2023 TinyML Challenge: Scalable and High-Performance TinyML Solutions for Wildlife Monitoring

FINAL REPORT

TEAM MEMBERS:

S/N	Full Name	Contribution
1	Dr. Jabhera Matogoro	Mentor
2	Zephania Reuben	Model Development
3	Madaraka Marco Masasi	Model & Prototype Development
4	Ipyana Issah Mwaisekwa	Model Development & Documentation
5	Paul Mkai	Documentation
6	Rogers kalunde	Model development
7	Ramadhani Massawe	Model Development
8	Fatuma Issa	Documentation

10th November 2023

Table of content

1 Introduction ... 1

1.1 Project Objectives ... 1

2 Architecture and Design ... 3

2.1 Data Preprocessing and Feature Engineering ... 3

2.2 Model Selection and Training ... 3

2.3 Model Evaluation and Validation ... 4

2.4 Model deployment ... 4

3 Results and Discussion ... 5

3.1 Model performance ... 5

3.2 Model Evaluation Results ... 6

3.2.1 Requirements Specification ... 7

3.3 Discussion ... 8

4 User Documentation ... 13

4.1 Introduction ... 13

4.2 User Documentation ... 13

4.2.1 Installation and Setup ... 13

1.1.1 Launching the Application ... 14

4.2.2 User Interface ... 14

4.2.3 Core Functionality ... 15

4.2.4 Interpreting Results ... 16

4.2.5 Customization (if applicable) ... 16

4.2.6 Saving and Exporting Data (if applicable) ... 16

5 Testing and Quality Assurance ... 17

5.1 Testing Methodologies ... 17

5.1.1 Unit Tests ... 17

5.1.2 Integration Tests ... 17

5.1.3 System Tests ... 17

5.2 Bug Reports and Test Results ... 18

5.2.1 Bug Reports ... 18

5.2.2 Test Results ... 18

6 Version Control and Change Log ... 19

6.1 Version Control System (VCS) 19
6.1.1 Git Repository Setup 19
6.1.2 Branch Management 19
6.1.3 Collaborative Workflow 19
6.2 Change Log 20
6.2.1 Change Log Structure 20
6.2.2 Updating the Change Log 20
6.2.3 Release Notes 20
7 Acknowledgements and Contributors 21
7.1 Acknowledgements 21
7.1.1 Gratitude to Contributors 21
7.1.2 Categories of Acknowledgments 21
7.2 Contributors 21
7.2.1 Identification of Contributors 21
7.2.2 Categories of Contributions 22

1 Introduction

The peripheral of the Internet of Things (IoT) ecosystem presents a notable prospect for the implementation of intelligent decision-making capabilities. The incorporation of machine learning algorithms into compact, resource-constrained embedded systems has become indispensable owing to the forward-looking nature of their applications. Hence, it is imperative to conduct an extensive investigation into the TinyML paradigm in order to further enhance the existing state of edge-aware machine learning. The transition towards the TinyML paradigm can be accomplished by placing careful emphasis on in-memory processing and neuromorphic computing. Additionally, it involves the creation of benchmarks, optimized datasets, TinyML process flows, and the seamless integration of TinyML into affordable and energy-efficient smart handheld devices, such as smartphones, microcontrollers, and IoT edge systems. The potential of TinyML is considerable, as it has the capacity to bring about a revolutionary transformation in the prevailing decision-making methodology, thereby effectively tackling big difficulties in smaller segments, particularly within edge networks that are limited in resources.

The TinyML models can effectively identify and classify animals from the camera trap dataset containing images/photos captured in natural habitats, featuring diverse wildlife species and their activities images while maintaining the ability to scale across different environments and species. These solutions contribute to wildlife monitoring and various conservation initiatives.

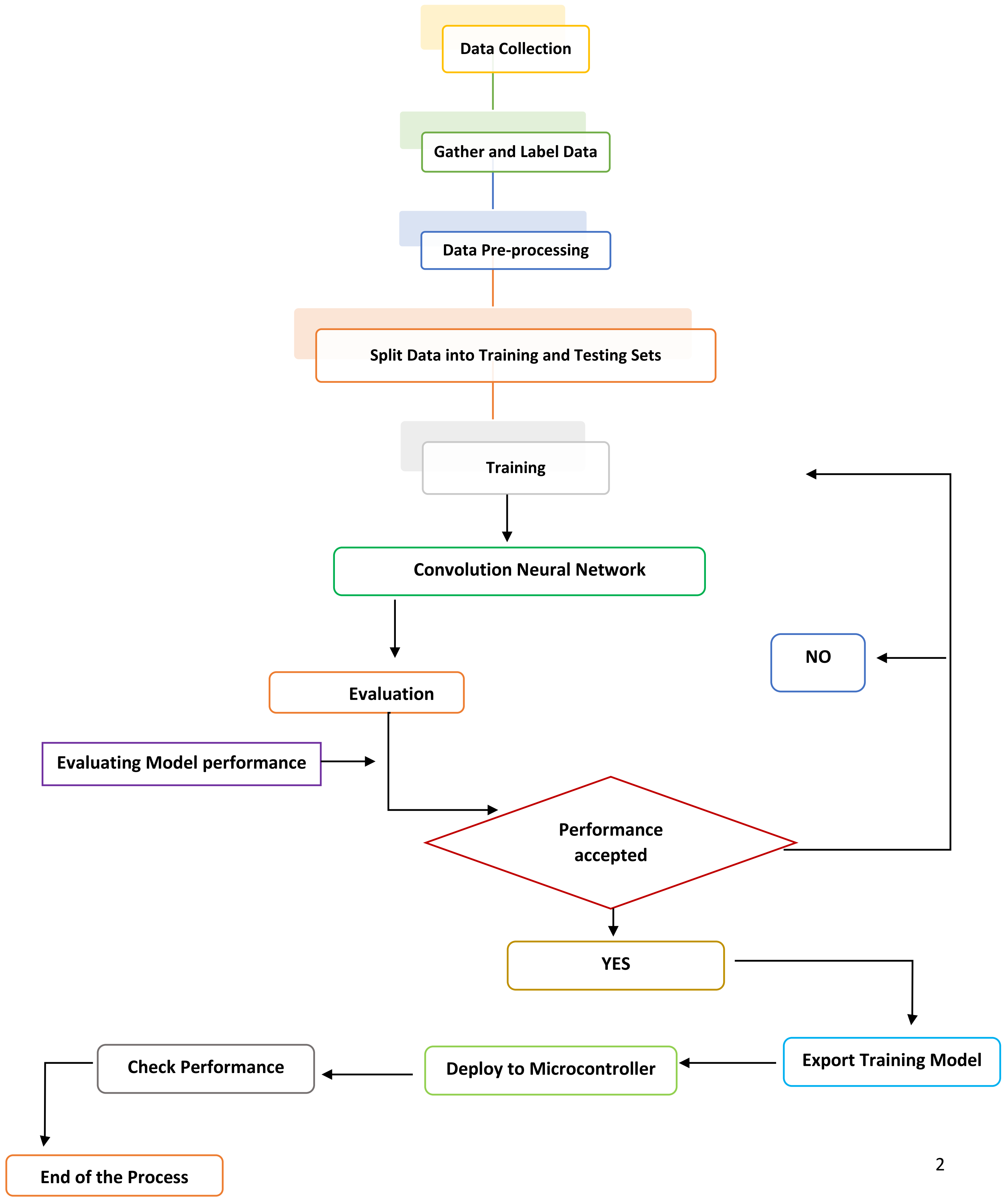
The primary purpose of this documentation is to provide a comprehensive report and documentation on what has been done to develop a wildlife monitoring tool by utilizing TinyML models.

