

**NANYANG  
TECHNOLOGICAL  
UNIVERSITY**

---

**SINGAPORE**

**CE4172: Internet of Things – Tiny Machine Learning  
Project Report**

CHEW LOH SENG

U1921147L

## Introduction

Using capacitive touch and gestures recorded by the Arduino Nano 33 BLE Sense, controls of a connected Raspberry Pi 4 and computer were explored. An RGB LED is attached to the Arduino for debugging purposes, ensuring correct controls were transmitted to the Pi for processing.

Hardware used for the project:

- Arduino Nano 33 Sense BLE (“Arduino”)
- Raspberry Pi 4 Model B 4GB (“Raspberry Pi” or “Pi”)
- Capacitive Touch Sensor
- RGB LED
- Dell Latitude 7490 with Wake-on-LAN capability enabled

## Features

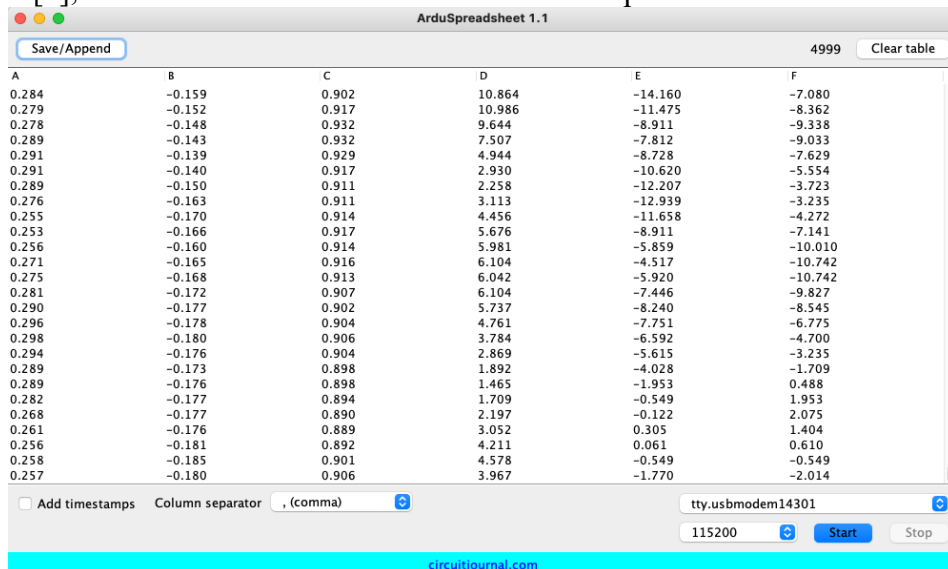
In this project, the following features were implemented:

- Touching the capacitive touch sensor – Turning on the connected computer
- Lifting the Arduino – Turning on the connected computer
- Twisting motion on the Arduino – Reboot the Raspberry Pi
- TensorFlow Lite Libraries failed to function properly – Shut down the Raspberry Pi

## Data Collection and Training Process

This project builds upon the Gesture Recognition Lecture conducted in Week 7, modifying the gesture detection from punching and flexing to lifting and twisting. The motions were selected to have reduced ambiguity between the motions, and have the priority given to turning on the connected Computer instead of turning off the Raspberry Pi.

At least 40 repeated actions were made for each gesture with slight variations to reduce the chance of overfitting. All the actions were recorded using the ArduSpreadsheet add-on for Arduino IDE [1], with the comma selected as the column separator.



A	B	C	D	E	F
0.284	-0.159	0.902	10.864	-14.160	-7.080
0.279	-0.152	0.917	10.986	-11.475	-8.362
0.278	-0.148	0.932	9.644	-8.911	-9.338
0.289	-0.143	0.932	7.507	-7.812	-9.033
0.291	-0.139	0.929	4.944	-8.728	-7.629
0.291	-0.140	0.917	2.930	-10.620	-5.554
0.289	-0.150	0.911	2.258	-12.207	-3.723
0.276	-0.163	0.911	3.113	-12.939	-3.235
0.255	-0.170	0.914	4.456	-11.658	-4.272
0.253	-0.166	0.917	5.676	-8.911	-7.141
0.256	-0.160	0.914	5.981	-5.859	-10.010
0.271	-0.165	0.916	6.104	-4.517	-10.742
0.275	-0.168	0.913	6.042	-5.920	-10.742
0.281	-0.172	0.907	6.104	-7.446	-9.827
0.290	-0.177	0.902	5.737	-8.240	-8.545
0.296	-0.178	0.904	4.761	-7.751	-6.775
0.298	-0.180	0.906	3.784	-6.592	-4.700
0.294	-0.176	0.904	2.869	-5.615	-3.235
0.289	-0.173	0.898	1.892	-4.028	-1.709
0.289	-0.176	0.898	1.465	-1.953	0.488
0.282	-0.177	0.894	1.709	-0.549	1.953
0.268	-0.177	0.890	2.197	-0.122	2.075
0.261	-0.176	0.889	3.052	0.305	1.404
0.256	-0.181	0.892	4.211	0.061	0.610
0.258	-0.185	0.901	4.578	-0.549	-0.549
0.257	-0.180	0.906	3.967	-1.770	-2.014

