An Edge-Cloud Infrastructure for Weed Detection in Precision Agriculture

Ashish Kaushal*†, Osama Almurshed†‡, Areej Alabbas†, Nitin Auluck*, and Omer Rana†
ashish.19csz0003@iitrpr.ac.in, almurshedo@cardiff.ac.uk, alabbasam@cardiff.ac.uk, nitin@iitrpr.ac.in, and ranaof@cardiff.ac.uk
Indian Institute of Technology Ropar, India*
Cardiff University, United Kingdom†
Prince Sattam Bin Abdulaziz University, Kingdom of Saudi Arabia‡

***Abstract*—Accurate identification of weeds plays a crucial role in helping farmers achieve efficient agricultural practices. An edge-cloud infrastructure can provide efficient resources for weed detection in resource-constrained rural areas. However, deployed applications in these areas often face challenges such as connectivity failures and network issues that affect their quality of service (QoS). We introduce a signal quality-aware framework for precision agriculture that allocates weed inference tasks to resource nodes based on the current network connectivity and quality. Two Machine Learning (ML) models based on *ResNet-50* and *MobileNetV2* are trained using the publicly available DeepWeeds image classification dataset. A rule-based approximation algorithm is formulated to execute tasks on resource-constrained computational nodes. We also designed a testbed setup consisting of Raspberry Pi (RPi), personal laptop, cloud server and Parsl environment for evaluating the framework. Reliability of the framework is tested in a controlled setting, under various dynamically injected faults. Experimental results demonstrate that the proposed setup can accurately identify weeds while ensuring high fault tolerance and low completion time, making it a promising solution for weed management in rural agriculture.**

***Index Terms*—edge-cloud computing, fault tolerance, Parsl, precision agriculture, robots, rural-AI, weed detection.**

I. Introduction

With a rapid increase in global population, there is an ever-growing need to ramp up sustainable food production. The Food and Agriculture Organisation (FAO) estimates that healthy diets are still unaffordable to over 3B people across the world, and a majority of these people are from low and middle income countries [1]. Efficient weed management in agricultural fields is an important factor that can improve crop productivity. Due to the spatial and temporal heterogeneity of weeds/plants in agricultural fields, many robotic weed control methods (aerial and ground) have been developed for site-specific weed management [2].

The effective use of robotic technology can benefit farmers by improving crop yield and reducing production costs, but it faces many challenges in real-life rural environments. Two challenges that affect the performance of such systems are: network availability and reliability. While network availability ensures that the infrastructure is available and operational at all times, reliability ensures that the infrastructure has been successfully deployed, and is operating error-free [3]. It is expected that a reliable network will maintain a high standard of service, even in the event of system faults [4]. This allows a system to operate without interruption when one or more faults occur. Fault detection and automated correction play a key role in supporting a system's fault tolerance.

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

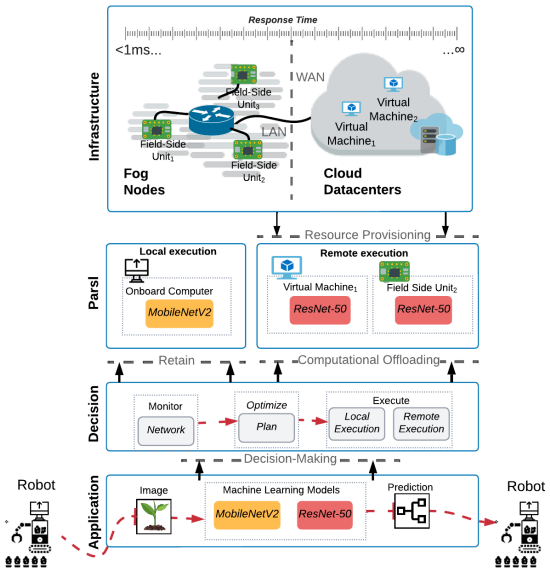


Fig. 1: Architecture of the proposed framework

One such application, namely Rural-AI, allows the use of machine learning (ML) within rural communities, with an aim to achieve better performance in terms of productivity, economic growth, impact of climate change and affordability [5]. Rural-AI [6] is the *engineering of cyber-physical systems for enabling sovereign, sustainable AI in locations with limited and/or unreliable power/networking infrastructure*. Due to the lack of proper development and infrastructure in rural areas,

