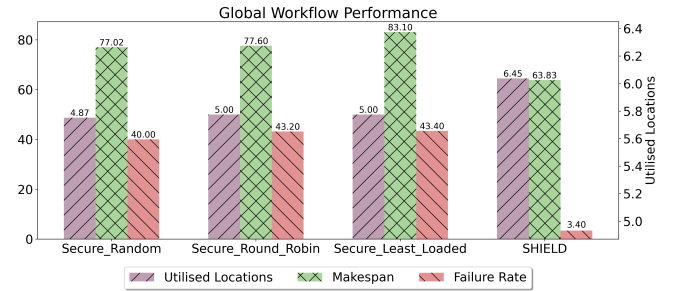


Figure 11. Global workflow evaluated on OpenWhisk platform.

Fig. 10 and 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 we increased the value of *k* to 100%, we 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).

The performance of local workflow on Parsl and OpenWhisk platforms are shown in Fig. 12, 13 respectively. In this workflow, SHIELD also demonstrated better performance over the other strategies. The makespan observed under SHIELD

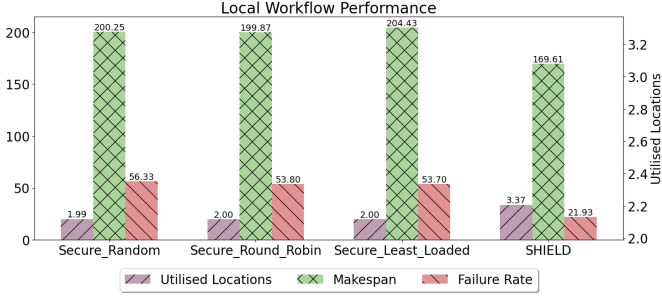


Figure 12. Local workflow evaluation on Parsl platform.

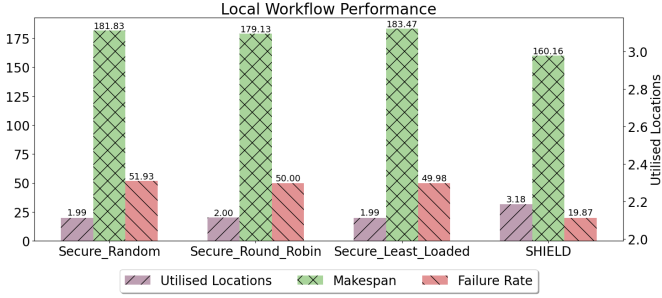


Figure 13. Local workflow evaluation on OpenWhisk.

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).

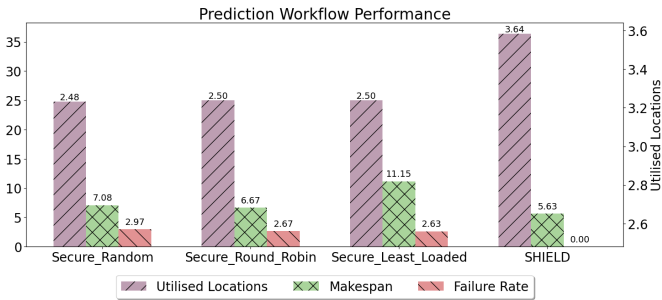


Figure 14. Prediction workflow evaluated on Parsl.

For the prediction workflow, the performance on Parsl and

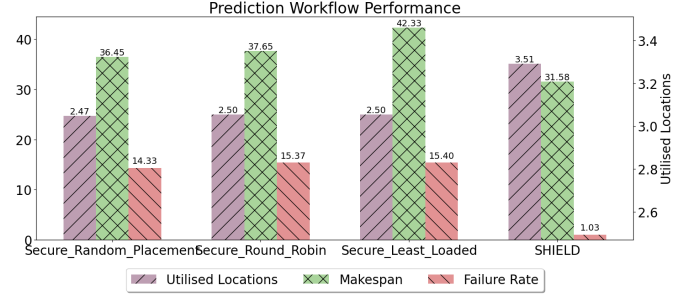


Figure 15. Prediction workflow evaluated on OpenWhisk.

OpenWhisk platforms are depicted in Fig. 14 and 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.

	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

Table 5. Average additional time on Parsl and OpenWhisk

8.2. 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 5 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.

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.

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

To analyse the load distribution capability of our proposed framework, we 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.

8.4. 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, we 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 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, we 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 12 and 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).