Figure 1: Example output recorded by ArduSpreadsheet

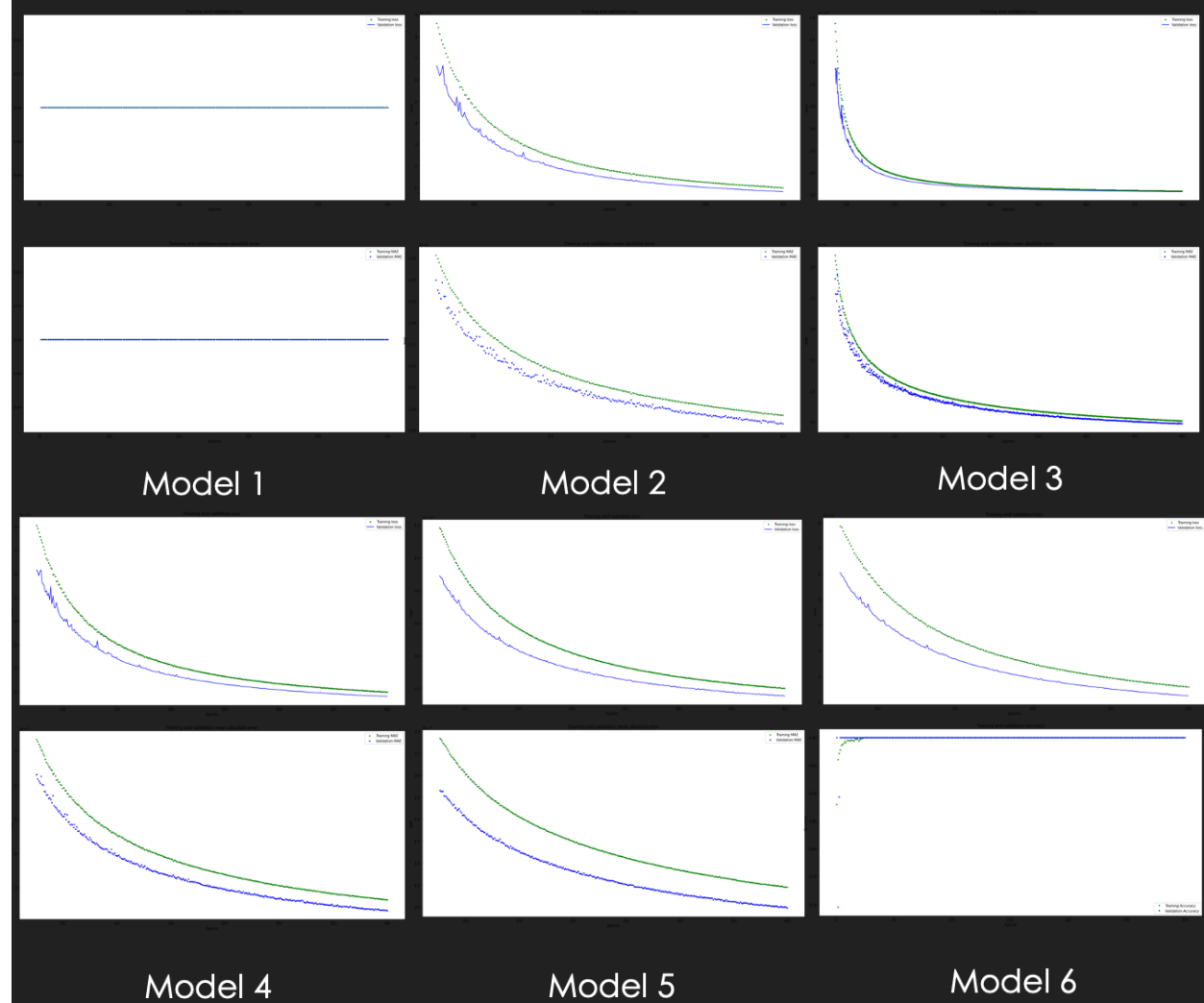
Using the data collected, the training was done using Google Colab. The methods deployed are similar to the ones taught in the course. 8 models were developed, with the new model built upon the results obtained from the previous models.