1.1 Project Objectives

This project was guided with the following specific objectives:

i. Develop a robust TinyML application capable of accurately identifying and classifying wildlife species from camera trap images.
ii. Optimize the model to run efficiently on edge devices, ensuring real-time processing in remote field locations.
iii. Provide a user-friendly interface for researchers and conservationists to interact with the application and access valuable insights.

Figure 1: General process of the model development



2 Architecture and Design

Detailed specifications for individual components, including image preprocessing, feature extraction, and model inference, were elaborated below. Additionally, data flow diagrams were employed to visualize the movement of data within the system.

2.1 Data Preprocessing and Feature Engineering

The data preprocessing steps involved normalization, outlier detection, and augmentation techniques to enhance the quality of camera trap images. Feature extraction and engineering techniques were applied to identify relevant features, optimizing the model's ability to classify species.

2.2 Model Selection and Training

Convolutional neural network (CNN) architecture was chosen to meet the resource constraints of edge devices. To achieve high performance on image classification and recognition with a relatively modest computation cost inception v3 was used(Szegedy et al., 2016). Hyperparameters were fine-tuned, and transfer learning techniques were employed to ensure efficient model training.

On other hand, edge impulse platform was used to train another model by considering different Neural Network settings such as Training Cycle, Learning Rate, Batch size, Argumentation with various pre-built models, namely FOMO (Faster Objects, More Objects) MobileNetV2 0.35, FOMO MobileNetV2 0.1, MobileNetV2 SSD FPN-Lite, and YOLOv5.

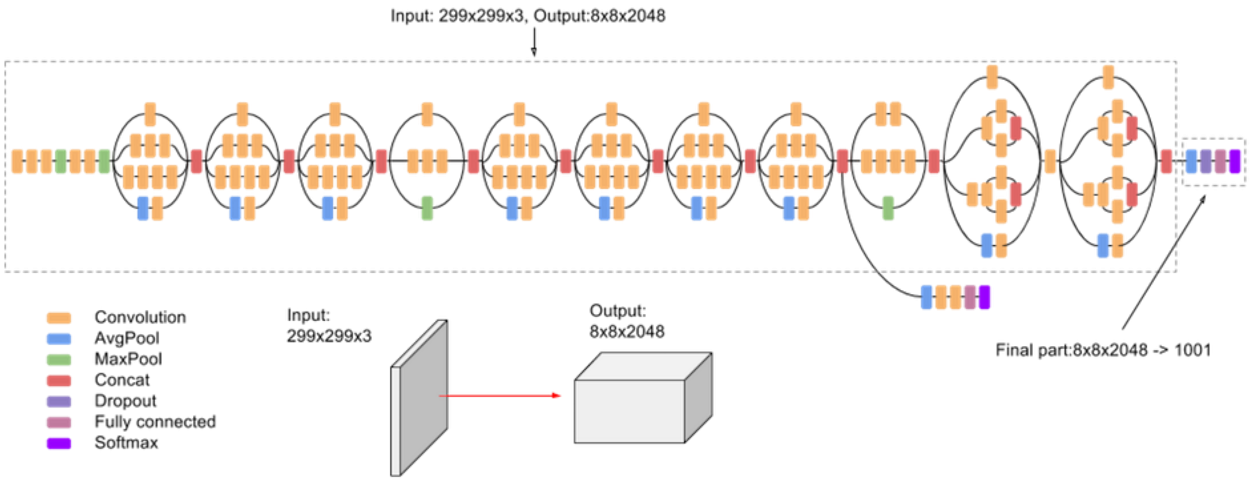


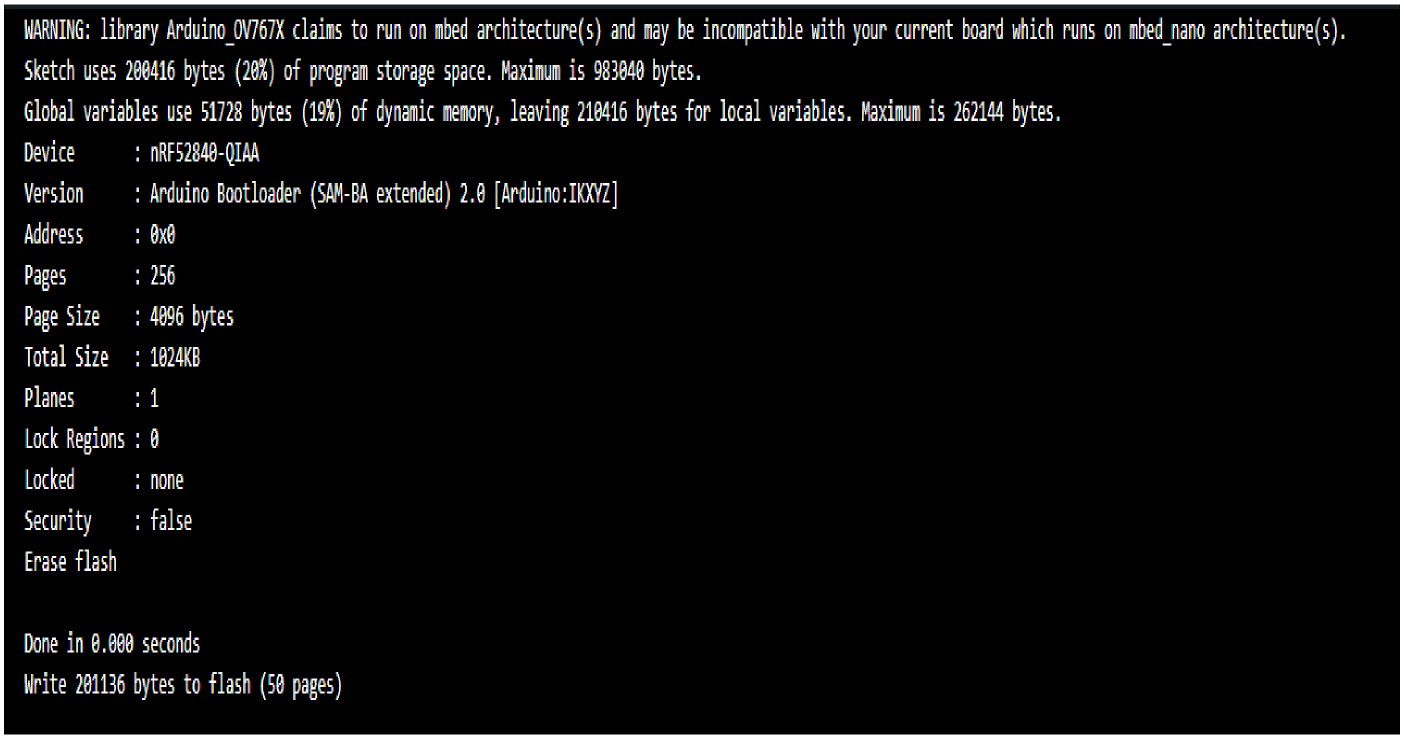
Figure 2: The architecture of CNN from cloud.google.com