8.5. Analysing Performance with different Data Distribution and Task Load

All the previous experimentation in the work has been performed with uniform data distribution. To evaluate the proposed framework for a different probability of task arrival, we analysed the performance of SHIELD over Gaussian distribution of data, as detailed in [51]. We 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. We also noticed that the makespan for both distributions was similar, with a slight variation of 1s-3s, on average. Gaussian

distribution simulates a more realistic and varied workload scenario where some nodes might be more heavily loaded than others; SHIELD still outperformed the other strategies. However, the slight increase in failure rate can be observed due to changing load pattern of the distribution, which can be more challenging than managing a consistent, evenly spread-out load.

Moreover, as we increase the task load to 5x and 20x, we observe a notable trend in both Parsl and OpenWhisk platforms. In most cases, the makespan marginally increases or remains stable, indicating a 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%, depending on 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 the attached supplementary file [51].

8.6. Analysing Performance in Heterogeneous Resource Environment

In order to analyse the performance in a heterogeneous resource environment, we evaluated SHIELD framework with two types of hardware resources (RPi and Jetson Nano). We observed that SHIELD outperforms other approaches in utilising locations (4.42, 4.63) 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. Detailed results can be found in [51]. 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 well-suited for complex computations and scenarios where faster execution and success rates are critical for performance. However, OpenWhisk offers a 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%).

Comparing the performance of SHIELD, when evaluated on a uniform mix of resources, we observed similar trends across all workflows, for both Parsl and OpenWhisk. The results show that the resource utilisation in both the 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 GPUs, due to their complexity and higher memory requirements. Moreover, GPUs 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 GPU is reloaded, thus resulting in higher execution cost during task execution. Furthermore, GPU 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.

9. Conclusion

This research presents a framework for managing computational tasks within a rural edge infrastructure, establishing a balance between effective load distribution and maintaining data privacy and security of end-users. A heuristic-based algorithm is used to assign tasks on private or shared resources, taking account of completion time, failure rate, and resource utilisation. SHIELD (Secure Heuristic Integrated Environment for Load Distribution), utilises three different task scenarios: local model training, global model aggregation, and prediction model. 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.

There are several promising directions for further enhancement of SHIELD. One includes designing an infrastructure that combines both Parsl and OpenWhisk, utilising cases where one implementation is better than the other. This choice can improve the performance and efficiency of federated learning (FL) applications. Introducing self-healing mechanisms that handle node failure and enhance the reliability of system, along with optimisation can also be deployed. Another key direction is to handle system recovery using optimisation algorithms such that predicting potential failure of nodes and interconnecting networks can provide a useful basis for node selection within such optimisations. Collectively, these approaches can improve precision agriculture frameworks and enhance the efficiency of limited computational resources at the edge of the network.

References

- [1] M. J. Carrer, H. M. de Souza Filho, M. d. M. B. Vinholis, C. I. Mozambani, Precision agriculture adoption and technical efficiency: An analysis of sugarcane farms in Brazil, *Technological Forecasting and Social Change* 177 (2022) 121510.
- [2] P. Kumar, A. Nelson, Z. Kapetanovic, R. Chandra, Affordable artificial intelligence-augmenting farmer knowledge with AI, *arXiv preprint arXiv:2303.06049* (2023).
- [3] The State of Food Security and Nutrition in the World 2022 [Online]. Available: <https://www.fao.org/3/cc0639en/online/sofi-2022/introduction.html>.
- [4] R. Dumont, *Types of Rural Economy: Studies in World Agriculture*, Vol. 8, Taylor & Francis, 2023.
- [5] H. Yuan, H. Nie, Edge computing driven sustainable development: A case study on professional farmer cultivation mechanism, *Expert Systems* (2023).
- [6] A. D. Balmos, F. A. Castiblanco, A. J. Neustedter, J. V. Krogmeier, D. R. Buckmaster, Isobole avena: A framework for agricultural edge computing and data sovereignty, *IEEE Micro* 42 (1) (2022) 78–86. doi:10.1109/MM.2021.3134830.
- [7] Y. Kalyani, R. Collier, A systematic survey on the role of cloud, fog, and edge computing combination in smart agriculture, *Sensors* (17) (2021) 5922.
- [8] A. A. Alzubi, K. Galyna, Artificial intelligence and internet of things for sustainable farming and smart agriculture, *IEEE Access* (2023).