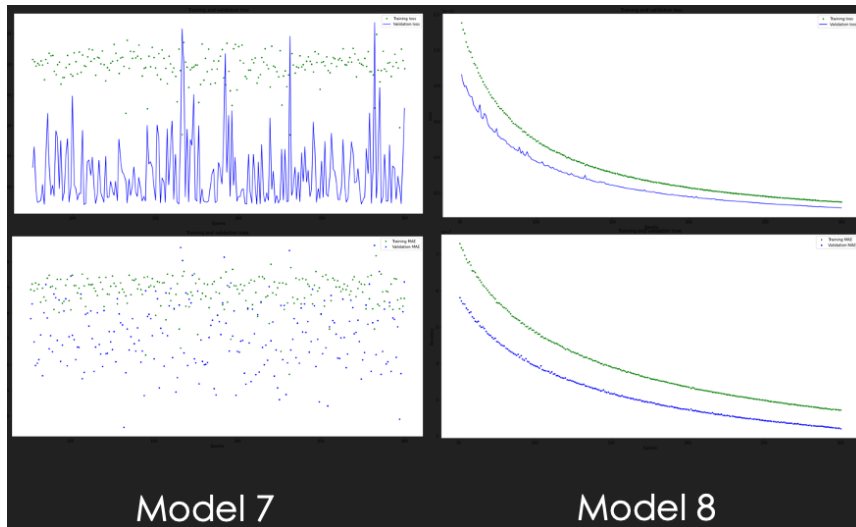
The table below describes the models and their results

Model	Details of the Model	Results	Commentary
1	<ul style="list-style-type: none"> <li>1 Hidden Layer (ReLU) with 30 neurons + Output Layer (ReLU) with 2 neurons (for both gestures)</li> <li>Epoch: 300. Batch Size: 5.</li> <li>Optimizer: RMSProp. Loss: MSE. Metric: MAE.</li> </ul>	Training loss: 0.5000 Training MAE: 0.5000 Validation loss: 0.5000 Validation MAE: 0.5000	This model set as a baseline.  All the loss and MAE recorded were 0.5000 throughout the training, and no improvement was observed throughout. This might be due to the activation function of ReLU, especially for the output.
2	<ul style="list-style-type: none"> <li>1 Hidden Layer (ReLU) with 30 neurons + Output Layer (SoftMax) with 2 neurons</li> <li>Epoch: 300. Batch Size: 5.</li> <li>Optimizer: RMSProp. Loss: MSE. Metric: MAE.</li> </ul>	Training loss: $9.9932 \times 10^{-12}$ Training MAE: $9.2803 \times 10^{-7}$ Validation loss: $8.0361 \times 10^{-12}$ Validation MAE: $8.2750 \times 10^{-7}$	Output activation function was replaced with SoftMax.  As expected, error rates detected declined significantly. The loss and MAE calculated was declining throughout the epoch counts.
3	<ul style="list-style-type: none"> <li>1 Hidden Layer (ReLU) with 30 neurons + Output Layer (SoftMax) with 2 neurons</li> <li>Epoch: 800. Batch Size: 5.</li> <li>Optimizer: RMSProp. Loss: MSE. Metric: MAE.</li> </ul>	Training loss: $3.3942 \times 10^{-12}$ Training MAE: $5.2333 \times 10^{-7}$ Validation loss: $2.7736 \times 10^{-12}$ Validation MAE: $4.7484 \times 10^{-7}$	Epoch count was increased to 800 to monitor the trend of loss and MAE.  While the downward trend continues, the rate of change declined significantly after around epoch=300. Also, the training period increased significantly as epoch count increased.
4	<ul style="list-style-type: none"> <li>2 Hidden Layers (ReLU) with 30 neurons + Output Layer (SoftMax) with 2 neurons</li> <li>Epoch: 400. Batch Size: 5.</li> <li>Optimizer: RMSProp. Loss: MSE. Metric: MAE.</li> </ul>	Training loss: $9.7787 \times 10^{-13}$ Training MAE: $2.6364 \times 10^{-7}$ Validation loss: $7.8363 \times 10^{-13}$ Validation MAE: $2.3307 \times 10^{-7}$	Hidden Layer count increased from 1 to 2. Epoch count was reduced to 400 based on results of Model 3.  Loss and MAE continues to decline, even when epoch count was cut by half. This shows that increasing the number of layers is likely to improve accuracy.
5	<ul style="list-style-type: none"> <li>2 Hidden Layers (ReLU) with 30 neurons + Output Layer (SoftMax) with 2 neurons</li> <li>Epoch: 400. Batch Size: 1.</li> <li>Optimizer: RMSProp. Loss: MSE. Metric: MAE.</li> </ul>	Training loss: $2.0243 \times 10^{-13}$ Training MAE: $9.8161 \times 10^{-8}$ Validation loss: $1.5453 \times 10^{-13}$ Validation MAE: $7.9075 \times 10^{-8}$	Batch size reduced from 5 to 1.  Significant improvements in all measurements, but the training time increased significantly, from approximately 2-4 sec per epoch to 11-14 sec per epoch.  However, even with batch size of 5, the loss and MAE is already very low.
6	<ul style="list-style-type: none"> <li>2 Hidden Layers (ReLU) with 30 neurons + Output Layer (SoftMax) with 2 neurons</li> <li>Epoch: 300. Batch Size: 5.</li> <li>Optimizer: RMSProp. Loss: MSE. Metric: Accuracy.</li> </ul>	Training loss: $1.5844 \times 10^{-12}$ Training accuracy: 1.0000 Validation loss: $1.2311 \times 10^{-12}$ Validation accuracy: 1.0000	[Modified from <b>Model 4</b> ] Changed metric from MAE to Accuracy, and reduced epoch count to 300.  Accuracy reached 1.0000 for both training and validation by epoch 30. Both losses are slightly higher than Model 4. Accuracy is less likely to be a good metric.