2.3 Model Evaluation and Validation

The performance of the model was assessed using metrics such as accuracy, precision, and recall on test datasets. The accuracy of the model was found to be 79% on testing and 77% after deployment. For recall and precision, the model had 66%and 76% on macro average respectively during the testing.

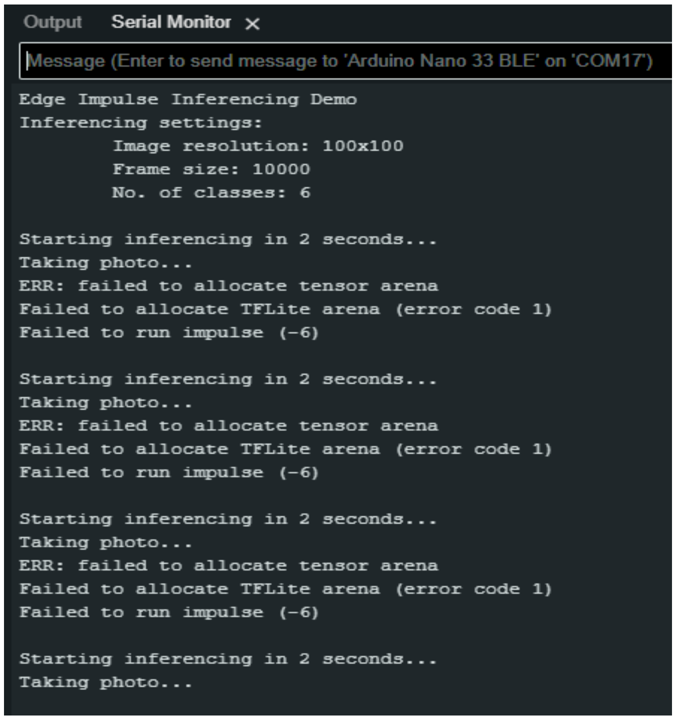
2.4 Model deployment

The model was converted and optimized using TensorFlow Lite, enabling seamless deployment on microcontrollers in remote field locations. The have successfully deployed the model in Arduino nano BLE 33 Sense via Arduino IDE and it was powered directly from the computer it was connected as can be seen in the picture below after a successful deployment of the model, the inference was performed and the following results were obtained from the deployment part, we have faced two problems.



```
WARNING: library Arduino_OV767X claims to run on mbed architecture(s) and may be incompatible with your current board which runs on mbed_nano architecture(s).
Sketch uses 200416 bytes (20%) of program storage space. Maximum is 983040 bytes.
Global variables use 51728 bytes (19%) of dynamic memory, leaving 210416 bytes for local variables. Maximum is 262144 bytes.
Device       : nRF52840-QIAA
Version      : Arduino Bootloader (SAM-BA extended) 2.0 [Arduino:IKXYZ]
Address      : 0x0
Pages        : 256
Page Size    : 4096 bytes
Total Size   : 1024KB
Planes       : 1
Lock Regions : 0
Locked       : none
Security     : false
Erase flash

Done in 0.000 seconds
Write 201136 bytes to flash (50 pages)
```


Output
Serial Monitor
Message (Enter to send message to 'Arduino Nano 33 BLE' on 'COM17')

```
Edge Impulse Inferencing Demo
Inferencing settings:
        Image resolution: 100x100
        Frame size: 10000
        No. of classes: 6

Starting inferencing in 2 seconds...
Taking photo...
ERR: failed to allocate tensor arena
Failed to allocate TFLite arena (error code 1)
Failed to run impulse (-6)

Starting inferencing in 2 seconds...
Taking photo...
ERR: failed to allocate tensor arena
Failed to allocate TFLite arena (error code 1)
Failed to run impulse (-6)

Starting inferencing in 2 seconds...
Taking photo...
ERR: failed to allocate tensor arena
Failed to allocate TFLite arena (error code 1)
Failed to run impulse (-6)

Starting inferencing in 2 seconds...
Taking photo...
```

i. The warning element due to the library that the camera has is Arduino_OV7576 but the Arduino ide has OV757X and thus claims to support Mbed_nano while the Arduino ide supports Mbed architecture

WARNING: library Arduino_OV767X claims to run on mbed architecture(s) and may be incompatible with your current board which runs on mbed_nano architecture(s).

ii. The model in inference failed to allocate the tensor arena and TFlite arena 1.1

3 Results and Discussion

3.1 Model performance

The models were trained on 10 classes that were common around the African continent: buffalo, Eland, Elephant, Giraffe, Hyaenabrown, Impala, Leopard, Lion, Rhino, Zebra. The results for the different models can be seen below. The training data consists of 56446 and Testing 14106 images. The distribution of animals in all the images is presented in Tables 1 and 2.

Table 1: Training Data

Animal Class	Count
Buffalo	2590
Elephant	5997

Rhinoceros	3268
Zebra	9564
Giraffe	4628
Impala	26616
Eland	2874
Hyaenabrown	2400
Lion	805
Leopard	923
Total	**56665**

Table 2: Testing Data

Animal Class	Count
Buffalo	647
Elephant	1499
Rhinoceros	817
Zebra	2390
Giraffe	1157
Impala	5903
Eland	718
Hyaenabrown	599
Lion	201
Leopard	230
Total	**14161**

3.2 Model Evaluation Results

The performance of the model based on Accuracy, Precision, Recall, and F1 Score is presented in the table below.