- [9] M. Trendov, S. Varas, M. Zeng, et al., Digital technologies in agriculture and rural areas: status report., Digital technologies in agriculture and rural areas: status report. (2019).
- [10] S. K. S. Tyagi, A. Mukherjee, S. R. Pokhrel, K. K. Hiran, An intelligent and optimal resource allocation approach in sensor networks for smart agri-IoT, *IEEE Sensors Journal* 21 (16) (2020) 17439–17446.
- [11] A. Mukherjee, S. Misra, A. Sukrutha, N. S. Raghuvanshi, Distributed aerial processing for IoT-based edge UAV swarms in smart farming, *Computer Networks* 167 (2020) 107038.
- [12] O. Friha, M. A. Ferrag, L. Shu, L. Maglaras, K.-K. R. Choo, M. Nafaa, FELIDS: Federated learning-based intrusion detection system for agricultural Internet of Things, *Journal of Parallel and Distributed Computing* 165 (2022) 17–31.
- [13] K. Haseeb, I. Ud Din, A. Almogren, N. Islam, An energy efficient and secure iot-based wsn framework: An application to smart agriculture, *Sensors* 20 (7) (2020) 2081.
- [14] T. H. Aldhyani, H. Alkahtani, Cyber Security for Detecting Distributed Denial of Service Attacks in Agriculture 4.0: Deep Learning Model, *Mathematics* 11 (1) (2023) 233.
- [15] J. He, J. Wei, K. Chen, Z. Tang, Y. Zhou, Y. Zhang, Multitier fog computing with large-scale IoT data analytics for smart cities, *IEEE Internet of Things Journal* 5 (2) (2017) 677–686.
- [16] S. Lu, J. Lu, K. An, X. Wang, Q. He, Edge computing on IoT for machine signal processing and fault diagnosis: A review, *IEEE Internet of Things Journal* (2023).
- [17] X. Wang, L. Ren, R. Yuan, L. T. Yang, M. J. Deen, Qtt-dlstm: A cloud-edge-aided distributed lstm for cyber-physical-social big data, *IEEE Transactions on Neural Networks and Learning Systems* (2022).
- [18] Q. Duan, S. Wang, N. Ansari, Convergence of networking and cloud/edge computing: Status, challenges, and opportunities, *IEEE Network* 34 (6) (2020) 148–155.
- [19] T. Wang, Y. Liang, X. Shen, X. Zheng, A. Mahmood, Q. Z. Sheng, Edge Computing and Sensor-Cloud: Overview, Solutions, and Directions, *ACM Computing Surveys* (2023).
- [20] R. Sparrow, M. Howard, Robots in agriculture: prospects, impacts, ethics, and policy, *precision agriculture* 22 (2021) 818–833.
- [21] G. Pantelimon, K. Tepe, R. Carriveau, S. Ahmed, Survey of multi-agent communication strategies for information exchange and mission control of drone deployments, *Journal of Intelligent & Robotic Systems* 95 (2019) 779–788.
- [22] J. Conesa-Muñoz, J. Valente, J. Del Cerro, A. Barrientos, A. Ribeiro, A multi-robot sense-act approach to lead to a proper acting in environmental incidents, *Sensors* 16 (8) (2016) 1269.
- [23] H. G. Tanner, D. K. Christodoulakis, Decentralized cooperative control of heterogeneous vehicle groups, *Robotics and autonomous systems* 55 (11) (2007) 811–823.
- [24] P. Patros, M. Ooi, V. Huang, M. Mayo, C. Anderson, S. Burroughs, M. Baughman, O. Almurshed, O. Rana, R. Chard, et al., Rural-AI: Serverless-powered federated learning for remote applications, *IEEE Internet Computing* 27 (2) (2022) 28–34.
- [25] O. Almurshed, P. Patros, V. Huang, M. Mayo, M. Ooi, R. Chard, K. Chard, O. Rana, H. Nagra, M. Baughman, et al., Adaptive edge-cloud environments for rural AI, in: *IEEE International Conference on Services Computing (SCC)*, 2022, pp. 74–83.
- [26] A. F. A. Fernandes, J. R. R. Dórea, G. J. d. M. Rosa, Image analysis and computer vision applications in animal sciences: an overview, *Frontiers in Veterinary Science* 7 (2020) 551269.
- [27] A. A. AlZubi, K. Galyna, Artificial intelligence and internet of things for sustainable farming and smart agriculture, *IEEE Access* (2023).
- [28] S. Condran, M. Bewong, M. Z. Islam, L. Maphosa, L. Zheng, Machine learning in precision agriculture: a survey on trends, applications and evaluations over two decades, *IEEE Access* 10 (2022) 73786–73803.
- [29] M. I. Naas, L. Lemarchand, J. Boukhobza, P. 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*, 2018, pp. 767–774.
- [30] S. Misra, N. Saha, Detour: Dynamic task offloading in software-defined fog for IoT applications, *IEEE Journal on Selected Areas in Communications* 37 (5) (2019) 1159–1166.