979-8-3503-0460-2/23/$31.00 © 2023 IEEE

ML/AI applications cannot be utilised to their full potential. Service outage along with unreliable network connectivity are two key challenges that significantly affect the growth of rural economy. These issues prevent IoT infrastructure from being efficiently deployed and used [7]. Therefore, there is a need for a framework that is capable of providing a *reasonable quality of service* (QoS), even when a connection failure occurs.

A framework for edge-cloud infrastructure that detects connection errors and adapts to such errors is proposed. It also triggers a fault tolerance mechanism to ensure that computational/AI tasks are executed with a minimal loss to performance, even when a network fault occurs. The architecture of our proposed framework is depicted in Figure 1. To demonstrate the efficacy of this conceptual architecture, two ML models based on *ResNet-50* and *MobileNetV2* have been trained for identifying weeds, using images captured from a robot mounted camera. Light version of these two models have been generated respectively using model quantization techniques – a process involving size reduction of the learned model, at the expense of accuracy, e.g. by mapping model parameters from floating-point numbers to low-precision fixed-point numbers [8].

Our primary goal is to prioritise the use of highly accurate pre-trained models for inference. As this inference is carried out remotely on computational units of the farm site, it can take longer time to execute. However, if we are unable to connect with field side units due to network faults, local models are utilised for inference. While this local model is moderately accurate, its execution time is significantly lower than the full model. An algorithm based on task deadlines is utilised to determine the location for inference. The task execution is then evaluated using a testbed created for this framework. The main contributions of this work are as follows:

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

The remainder of this paper is organised as follows: Section II outlines recent work in adaptive edge-cloud frameworks. The components and associated pre-requisites for the framework are described in Section III. Section IV describes fault-tolerance mechanisms supported and the proposed decision making algorithms. Section V and VI include simulation experiments and results. Section VII provides concluding remarks and future work for the designed framework.

II. Adaptive Edge-Cloud Frameworks

Nvidia Jetson (Nano) and Raspberry Pi are now widely used to support edge-based AI computing. Dependable use of these devices require addressing operational issues such as inconsistent internet, communication delays, and service disruptions – that require proactive strategies [9], [6]. Fault-tolerance is a crucial requirement of agricultural robots, especially as they need to be operated by non-experts. Researchers have extensively focused on fault-tolerance across platform and infrastructure layers, such as use of self-adaptive systems which provide quick backups and reduced recovery times [10]. Other methods like greedy nominator heuristic ensure service reliability through service replication [11], [12]. With the rising demand of AI in rural environments, such applications require support for fault-tolerance, to ensure performance and efficiency in agricultural applications.

Recent literature highlights advancements in weed detection and robotic weed management in agriculture, such as studying weed classification using AI [13], or adapting to rural infrastructure to securely train an AI model [14]. However, literature addressing infrastructure unreliability, especially in rural farming areas [6], [15] is limited. Even though there are researchers working on optimising machine learning inference to reduce costs [16], delays [17], [16], and balance workload [17], they often lack solutions tailored for agricultural settings in rural areas. This work focuses on edge-deployed applications for agriculture, especially when using an unreliable (rural) data network.

III. Configuring the Infrastructure

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

A. Serverless Computing Platforms

A number of serverless platforms are available – ranging from those that are: (i) used commercially, such as [1]Amazon Lambda, Google functions, Azure functions, etc; (ii) available as open source systems, such as Apache OpenWhisk, [2]Fissions, [3]OpenFaaS, etc. Some variants include pre-deployed commercial versions of open source platforms, e.g. OpenFaaS Pro, which offers additional features and support.