Table 3: Model performance

Classes	precision	recall	f1-score	support
0	0.84	0.23	0.36	647
1	0.97	0.36	0.52	718

2	0.87	0.53	0.66	1485
3	0.97	0.79	0.87	1124
4	0.86	0.56	0.68	599
5	0.93	0.91	0.92	5903
6	0.13	0.97	0.24	230
7	0.60	0.52	0.55	196
8	0.43	0.82	0.56	814
9	0.95	0.92	0.94	2390
Accuracy			0.78	14106
Macro avg	0.76	0.66	0.63	14106
Weighted avg	0.88	0.78	0.80	14106

3.2.1 Requirements Specification

The project was guided by requirements focused on performance metrics, scalability, portability, and ethical considerations, ensuring the tool met industry standards and ethical imperatives. In this challenge, the requirements (evaluation criteria) were fulfilled as follows:

Table 4: Model Results

Scalar	**Result**	
Accuracy	Train	After virtual deployment
	79%	**77%**
Size and Complexity	before optimization	After optimization
	270MB	**88**MB
Energy Consumption	RAM	0.020609 kWh (When RAM 3.0W)
	CPU	**0.292214** KWh
	Total CPU power	**42.5** W
	Overall energy	**0.312823** KWh
Latency	Before Deployment	After virtual deployment
	4 Second	**6 Second**

Scalability: The dataset used in developing the model had 10 species in different habitats, and pictures were captured day and night. After model deployment, a test was performed with on-screen pictures of wild animals as well as real animals to check its scalability beyond the training dataset. The model was scalable enough and was able to detect actual animals.

Robustness: The model was robust since images used in training were captured day and night in different angles and different environment

3.3 Discussion

The model development method involved the utilization of the Edge Impulse framework and Python programs. The outcomes derived from these methodologies exhibit variations in terms of precision and performance measures, owing to the disparities in the datasets employed and the neural networks utilized for training the models. The Edge Impulse model consisted of ten (10) distinct classes, containing a total of 2545 samples, as depicted in Figure 4. The dataset is divided into two subsets, with 80% of the samples allocated for training purposes and the remaining 20% reserved for testing, as illustrated in Figure 4.

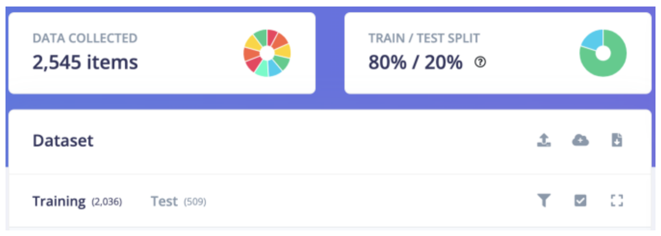
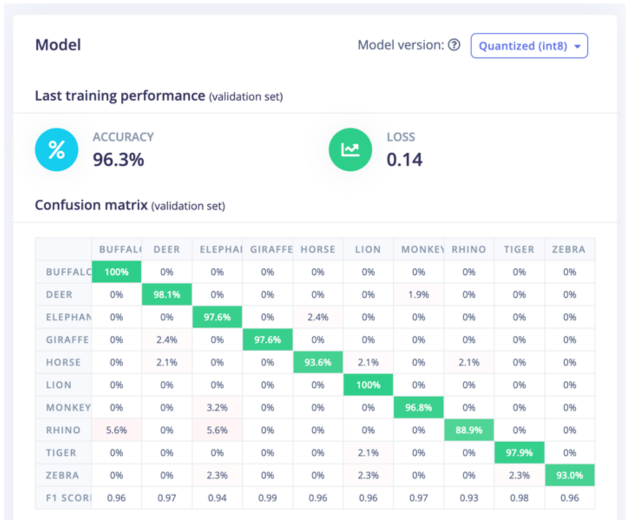

DATA COLLECTED
2,545 items
TRAIN / TEST SPLIT
80% / 20%
Dataset
Training (2,036) Test (509)

Figure 4: Data collected

We continued to use the Transfer learning and different Neural Network settings such as Training Cycle, Learning Rate, Batch size, Argumentation with various pre-built models, namely FOMO (Faster Objects, More Objects) MobileNetV2 0.35, FOMO MobileNetV2 0.1, MobileNetV2 SSD FPN-Lite, and YOLOv5. This study demonstrates disparities in accuracy between MobileNetV1 models and MobileNetV2 models, specifically in relation to Training Accuracy when comparing optimized and unquantized models. Consequently, these discrepancies also extend to the accuracies seen during deployment. The MobileNetV2 model demonstrated superior performance in both training and testing stages. However, a notable issue arose about the latency and RAM requirements for on-device performance after deploying the model. These requirements exceeded the capacity of the microcontroller, as depicted in Figure 5 and 6.



Model

Model version: Quantized (int8)

Last training performance (validation set)

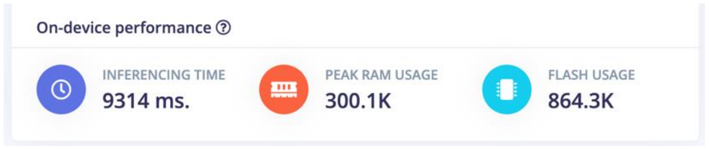
ACCURACY 96.3%

LOSS 0.14

Confusion matrix (validation set)

	BUFFAL(	DEER	ELEPHAI	GIRAFFE	HORSE	LION	MONKEY	RHINO	TIGER	ZEBRA
BUFFALC	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%
DEER	0%	98.1%	0%	0%	0%	0%	1.9%	0%	0%	0%
ELEPHAN	0%	0%	97.6%	0%	2.4%	0%	0%	0%	0%	0%
GIRAFFE	0%	2.4%	0%	97.6%	0%	0%	0%	0%	0%	0%
HORSE	0%	2.1%	0%	0%	93.6%	2.1%	0%	2.1%	0%	0%
LION	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%
MONKEY	0%	0%	3.2%	0%	0%	0%	96.8%	0%	0%	0%
RHINO	5.6%	0%	5.6%	0%	0%	0%	0%	88.9%	0%	0%
TIGER	0%	0%	0%	0%	0%	2.1%	0%	0%	97.9%	0%
ZEBRA	0%	0%	2.3%	0%	0%	2.3%	0%	0%	2.3%	93.0%
F1 SCORI	0.96	0.97	0.94	0.99	0.96	0.96	0.97	0.93	0.98	0.96

Figure 5: Training performance using MobileNetV2 model



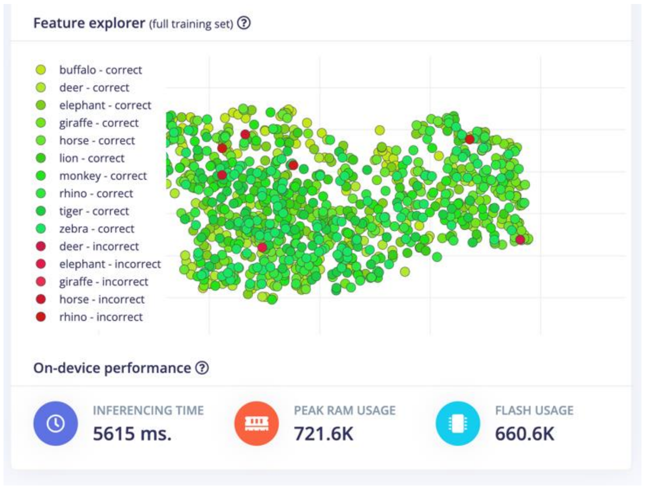
On-device performance

INFERENCING TIME 9314 ms.