7	<ul style="list-style-type: none"> <li>2 Hidden Layers (ReLU) with 30 neurons + Output Layer (SoftMax) with 2 neurons</li> <li>Epoch: 300. Batch Size: 5.</li> <li>Optimizer: Adam. Loss: MSE. Metric: MAE.</li> </ul>	Training loss: 0.2500 Training MAE: 0.4999 Validation loss: 0.2499 Validation MAE: 0.4999	[Modified from <b>Model 4</b> ] Changed optimizer from RMSProp to Adam, and reduced epoch count to 300.  Unknown reason for the high loss and MAE. That happened after epoch 13. Repeated the training twice, no difference measured. Therefore, Adam is not a good optimizer for this application.
8	<ul style="list-style-type: none"> <li>2 Hidden Layers (ReLU) with 60 neurons + Output Layer (SoftMax) with 2 neurons</li> <li>Epoch: 300. Batch Size: 5.</li> <li>Optimizer: Adams. Loss: MSE. Metric: Accuracy.</li> </ul>	Training loss: $1.4724 \times 10^{-12}$ Training MAE: $2.7078 \times 10^{-7}$ Validation loss: $1.1354 \times 10^{-12}$ Validation MAE: $2.1939 \times 10^{-7}$	[Modified from <b>Model 4</b> ] Increased neuron count from 30 to 60 for each layer, and reduced epoch count to 300.  However, the increase in neuron count have minimal effects on the results, but the model file size increased significantly compared to Model 4.

After adding in skipping of first 50-75 epochs, the output graphs of the results are as follows:





After evaluating the results above, the observations were made:

- Typically, an increase in hidden layers resulted in improved accuracy. However, the improvement might not be significant if the results are already quite accurate.
- With different activation, optimizer, loss, and metric functions, the results can differ significantly. Therefore, experiments using as many of them as possible are recommended.
- Balancing between raw results and complexity is required. This is especially true for applications in TinyML.