These platforms differ in the types and range of capabilities they offer, for instance, some utilise an existing pre-deployed platform (e.g. Kubernetes) enabling users to write and offload executable functions. These types of platforms enable users to build and manage their own functions, rather than the infrastructure on which these functions are hosted. Others include support for deploying (and managing) the hosting environment on which these functions are executed (e.g. OpenWhisk). *Parsl* [18] provides a Python-based development environment for functions, and can be hosted on both edge devices (e.g. RPi) and on a high performance computing cluster. This is achieved through the use of custom executors designed for the resource being used in the architecture. *Parsl* also provides the basis for dynamically distributing functions

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

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

B. Dataset and ML Models

DeepWeeds [19] is a multiclass image dataset for deep learning consisting of 17,509 images. These images belong to 8 different categories of weeds found in vast regions of northern Australia. This dataset contains 15K training images and 2.5K test images of size 256x256 pixels. Two different ML models: *ResNet-50* [20], and *MobileNetV2* [19] have been trained on the DeepWeeds image classification dataset and utilised for plant/weed identification in this work.

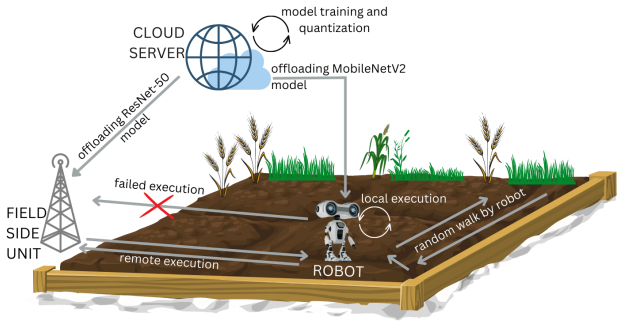


Fig. 2: Interaction between robot, FSU, and cloud node in the agricultural field

C. System Design and Use Case

We develop a three-tier architecture comprising of IoT devices, storage, and computation resources positioned in a hierarchical manner. The control flow of the framework is based on a master-worker configuration. We consider a rural agriculture use case where robots identify and remove weeds. Robots interact with Field Side Units (FSUs) that are located in proximity to a robot on the same field.

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

IV. Proposed Approach

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

A. ML Models

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

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

B. Signal Monitoring

A signal monitor continuously examines the network connectivity between the robot and the FSU. We have divided the 'signal' parameter into three different categories: no signal, low signal, and strong signal. When the robot is unable to establish connection with an FSU, it is considered to have no signal, no distortion between the robot and FSU indicates a

strong signal, and low signal falls between these conditions. During our experiments, when the signal strength drops below a threshold value, it is considered as a low signal; otherwise the signal is considered to be a strong signal.

C. Execution Workflow for Inference Tasks

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

Our deadline-aware approach starts by initially checking the network connection. Depending on the current network signal quality, a deadline is established for each task and the inference process is initiated if the connection is available. The estimated deadline is the approximate time it takes to perform ML inference on a FSU, if the current signal quality is maintained throughout the execution of task. It is calculated by considering the execution time on a FSU, transmission time and link time for completing the inference. The algorithm offloads ML tasks to remote FSUs and starts a timer. If the outcome of inference task is not retrieved before the deadline, the task is immediately discarded. Inference is then performed using a local model. We have assumed that if the results are not retrieved within the deadline, a network fault might have occurred, thereby causing the delay. Executing a model locally ensures that a reasonably accurate result is achieved, even when the inference fails on the FSU. The proposed deadline-aware approach is described in Algorithm 1.