PEAK RAM USAGE 300.1K

FLASH USAGE 864.3K

Figure 6: Expected on-device performance after Model deployment



Feature explorer (full training set)

On-device performance

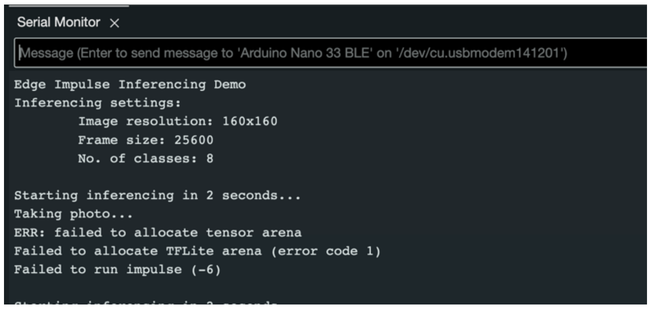
INFERENCING TIME 5615 ms.

PEAK RAM USAGE 721.6K

FLASH USAGE 660.6K

Figure 7: Expected On-device performance using MobileNetV2 model

Thereafter deployment of MobileNetV2 models on the microcontrollers, the device failed to start inferences with request of working on the memory for a Neural Network to run smoothly as shown in figure 8.

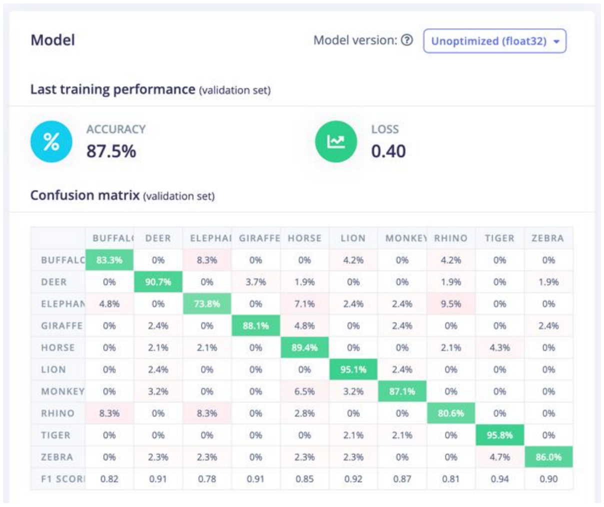

Serial Monitor
Message (Enter to send message to 'Arduino Nano 33 BLE' on '/dev/cu.usbmodem141201')

```
Edge Impulse Inferencing Demo
Inferencing settings:
        Image resolution: 160x160
        Frame size: 25600
        No. of classes: 8

Starting inferencing in 2 seconds...
Taking photo...
ERR: failed to allocate tensor arena
Failed to allocate TFLite arena (error code 1)
Failed to run impulse (-6)
```

Figure 8: Failed to allocate TFLite arena

In addition to MobileNetV2, the MobileNetV1 model exhibited strong performance during both the training and testing phases. The deployment of the model after fine-tuning it using EON Tuner yielded good outcomes in terms of latency and RAM needs, as illustrated in Figures 9 and 10. The Model was effectively implemented on a microcontroller to perform real-time inferences, exhibiting high accuracy as demonstrated in Figures 12 and 13.



Last training performance (validation set)

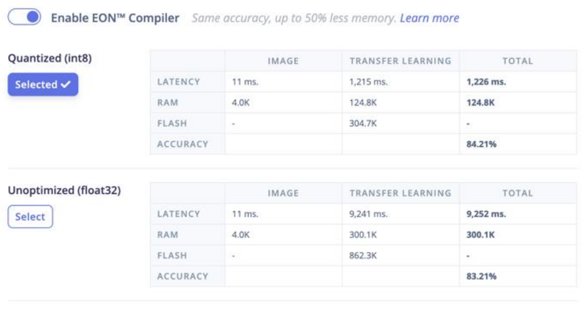
ACCURACY 87.5%

LOSS 0.40

Confusion matrix (validation set)

	BUFFAL(	DEER	ELEPHAI	GIRAFFE	HORSE	LION	MONKEY	RHINO	TIGER	ZEBRA
BUFFALC	83.3%	0%	8.3%	0%	0%	4.2%	0%	4.2%	0%	0%
DEER	0%	90.7%	0%	3.7%	1.9%	0%	0%	1.9%	0%	1.9%
ELEPHAN	4.8%	0%	73.8%	0%	7.1%	2.4%	2.4%	9.5%	0%	0%
GIRAFFE	0%	2.4%	0%	88.1%	4.8%	0%	2.4%	0%	0%	2.4%
HORSE	0%	2.1%	2.1%	0%	89.4%	0%	0%	2.1%	4.3%	0%
LION	0%	2.4%	0%	0%	0%	95.1%	2.4%	0%	0%	0%
MONKEY	0%	3.2%	0%	0%	6.5%	3.2%	87.1%	0%	0%	0%
RHINO	8.3%	0%	8.3%	0%	2.8%	0%	0%	80.6%	0%	0%
TIGER	0%	0%	0%	0%	0%	2.1%	2.1%	0%	95.8%	0%
ZEBRA	0%	2.3%	2.3%	0%	2.3%	2.3%	0%	0%	4.7%	86.0%
F1 SCORI	0.82	0.91	0.78	0.91	0.85	0.92	0.87	0.81	0.94	0.90

Figure 9: MobileNetV1 training performance results



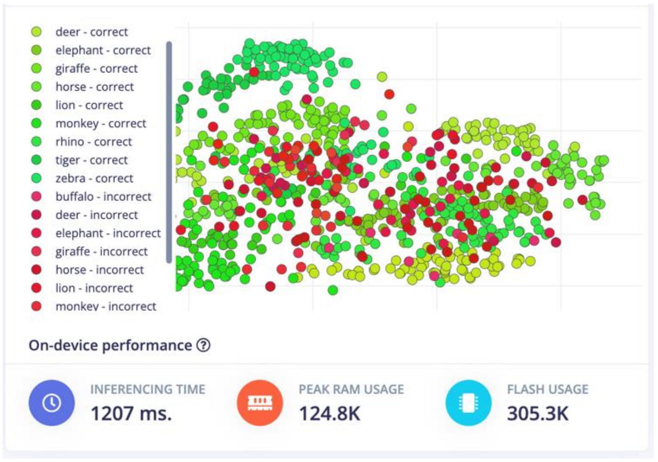
Quantized (int8) [✓ Selected]

	IMAGE	TRANSFER LEARNING	TOTAL
LATENCY	11 ms.	1,215 ms.	**1,226 ms.**
RAM	4.0K	124.8K	**124.8K**
FLASH	-	304.7K	-
ACCURACY			**84.21%**

Unoptimized (float32) Select

	IMAGE	TRANSFER LEARNING	TOTAL
LATENCY	11 ms.	9,241 ms.	**9,252 ms.**
RAM	4.0K	300.1K	**300.1K**
FLASH	-	862.3K	-
ACCURACY			**83.21%**

Figure 10: Model optimization for On-device performance.