- [31] A. M. Maia, Y. Ghamri-Doudane, D. Vieira, M. F. de Castro, An improved multi-objective genetic algorithm with heuristic initialization for service placement and load distribution in edge computing, *Computer networks* 194 (2021) 108146.
- [32] A. Vangala, A. K. Das, V. Chamola, V. Korotaev, J. J. Rodrigues, Security in IoT-enabled smart agriculture: Architecture, security solutions and challenges, *Cluster Computing* 26 (2) (2023) 879–902.
- [33] A. R. de Araujo Zanella, E. da Silva, L. C. P. Albini, Security challenges to smart agriculture: Current state, key issues, and future directions, *Array* 8 (2020) 100048.
- [34] L. Wiseman, J. Sanderson, A. Zhang, E. 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 (2019) 100301.
- [35] W.-Z. Zhang, I. A. Elgendy, M. Hammad, A. M. Ilyasu, X. Du, M. Guizani, A. A. Abd El-Latif, Secure and optimized load balancing for multitier iot and edge-cloud computing systems, *IEEE Internet of Things Journal* 8 (10) (2020) 8119–8132.
- [36] I. A. Elgendy, W. Zhang, Y.-C. Tian, K. Li, Resource allocation and computation offloading with data security for mobile edge computing, *Future Generation Computer Systems* 100 (2019) 531–541.
- [37] I. A. Elgendy, W.-Z. Zhang, Y. Zeng, H. He, Y.-C. Tian, Y. Yang, Efficient and secure multi-user multi-task computation offloading for mobile-edge computing in mobile iot networks, *IEEE Transactions on Network and Service Management* 17 (4) (2020) 2410–2422.
- [38] D. Jiang, P. Zhang, Z. Lv, H. Song, Energy-efficient multi-constraint routing algorithm with load balancing for smart city applications, *IEEE Internet of Things Journal* 3 (6) (2016) 1437–1447.
- [39] R. Beraldi, C. Canali, R. Lancellotti, G. P. Mattia, Distributed load balancing for heterogeneous fog computing infrastructures in smart cities, *Pervasive and Mobile Computing* 67 (2020) 101221.
- [40] K. Ray, A. Banerjee, Prioritized fault recovery strategies for multi-access edge computing using probabilistic model checking, *IEEE Transactions on Dependable and Secure Computing* 20 (1) (2023) 797–812. doi:10.1109/TDSC.2022.3143877.
- [41] A. Aral, I. Brandić, Learning spatiotemporal failure dependencies for resilient edge computing services, *IEEE Transactions on Parallel and Distributed Systems* 32 (7) (2021) 1578–1590.
- [42] D. Puthal, R. Ranjan, A. Nanda, P. Nanda, P. P. Jayaraman, A. Y. Zomaya, Secure authentication and load balancing of distributed edge datacenters, *Journal of Parallel and Distributed Computing* 124 (2019) 60–69.
- [43] H. Devaraj, S. Sohail, B. Li, N. Hudson, M. Baughman, K. Chard, R. Chard, E. Casella, I. Foster, O. Rana, Ruralai in tomato farming: Integrated sensor system, distributed computing and hierarchical federated learning for crop health monitoring, *IEEE Sensors Letters* (2024).
- [44] B. Koonce, ResNet 50, Convolutional Neural Networks with Swift for Tensorflow: Image Recognition and Dataset Categorization (2021) 63–72.
- [45] A. Olsen, D. A. Konovalov, B. Philippa, P. Ridd, J. C. Wood, J. Johns, W. Banks, B. Girgenti, O. Kenny, J. Whinney, et al., DeepWeeds: A multiclass weed species image dataset for deep learning, *Scientific reports* 9 (1) (2019) 2058.
- [46] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database (2009) 248–255.
- [47] Y. Babuji, A. Woodard, Z. Li, D. S. Katz, B. Clifford, R. Kumar, L. Lacin-ski, R. Chard, J. M. Wozniak, I. Foster, et al., Parsl: Pervasive parallel programming in python (2019) 25–36.
- [48] A. Alabbas, A. Kaushal, O. Almurshed, O. Rana, N. Auluck, C. Perera, Performance analysis of apache openwhisk across the edge-cloud continuum.
- [49] R. Diesch, M. Pfaff, H. Krcmar, A comprehensive model of information security factors for decision-makers, *Computers & Security* 92 (2020) 101747.
- [50] O. Almurshed, O. Rana, K. Chard, Greedy nominator heuristic: Virtual function placement on fog resources, *Concurrency and Computation: Practice and Experience* 34 (6) (2022) e6765.
- [51] A. Kaushal, O. Almurshed, O. Almoghamis, A. Alabbas, N. Auluck, B. Veeravalli, O. Rana, Supplementary Material, <https://github.com/ashkaushal/Supplementary-Material-SHIELD.git>.