Algorithm 1 Deadline-Aware Approach

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

Another signal quality-aware approach for the proposed fault tolerance mechanism can be considered. The signal quality of the network is repeatedly monitored and a threshold value is fixed before we begin execution. If signal quality is found below the specified threshold value, it is considered to have low quality for performing the FSU-based execution and inference is directly performed on a local node. This is because low signal increases the probability of task failure in case any network fault occurs. By adjusting the threshold value, the sensitivity of model performance can be controlled in the framework. In a real-life scenario, the threshold can be specified based on the network requirements of that application.

Fig. 3: Weed image: (a) original (b) blurred (c) black patched

V. Experiments and Simulation

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

A. Experimental Setup

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

B. Testing ML Model Capabilities

In a real-world application, it is not possible for the robot to capture perfect images of plants and weeds all the time. Sometimes, due to extraneous factors such as weather condition and crop density, the images captured are either unclear or have obstructed line of sight. This results in unfavourable conditions for model evaluation and can affect system performance and efficiency. To test the ML model capabilities for environmental variations, we have intentionally injected noise in the images and then performed evaluation of the generated models. The noise has been injected in images in mainly two forms: blur and black patch. Blur function is applied on the entire image, whereas two random size black patches are applied at a random location in the image. The blurred image can simulate a situation when there is dust, rain drops on the camera lens that make the entire image unclear. In a similar manner, the black patched image can replicate a scenario when a leaf/insect is covering part of the lens or there is a stem obstructing the

line of sight to the actual object. A sample weed image with blur and black patch is described in Figure 3. It is important to note that we have not performed any training using noisy images. However, blur and black patch noise was injected in all the test images of DeepWeeds dataset, and then *ResNet-50* and *MobileNetV2* were used for evaluation. The accuracy observed with and without noisy images for *ResNet-50* and *MobileNetV2* models are shown in Table I.

TABLE I: Model accuracy with and without noisy images

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

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

C. Node Selection for Training and Inference

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

TABLE II: Time for training 1 epoch on 1 image

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

It can be observed that the training time on edge node, FSU is 4x and 10x more respectively, than the cloud node, for both the models. Therefore, we have selected the cloud node for training our two ML models and generate light versions of these models. It takes around 64sec, 42sec to generate light models of *ResNet-50* and *MobileNetV2* respectively.

A stress test to analyse the performance capabilities of two models on an edge node and FSU is also presented in this work. We ran concurrent inferences of weed identification tasks and observed the change in their waiting and execution times. Tables III and IV describe the average waiting and execution time for one inference when *n* concurrent inferences are performed.

TABLE III: Concurrent ML inferences on edge node

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

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

If we consider the following four factors for decision making: (1) model adaptation to a harsh environment, (2) number of concurrent executions possible, (3) average execution time, and (4) average waiting time, we need to establish a trade-off between choosing a more accurate model or reduce time to develop a model. Therefore, in this work, we have decided to deploy the highly accurate *ResNet-50* model on the FSU (as a primary option), but if the network connection is unreliable/ unavailable, we can resort to utilising the *MobileNetV2* model for local inference, which offers faster processing than the former (with reasonable accuracy).

TABLE IV: Concurrent ML inferences on FSU

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

D. Execution Workflow

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

E. Communication and Monitoring Setup

It is difficult to estimate an exact signal strength throughout the communication channels. Therefore, we have estimated the signal strength on controller node and used it as a reference point throughout the inference. In simulation, patterns are generated through a random process for determining the signal quality of each task. For an ideal circumstance, the

TABLE V: Simulation Parameters

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

time it takes to send data over the network is given as $(link\ time = data\ size/link\ speed)$. The link transfer speed considered in this work is 1Mbps. Link delay is determined as $(link\ delay = link\ time/quality)$. When the quality is set to 1.0, the wireless link is utilised at full capacity and the execution is performed at the fastest possible speed. When quality drops below 1.0, the link delay increases and the task takes more time to execute. This technique has been utilised to estimate variable network latency in the wireless network. A list of all the variables considered in this simulation are also described in Table V.