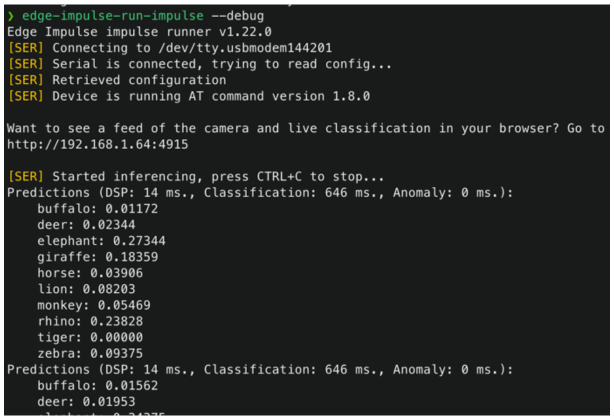


On-device performance

INFERENCING TIME	PEAK RAM USAGE	FLASH USAGE
1207 ms.	124.8K	305.3K

Figure 11: Features exploration and expected on-device performance results using MobileNetV1

The findings indicate that MobileNetV1 demonstrates favorable outcomes when implemented for doing inferences on the device. The model exhibits a high level of accuracy in accurately classifying animal species, as depicted in Figure 10. Despite the increased accuracies, MobilNetV2 has faced issues in terms of adaptation. In contrast, MobileNetV1 has demonstrated good performance in both training and testing. Furthermore, even after the deployment of the model, the inferences remain accurate.

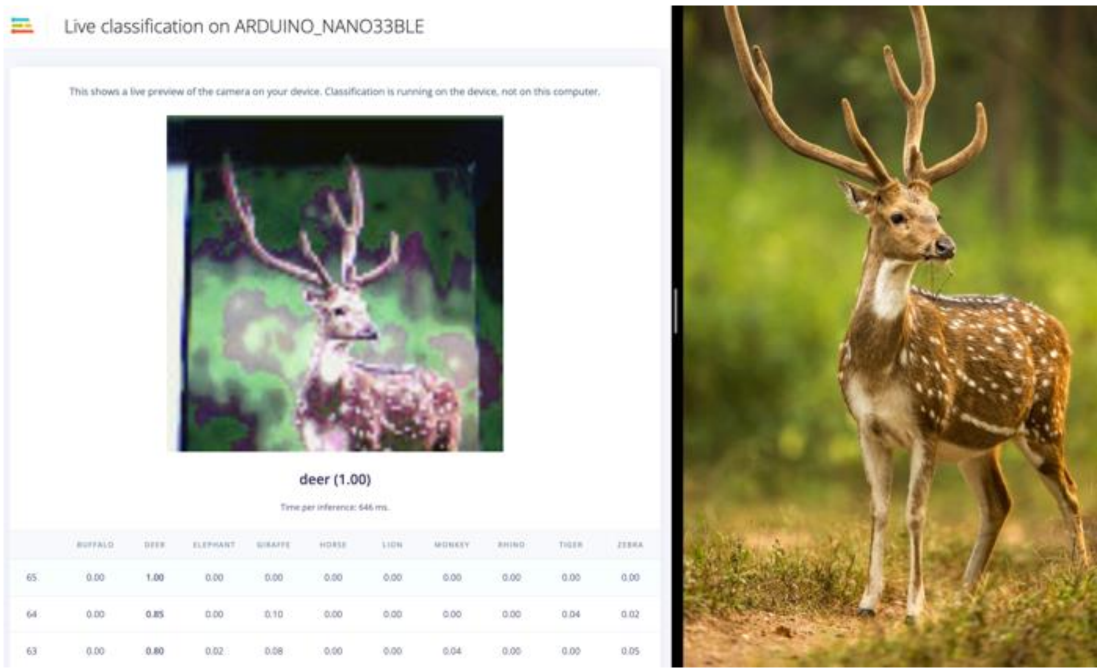


```
❯ edge-impulse-run-impulse --debug
Edge Impulse impulse runner v1.22.0
[SER] Connecting to /dev/tty.usbmodem144201
[SER] Serial is connected, trying to read config...
[SER] Retrieved configuration
[SER] Device is running AT command version 1.8.0

Want to see a feed of the camera and live classification in your browser? Go to
http://192.168.1.64:4915

[SER] Started inferencing, press CTRL+C to stop...
Predictions (DSP: 14 ms., Classification: 646 ms., Anomaly: 0 ms.):
    buffalo: 0.01172
    deer: 0.02344
    elephant: 0.27344
    giraffe: 0.18359
    horse: 0.03906
    lion: 0.08203
    monkey: 0.05469
    rhino: 0.23828
    tiger: 0.00000
    zebra: 0.09375
Predictions (DSP: 14 ms., Classification: 646 ms., Anomaly: 0 ms.):
    buffalo: 0.01562
    deer: 0.01953
```

Figure 12: Inferences from a device displayed on the terminal



Live classification on ARDUINO_NANO33BLE

This shows a live preview of the camera on your device. Classification is running on the device, not on this computer.

deer (1.00)

Time per inference: 646 ms.

	BUFFALO	DEER	ELEPHANT	GIRAFFE	HORSE	LION	MONKEY	RHINO	TIGER	ZEBRA
65	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
64	0.00	**0.85**	0.00	0.10	0.00	0.00	0.00	0.00	0.04	0.02
63	0.00	**0.80**	0.02	0.08	0.00	0.00	0.04	0.00	0.00	0.05

Figure 13: Live classification on the device and displayed on web API

4 User Documentation

4.1 Introduction

The Wildlife Monitoring Tool, designed for microcontroller deployment, aims to automate wildlife monitoring processes while operating under resource constraints. This guide details the user documentation, emphasizing installation, interaction, and best practices for users of the microcontroller version to provide comprehensive guidance for end-users, enabling them to interact effectively with the Tiny Machine Learning (TinyML) application in the unique context of microcontroller-based deployments.