Therefore, Model 4 was chosen as it combines relatively good results with decent complexity. However, Model 5 would be a better choice if time and computation power is not a concern. Deploying the model with other codes included, the file size is approximately 26% and the RAM usage is approximately 32%, as shown below:

```
Library Arduino_TensorFlowLite has been declared precompiled:
Using precompiled library in /Users/lohseng97/Documents/Arduino/libraries/Arduino_TensorFlowLite/src/cortex-m4/fpv4-sp-d16-softfp
Sketch uses 261160 bytes (26%) of program storage space. Maximum is 983040 bytes.
Global variables use 85624 bytes (32%) of dynamic memory, leaving 176520 bytes for local variables. Maximum is 262144 bytes.
Write 261168 bytes to flash (64 pages)
[=====] 100% (64/64 pages)
Done in 10.153 seconds
```

## Communicating between the devices

Serial communication was created through the micro-USB cable, allowing the Arduino and Raspberry Pi to communicate with each other. While bidirectional serial communication is possible, the current setup only requires the Arduino to send commands to the Pi.

Node-RED [2] and etherwake [3] were installed in Raspberry Pi OS to allow Command controls and Wake-on-LAN functionality. Node-RED takes in and parses the messages transmitted from Arduino, and checks for appropriate “messages”.

Upon receiving the “<turnon>”, “<shutdown>”, or “<reboot>” messages, Node-RED would execute the appropriate shell scripts (specifically bash shell) to turn on the connected computer

through Magic Packet, shut down the Pi, or reboot the Pi respectively. For all other messages, the Node-RED would just print the message in the debugger and not execute any instructions.

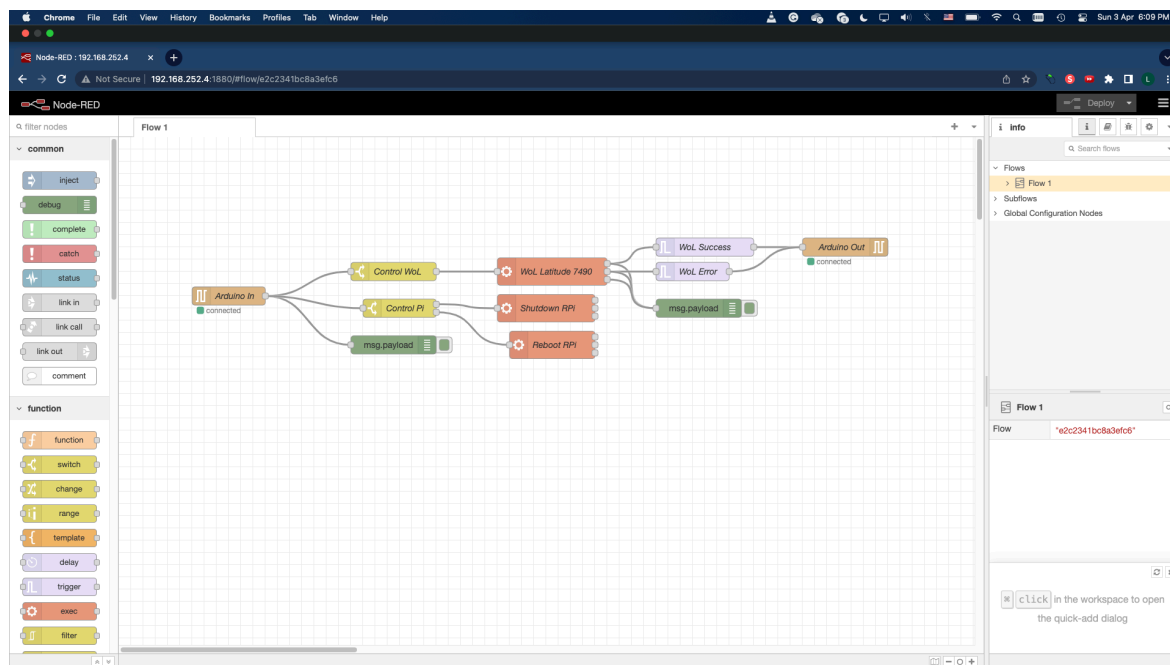


Figure 2: Node-RED Configuration for deployment in this Project

As Node-RED does not operate with “root” privileges, “chmod 777” (i.e. enable RWX for all users) has to be implemented for all shell scripts. As this could pose serious security threats, it would be wise if the “chmod 711” (i.e. only execute right is allowed for group and others) is implemented in a commercial setting.

```
pi@raspberrypi:~ $ ls *.* -l
-rw-r--r-- 1 pi pi 5901 Apr  3 18:09 flows.json
-rwxrwxrwx 1 pi pi  40 Apr  1 21:49 reboot.sh
-rwxrwxrwx 1 pi pi  51 Apr  1 17:31 shutdown.sh
-rwxrwxrwx 1 pi pi  69 Apr  1 17:31 wol.sh
```

Figure 3: Files used for this project and its read, write, and execute access rights.