VI. Results and Evaluation

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

A. Results

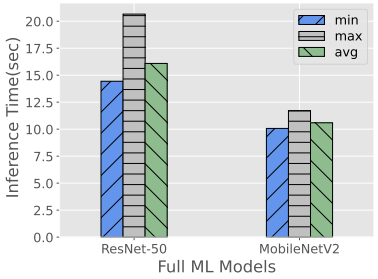
Fig. 4a and 4b display the average time taken to perform image inference on full and light models, respectively. We have only utilised light models in this work, but in order to justify not using full models, we have performed inference on full models as well. The results show that inference time on full models are almost 10x in comparison to the light models. Therefore, light models are a better choice over full models for weed/plant inference. TensorFlow Lite models have a smaller file size compared to TensorFlow, and the light model can be directly accessed without the need for additional parsing or unpacking steps, which in turn speeds up the inference process. As a result, this allows a time efficient and effective execution of ML inference tasks on resource constrained devices, having low memory and less computational power in comparison to cloud nodes. We note that execution time for *MobileNetV2* based model is less than *ResNet-50* model – as *ResNet-50* has 177 layers and about 25.5M parameters, while *MobileNetV2* has 156 layers and only 3.5M parameters.

A comparison of inference time on two different platforms, namely *Parsl* and *Lean OpenWhisk*, is shown in Fig. 4c. We wanted to test our model execution on another comparable serverless platform. Lean OpenWhisk is a lightweight version of the open-source OpenWhisk serverless computing platform that can be deployed on the edge layer, and offers all the basic functionalities of the full version of OpenWhisk. We selected locally executed *MobileNetV2* model as a task for this evaluation. The results show that execution time on *Parsl* is significantly less in comparison to OpenWhisk. This is because Docker instances are created and initialised for performing execution on OpenWhisk platform. However, *Parsl* functions can be directly run on the node using pre-developed executors. We realised that using these custom executors and dynamic function distribution makes *Parsl* a good platform for performing real-time function execution.

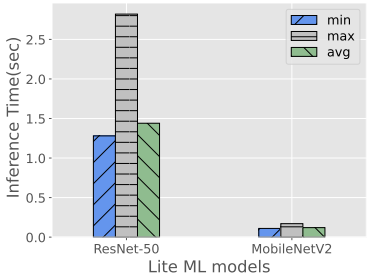
Fig. 5a illustrates the completion time of deadline-aware and quality-aware methods put forward in the proposed weed inference model. As the threshold for signal quality increases, the completion time shortens, as more inference tasks will utilise the local model for predictions instead of the full model. The completion time for the deadline-aware approach is significantly higher in comparison to signal quality-aware approach. Using the deadline-aware approach, the system has to wait for the deadline to expire before executing the local model for prediction, whereas with the signal based approach it immediately runs the local predictions if the signal quality is below the specified threshold. Moreover, the completion time for the random allocation approach is high in this experimentation. This is because the algorithm selected full models for most of the evaluations, resulting in high execution time. It can also be verified by high accuracy of random allocation approach observed in Fig. 5c (because full models are more accurate). For round-robin approach, the completion time increased from 0.78sec to 1.28sec with an increase in signal quality. At low threshold values, most of the jobs are unable to complete execution because of poor signal quality. However, as the signal quality increases, both completion time and accuracy increases – as the approach selects full models for execution alternatively and successfully completes the execution.

The Fig. labeled 5b illustrates the impact of increasing accuracy on task completion time. It is observed that completion time increases as high accuracy is achieved by the evaluation model. To improve accuracy, we need to perform the inference by utilising the full model (which takes more time to execute). Even at low accuracy, the computation time for the deadline-aware approach is much higher than that offered by the quality-aware approach. This is because the deadline-aware approach uses a local model for inference after the deadline has expired, which adds extra latency to the overall execution time of proposed framework.