4.2 User Documentation

4.2.1 Installation and Setup

The user documentation offers step-by-step instructions to guide users through the installation and setup process. This includes downloading and configuring the application for the specific microcontroller used.

System Requirements

To ensure optimal performance of the Wildlife Monitoring Tool on the microcontroller, please verify the following:

- Microcontroller model and specifications (**Arduino Nano 33 BLE Sense light**).
- Power supply for the microcontroller.
- Stable internet connection (if applicable for remote deployment).

Microcontroller Compatibility

The Wildlife Monitoring Tool is compatible with the microcontroller with the following pieces of equipment:

1. TinyML Microntroller (Arduino Nano 33 BLE Sense)
2. Camera (OV7675 Camera module)
3. USB A Micro USB Cable

Firmware Installation

Follow these step-by-step instructions to install the Wildlife Monitoring Tool on your microcontroller:

- Connect the microcontroller to a power source and ensure it is properly powered on.
- Insert the provided installation media (Arduino IDE) into the microcontroller's designated slot.

- Use the microcontroller's interface to navigate to the "Installation" section.
- Select the option to install the Wildlife Monitoring Tool from the provided media.
- Follow the on-screen prompts to complete the installation process

1.1.1 Launching the Application

Powering On the Microcontroller

To power on the microcontroller, follow these steps:

- Connect the microcontroller to a power source using the provided power cable.
- Press and hold the power button until the device powers on.
- Wait for the microcontroller to boot up and display the main interface.

Navigating to the Application

Once the microcontroller has powered on, access the application interface using the provided method (connecting via USB). Follow the specific instructions provided for your chosen microcontroller model. Use the navigation controls on the microcontroller to access the "Applications" or "Tools" menu. Locate and select the "Wildlife Monitoring Tool" from the list of available applications.

4.2.2 User Interface

The interface consists of the following components:

Menu Bar: Contains options for uploading images, adjusting settings, and accessing results.

Image Upload Button: This allows you to upload camera trap images for analysis.

Result Display Area: Displays the species classification results.

Navigating Menus and Options

To navigate menus and options, use the directional buttons on the microcontroller:

Use the "Up" and "Down" buttons to navigate through menu options.

Press the "Enter" or "Select" button to choose an option.

Use the "Back" or "Exit" button to return to the previous menu.

4.2.3 Core Functionality

This section of the user documentation focuses on providing users with a clear understanding of how to use the primary features and functionalities of the application. The Core Functionality section serves as a comprehensive guide to the key features and operations of the Wildlife Monitoring Tool on the microcontroller. It aims to ensure users have a solid grasp of how to perform critical tasks that are central to the tool's purpose.

Capturing Camera Trap Images

One of the pivotal functions of the Wildlife Monitoring Tool is capturing images from camera traps. This process is crucial for obtaining visual data on wildlife in their natural habitat. Users can follow the steps below:

1. Power on the Camera Traps: Ensure that the camera traps are powered and functioning properly.

2. Positioning of Camera Traps: Strategically place the camera traps in locations where wildlife activity is anticipated.

3. Adjusting Settings (if applicable): Configure the camera traps based on environmental conditions, such as sensitivity, motion detection, and image resolution.

4. Activating Camera Traps: Start the camera traps to begin capturing images.

Processing Captured Images

Once images are captured, the next core functionality is the processing of these images using the microcontroller-based application. This step is essential for species identification and data analysis, a user can follow the following steps.

1. Uploading Images: Use the provided interface to upload the captured images to the microcontroller.

2. Initiating Species Identification: Start the identification process to analyze the uploaded images.

Reviewing Species Classification

After the identification process is complete, users will need to review the species classification results. This step below allows users to validate and make sense of the identified species.

1. Navigating Through Results: Use the provided controls to navigate through the list of species classifications.

2. Analyzing Confidence Scores: Pay attention to the confidence scores associated with each identified species.

The Core Functionality section ensures that users have a solid understanding of the essential operations of the Wildlife Monitoring Tool. By following these detailed instructions, users can effectively capture, process, and review camera trap images, ultimately contributing to wildlife conservation efforts.

4.2.4 Interpreting Results

After the identification process is complete, you will see the species classification results on the screen. Each result will display the identified species along with a confidence score.

i. Use the "Next" or "Previous" buttons to navigate through the results.
ii. Take note of the species names and confidence scores for further analysis.

4.2.5 Customization (if applicable)

If the Wildlife Monitoring Tool allows for customization, you can adjust settings to suit your specific requirements. Navigate to the "Settings" or "Preferences" menu in the tool's interface. Use the provided options to customize settings such as image resolution, analysis threshold, or data storage preferences.

4.2.6 Saving and Exporting Data (if applicable)

To save the results on the microcontroller for future reference, follow these steps:

- Navigate to the "Save Results" option in the tool's interface.
- Choose a location on the microcontroller to store the results and confirm the save.

If you need to further analyze the data on a separate device, follow these steps to export it:

- Connect a storage device (e.g., USB drive) to the microcontroller.
- Navigate to the "Export Data" option in the tool's interface.
- Select the data you want to export and choose the destination on the connected storage device.

This User Documentation aims to empower users in effectively utilizing the Wildlife Monitoring Tool on your microcontroller. Your contributions to wildlife conservation efforts are invaluable. If you have any further questions or need assistance, do not hesitate to reach out to our dedicated support team.

5 Testing and Quality Assurance

The Testing and Quality Assurance phase in the development of the Wildlife Monitoring Tool was a process aimed at ensuring the reliability, functionality, and performance of the application. This crucial stage involved a series of systematic tests and checks to identify and address any potential issues or discrepancies. The goal was to deliver a robust and dependable tool for wildlife conservation efforts.

5.1 Testing Methodologies

5.1.1 Unit Tests

Unit testing was employed to examine individual components or functions of the application in isolation. This rigorous approach allowed us to verify that each component operated as intended and produced the correct output given specific inputs. The process was as follows:

- Implementation: Our development team carefully created test cases for each function or module, covering a wide range of scenarios and edge cases.
- Purpose: This phase was essential in ensuring that each building block of the application functioned flawlessly.

5.1.2 Integration Tests

Integration testing was instrumental in evaluating the interactions and interfaces between different components of the application. By conducting these tests, we confirmed that integrated components worked together seamlessly. The process was as follows:

- Implementation: Our testing team designed test cases to validate the flow of data and interactions between integrated modules.
- Purpose: This phase was crucial in guaranteeing that integrated components functioned cohesively.

5.1.3 System Tests

System testing involves the evaluation of the application as a whole, simulating a real-world environment. Through these comprehensive tests, we ensured that the application behaved as expected and fulfilled its intended purpose. The process was as follows:

- Implementation: Our testing team executed detailed test scenarios, replicating end-user interactions and workflows.

- Purpose: This phase provided validation that the entire application met the specified requirements and operated seamlessly.