GNU nano 5.4	GNU nano 5.4	GNU nano 5.4
<code>#!/bin/sh</code>	<code>#!/bin/sh</code>	<code>#!/bin/bash</code>
<code>echo "Reboot now"</code>	<code>echo "Shutdown now"</code>	<code>sudo etherwake -i eth0 8C:EC: [redacted]</code>
<code>sudo reboot</code>	<code>sudo shutdown -h now</code>	<code>echo "Success!"</code>

Figure 4 to 6: Bash shell scripts used for "<reboot>", "<shutdown>", and "<turnon>" commands respectively.

However, as Wake-on-LAN (“WoL”) requires the exact MAC address of the connected computer, the user has to modify the “wol.sh” shell script with “sudo” access or with the “root” account.

This setup also assumes that the user’s computer has an Ethernet port with WoL functionality enabled in the BIOS/UEFI and the Operating System. It also requires the Ethernet port to not be disabled when the computer is powered off, which can happen if the system reduces into hybrid shutdown mode. Note that the hybrid shutdown (Fast Startup) option is enabled by default in Windows 10 and later [4].

## Inference performance

As this uses a one-hot encoding approach, whenever an inference was made, the probability of both actions should add up to 1. To reduce the chance of false positives, a minimum of 99% probability was added for the action to be deemed accurate.

Typically, the inference duration is approximately 4-5ms, for both Twisting and Lifting motions.

Twisting: 0.003906	Twisting: 0.996094
LiftUpDown: 0.996094	LiftUpDown: 0.000000
Inference Duration = 5	Inference Duration = 4
<turnon>	<reboot>

Also, as the first action typically requires the user to lift up the board from a stationary state, a 10-second “inactivity timer” was added, where any gestures recorded beyond the 10-second timer will be ignored.

```
Twisting: 0.000000
LiftUpDown: 0.996094
Inference Duration = 5
Inactivity period exceeded. Please try again.
```

## Optimization Techniques used in Implementation

### Code size minimization

As mentioned earlier, Model 4 was chosen as a compromise between results and complexity. By doing so, the model would be able to reduce the space required, especially compared to Model 8. The reduction in the number of neurons would also reduce the size of the model substantially.

Exporting the TensorFlow Lite Model using Model 4 and Model 8, the file sizes are as follows:

	Model 4	Model 8
No Quantization	91 592 bytes for tflite file; 565 008 bytes for h file	188 624 bytes for tflite file; 1 163 373 bytes for h file
Dynamic Quantization	91 592 bytes for tflite file; 565 013 bytes for h file	188 624 bytes for tflite file; 1 163 378 bytes for h file
Full Integer Quantization	25 160 bytes for tflite file; 155 354 bytes for h file	49 648 bytes for tflite file; 306 634 bytes for h file

Surprisingly, the Dynamic Quantization did not reduce both TFLite and Header files compared to No Quantization, unlike the Full Integer Quantization counterpart. The reduction of neuron size from 60 to 30 for the two hidden layers resulted in file size drops of almost half.

However, further code size minimization can also be achieved, for example, by changing the “*AllOpsResolver*” to “*MicroOpsResolver*”, and importing the required operations instead. As the operations imported might not include features used in the new models, it might cause a reduction in functionality or even cause the microcontroller to crash during runtime.