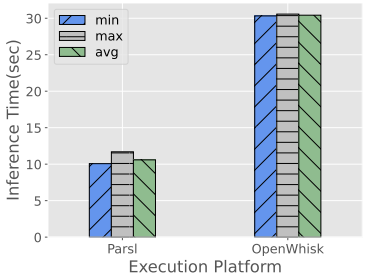
It can be seen in Fig. 5c that for low signal quality threshold, the accuracy of inference is high. This is because for most of the inferences, current signal quality will be above the threshold and it will be using the full model for inference. However, as the threshold value increases, higher number of jobs will be running the local model which is less accurate. Hence, the accuracy decreases with increase in signal threshold. Along with that, it can also be observed that both the approaches are giving almost same accuracies with changing threshold values. This is because in both the approaches,



(a) avg. inference time on full models

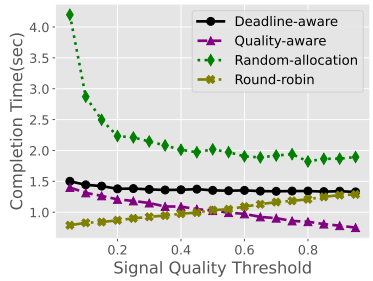


(b) avg. inference time on light models

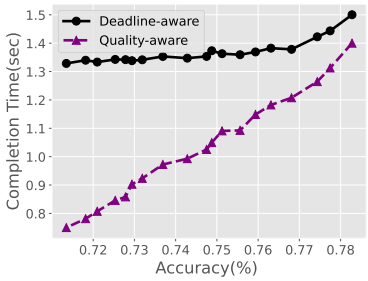


(c) avg. inference time on 2 platforms

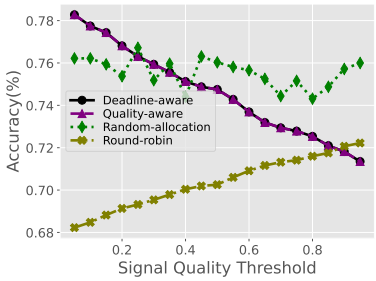
Fig. 4: Average plant/weed inference time for different ML models



(a) Effect on completion time with change in signal quality



(b) Effect on completion time with change in accuracy



(c) Effect on accuracy with change in signal quality

Fig. 5: Performance evaluation of the proposed algorithms

threshold is a measure that mainly affects the decision on where the execution will happen. For signal quality-aware approach, the threshold decides whether to execute the job locally or remotely, whereas in deadline-aware approach, the threshold is used for estimating the time it will take to execute the job remotely. The estimated time is then compared with job deadline to evaluate whether the remote execution is success or not. Therefore, the same model (either local or remote) is picked for evaluation in both the approaches. The accuracy for random allocation is high but the completion time for this approach is also higher because of randomly picking full models for execution. For the round-robin approach, the full models are selected and successfully executed with an increase in signal threshold. This is because high threshold value ensures a higher chance of successful task execution (with the current signal quality).

B. Analysis

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

VII. Conclusion and Future Work

An edge-cloud framework that can be used within mobile agricultural robots under intermittent network connectivity is proposed. This approach is aimed at addressing network faults, such as unreliable connections and service outages, that can significantly affect the performance of precision agriculture applications. Using on-board machine learning (ML) models for classification and inference, the robot analyses plants/ weeds by taking images through a robot-mounted camera. For demonstration, two ML models were trained for weed identification and prediction using the DeepWeeds image classification dataset with noise.

We evaluate our algorithm using experiments performed on a testbed, demonstrating that our approach provides accurate predictions under variable network signal quality. The proposed approach offers better performance in terms of completion time, whereas a more traditional deadline-aware approach (used as a comparison) is more accurate but takes longer to execute. Moreover, our approach can be extended and adapted to other image classification or object detection datasets in precision agriculture. By training ML models on specific datasets, similar frameworks can be developed for identifying other types of plants, diseases or pests.