5.2 Bug Reports and Test Results

5.2.1 Bug Reports

Throughout the testing phase, our team diligently documented any identified issues, anomalies, or deviations from expected behavior in detailed bug reports. Each report included critical information:

- Description of the Issue: We provided a clear and concise description of the problem, ensuring all people understood the nature of the issue.
- Steps to Reproduce: We outlined detailed steps to recreate the issue, facilitating efficient debugging and resolution.
- Impact: We assessed the impact of the issue on the application, prioritizing critical fixes for immediate attention.

5.2.2 Test Results

Our team maintained a comprehensive record of test results, offering a clear overview of the application's performance throughout the testing phase. These results served as a vital reference point for tracking progress and identifying areas for improvement.

- Passed: Test cases that produced the expected results were marked as 'Passed', indicating that the application functioned as intended.
- Failed: Test cases that did not produce the expected results were marked as 'Failed', signalling a potential issue that required further investigation and resolution.

The Testing and Quality Assurance phase played a pivotal role in ensuring the reliability and performance of the Wildlife Monitoring Tool. Through a combination of unit, integration, and system tests, the application underwent rigorous evaluation. Identified issues were meticulously documented in bug reports, and test results provided a clear overview of the application's performance. This meticulous testing process was essential in delivering a robust and dependable tool for wildlife conservation efforts.

6 Version Control and Change Log

Version Control and maintaining a Change Log were crucial components of the development process for the Wildlife Monitoring Tool. These practices ensured efficient collaboration, code management, and transparency in tracking the evolution of the application. This section provides an in-depth look at how version control and change logs were implemented in the project.

6.1 Version Control System (VCS)

6.1.1 Git Repository Setup

The project utilized Git as the Version Control System (VCS) to track changes, manage branches, and facilitate collaborative development. The following steps were taken to set up the Git repository:

1. Initialization: A new Git repository was created to house the project codebase.

2. Branching Strategy: We established a branching strategy, utilizing branches for features, bug fixes, and release versions.

3. Remote Repositories: The repository was linked to a remote Git server (e.g., GitHub, GitLab) to enable seamless collaboration among team members.

6.1.2 Branch Management

Effective branch management was crucial in maintaining a structured development process. The following practices were implemented:

- Feature Branches: Each new feature or enhancement was developed on a separate branch to isolate changes and enable parallel development.
- Bug Fix Branches: Bug fixes were implemented on dedicated branches to ensure they could be independently verified and merged.
- Release Branches: Release branches were created for versioning milestones, allowing for stable snapshots of the application.

6.1.3 Collaborative Workflow

Collaboration was streamlined through Git workflows:

- Pull Requests: Changes were proposed and reviewed through pull requests, enabling thorough code review before merging into the main branch.
- Code Reviews: Team members conducted code reviews to ensure code quality, adherence to coding standards, and identification of potential issues.

6.2 Change Log

6.2.1 Change Log Structure

The Change Log served as a detailed record of revisions, updates, and bug fixes. It followed a standardized format, including the following information for each change:

- Version Number: Clearly defined version numbers (e.g., Semantic Versioning) for easy tracking.
- Date of Change: Timestamp indicating when the change was implemented.
- Summary of Change: A concise description of the change, providing context to stakeholders.

6.2.2 Updating the Change Log

The Change Log was diligently maintained throughout the development process. The steps taken included:

1. Commit Messages: Developers provided clear and descriptive commit messages, summarizing changes made.
2. Aggregated Updates: Changes from individual commits were aggregated into meaningful entries for the Change Log.
3. Categorization: Changes were categorized into features, bug fixes, enhancements, and other relevant classifications.

6.2.3 Release Notes

For major releases, comprehensive release notes were generated. These included:

- New Features: Descriptions of newly introduced features or functionalities.
- Bug Fixes: Details on resolved issues or bugs.
- Enhancements: Improvements made to existing features.
- Deprecations: Notifications of any deprecated features.

Version Control and Change Log practices were instrumental in maintaining a structured and transparent development process for the Wildlife Monitoring Tool. Git facilitated efficient collaboration, while the Change Log provided a detailed history of revisions. These practices ensured code integrity, streamlined development workflows, and enhanced communication among team members.

7 Acknowledgements and Contributors

The Acknowledgments and Contributors section is a fundamental part of recognizing and appreciating the contributions made by individuals and organizations towards the development of the Wildlife Monitoring Tool. This section provides a detailed account of how acknowledgements were curated and contributors were recognized in the project. It is in that direction; the team acknowledge the following:

Artificial Intelligence for Development (AI4D) for providing their technical guidance and mentorship, and the University of Dodoma for providing space and utilities such as Internet, electricity and water to members working from the University of Dodoma.

ITU and ICTP for providing micro-controller which has enabled the time to have real deployment in the field.

7.1 Acknowledgements

7.1.1 Gratitude to Contributors

This section expresses gratitude to organizations and entities that have made significant contributions to the development process. Within this segment, we take the opportunity to recognize and appreciate the invaluable support extended by these parties. Their dedication and collaboration have played a crucial role in shaping and advancing the project, and their contributions are sincerely acknowledged and appreciated.

7.1.2 Categories of Acknowledgments

1. Development Team: The core development team members were acknowledged for their tireless efforts, expertise, and dedication in bringing the Wildlife Monitoring Tool to fruition.

2. Challenge organizer: Acknowledgments were extended to the organizer of the challenge who managed to provide the Arduino tool kit to facilitate the deployment of the model in real-time.

7.2 Contributors

7.2.1 Identification of Contributors

The Contributors section was carefully curated to provide a comprehensive list of individuals and organizations who actively contributed to the project. Contributions encompassed a wide range of activities including code development, testing, documentation, and domain expertise.

7.2.2 Categories of Contributions

i. Code Contributions: Individuals who actively contributed to the codebase, either through feature development, bug fixes, or enhancements, were recognized for their technical expertise.

ii. Testing and Quality Assurance: Testers and QA experts who rigorously evaluated the application, identified bugs, and ensured its reliability were acknowledged for their vital role in maintaining quality.

iii. Documentation: Contributors who played a pivotal role in creating and maintaining documentation, ensuring that users had clear and comprehensive resources for utilizing the tool, were recognized.

iv. Mentorship: Artificial Intelligence for Development for providing technical guidance and mentorship throughout the project

v. Space and utilities: The University of Dodoma provides space and utilities such as Internet, electricity and water to members working from the University of Dodoma.

The Acknowledgments and Contributors section served as a tribute to the collaborative spirit and collective effort that propelled the development of the Wildlife Monitoring Tool. By recognizing the diverse contributions of individuals and organizations, we celebrated the collective achievement and underscored the importance of community-driven initiatives in the field of wildlife conservation. This section stands as a testament to the collaborative ethos that defined the project.