### Real-time performance

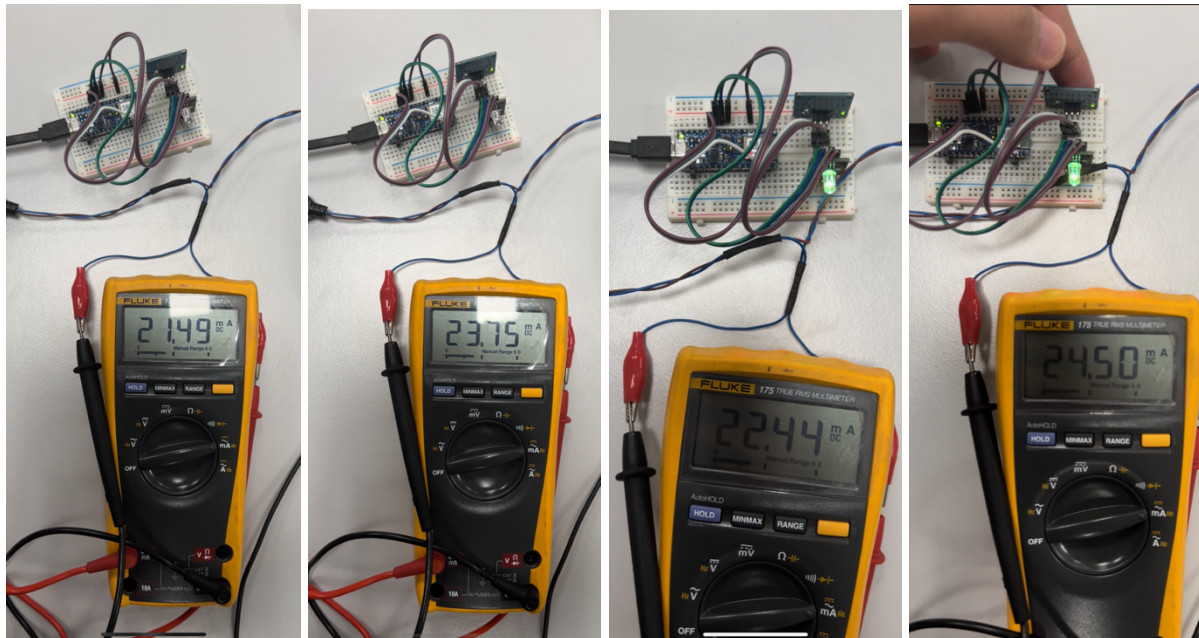
The reduction in file sizes would have a direct impact on the microcontroller's performance too, as lesser storage used would require a reduction in memory read and write times. As memory access incurs a significant penalty in latency, the reduction in memory access time would improve performance too. Lesser memory pressure on the SRAM would also ensure system stability, protecting the microcontroller from memory damage.

### Power-efficiency optimization

For the Arduino Nano, the peak power draw includes both inference and LED output. This is on top of the idle power consumption which includes the microcontroller and the touch sensor. As such, this means that there are multiple areas in which we can optimize power efficiency.

Using a Digital Multi-meter (DMM) to measure the power consumption, the following observations were made:

- When the device powers up, approximately 21.5 mA was consumed, including the connected peripherals.
- While the device waits for the gestures and during inference, approximately 23.75mA was drawn, approximately 10% higher.
- After the inference is complete and the corresponding debug LEDs were turned on, the current drawn dropped to approximately 22.5mA. This means the LEDs used approximately 1mA, or slightly under 5% higher.
- Additionally, while the device waited for the gestures and the touch sensor was activated, approximately 24.5mA was drawn, the highest power drawn in any situation.



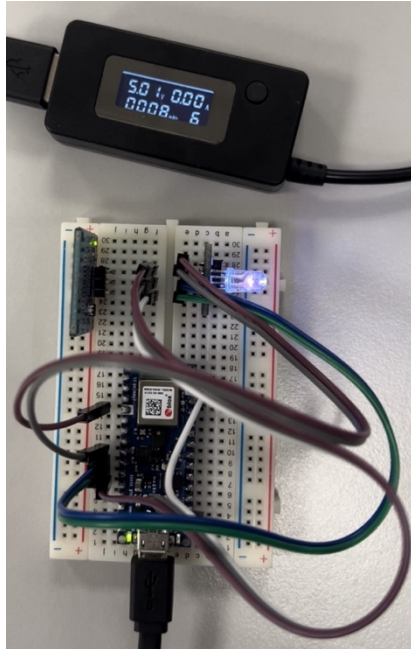
In the current setup, both inbuilt LED and the external RGB LEDs were activated after inferencing. This would mean an extra power draw would be required. In actual deployment, using either one would be sufficient for debugging purposes.



As inference accuracy is typically at odds with power consumption, a balance had to be done. With the current investigation, the imported model deployed attempts to strike that balance. However, more can be done in the future.

Underclocking the Arduino Nano and Raspberry Pi is also possible to reduce power efficiency. While that reduces both peak and idle power consumption, the total energy usage reduction might be limited to idle periods, as the reduction of clock speed would lead to increased inference processing duration, and since  $E = P * t$ , the energy consumption might increase due to additional overheads imposed by other applications.