To enhance classification results, we will extend our approach with optimisation techniques like genetic algorithms or particle swarm optimisation. However, real-time edge analysis may require specific variants of these methods due to their high computational demands. We also aim to design a real-time dynamic approach that automatically selects the most suitable algorithm for inference (from multiple locally available options) based on past system performance and user-defined threshold values.

REFERENCES

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

[2] C. Hu, J. A. Thomasson, and M. V. Bagavathiannan, "A powerful image synthesis and semi-supervised learning pipeline for site-specific weed detection," *Computers and Electronics in Agriculture*, vol. 190, p. 106423, 2021.

[3] S. S. Gill, "Quantum and blockchain based serverless edge computing: A vision, model, new trends and future directions," *Internet Technology Letters*, p. e275, 2021.

[4] A. Samanta, F. Esposito, and T. G. Nguyen, "Fault-tolerant mechanism for edge-based IoT networks with demand uncertainty," *IEEE Internet of Things Journal*, vol. 8, no. 23, pp. 16963–16971, 2021.

[5] H. Saxby, M. Gkartzios, and K. Scott, "'farming on the edge': wellbeing and participation in agri-environmental schemes," *Sociologia Ruralis*, vol. 58, no. 2, pp. 392–411, 2018.

[6] 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*, 2022.

[7] N. Brisson, C. Gary, E. Justes, R. Roche, B. Mary, D. Ripoche, D. Zimmer, J. Sierra, P. Bertuzzi, P. Burger, F. Bussière, Y. Cabidoche, P. Cellier, P. Debaeke, J. Gaudillère, C. Hénault, F. Maraux, B. Seguin, and H. Sinoquet, "An overview of the crop model stics," *European Journal of Agronomy*, vol. 18, no. 3, pp. 309–332, 2003. Modelling Cropping Systems: Science, Software and Applications.

[8] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

[9] J. P. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Computer Networks*, vol. 54, no. 8, pp. 1245–1265, 2010.

[10] M. A. Kamel, X. Yu, and Y. Zhang, "Formation control and coordination of multiple unmanned ground vehicles in normal and faulty situations: A review," *Annual reviews in control*, vol. 49, pp. 128–144, 2020.

[11] O. Almurshed, O. Rana, and K. Chard, "Greedy nominator heuristic: Virtual function placement on fog resources," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 6, p. e6765, 2022.

[12] O. Almurshed, O. Rana, Y. Li, R. Ranjan, D. N. Jha, P. Patel, P. P. Jayaraman, and S. Dustdar, "A fault tolerant workflow composition and deployment automation IoT framework in a multi cloud edge environment," *IEEE Internet Computing*, 2021.

[13] 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*, vol. 9, no. 1, pp. 1–12, 2019.

[14] 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 *2022 IEEE International Conference on Services Computing (SCC)*, pp. 74–83, IEEE, 2022.

[15] R. Hadidi, J. Cao, M. S. Ryoo, and H. Kim, "Toward collaborative inferencing of deep neural networks on internet-of-things devices," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4950–4960, 2020.

[16] X. Chen, J. Zhang, B. Lin, Z. Chen, K. Wolter, and G. Min, "Energy-efficient offloading for DNN-based smart iot systems in cloud-edge environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 3, pp. 683–697, 2021.

[17] S. Javanmardi, M. Shojafar, V. Persico, and A. Pescapè, "FPFTS: a joint fuzzy particle swarm optimization mobility-aware approach to fog task scheduling algorithm for internet of things devices," *Software: Practice and Experience*, vol. 51, no. 12, pp. 2519–2539, 2021.

[18] Y. Babuji, A. Woodard, Z. Li, D. S. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, J. Wozniak, I. Foster, M. Wilde, and K. Chard, "*parsl*: Pervasive parallel programming in python," in *28th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2019.

[19] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.