Additional info: When using the USB Power Monitor, the current consumed is so low that 0.00A was recorded instead. Therefore, actual measurement cannot be determined using this method.

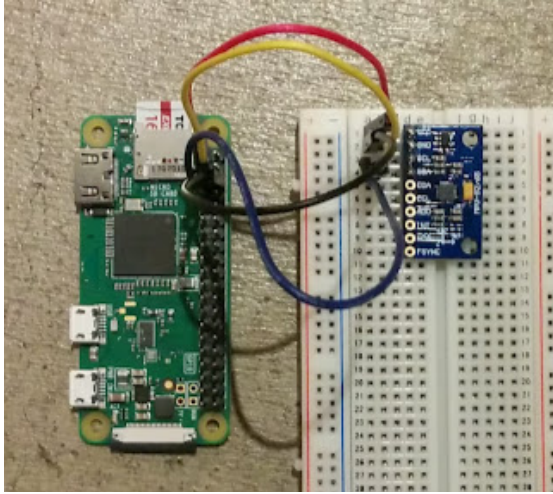


## Future Work

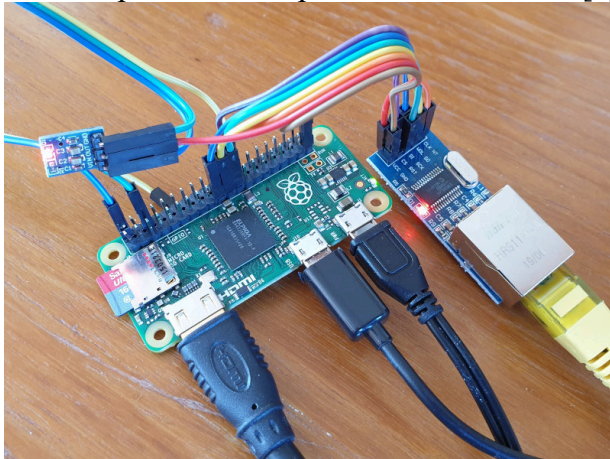
The Raspberry Pi 4 requires far more power than its predecessors [5] and the current setup would lead to the Pi mostly being in an idle state, reducing clock speed and/or using lower-powered single-board computers (SBCs) could likely provide similar performance while reducing the power consumption significantly.

Instead of using the Arduino as the Inertial Measurement Unit (IMU), an add-on to the Raspberry Pi would likely be sufficient. For example, using the MPU9250 on Raspberry Pi Zero 2 W (“Pi Zero”) would allow for direct communication between the Pi Zero and IMU, eliminating the overhead imposed by the Arduino. This would likely reduce energy usage significantly (faster inference, reduced latency, and reduced total power consumption).

An example of the setup can be found below [6]:



However, as the Pi Zero lacks an Ethernet port, the Ethernet modules like the ENC28J60 [7] have to be soldered to the GPIO pins of the Pi Zero to enable Wake-on-LAN functionality. As such, modifications to the configuration file and “wol.sh” shell script might be necessary. An example of the setup can be found below [8]:



## Conclusion

In this project, we deployed the Arduino Nano 33 Sense BLE microcontroller as a sensing device to control a Raspberry Pi 4 Model B and a Personal Computer with Wake-on-LAN functionality. We also studied the impacts of different models, and understand the importance of balancing model complexity and computing limitations imposed by the microcontrollers. This report also talks about the deployment of other embedded systems, which could reduce energy consumption at minimal performance costs.

## Bibliography

- [1] I. Luuk, “Logging Arduino Serial Output to CSV/Excel (Windows/Mac/Linux),” Circuit Journal, n.d.. [Online]. Available: <https://circuitjournal.com/arduino-serial-to-spreadsheet>. [Accessed 20 March 2022].
- [2] OpenJS Foundation, “Node-RED,” Node-RED, n.d.. [Online]. Available: <https://nodered.org/>. [Accessed 24 March 2022].
- [3] Hewlett-Packard, “Details of package etherwake in stretch,” Debian, n.d.. [Online]. Available: <https://packages.debian.org/stretch/etherwake>. [Accessed 22 March 2022].
- [4] Microsoft Corporation, “Wake on LAN (WOL) behavior in Windows 10,” Microsoft Docs, 23 September 2021. [Online]. Available: <https://docs.microsoft.com/en-us/troubleshoot/windows-client/deployment/wake-on-lan-feature>. [Accessed 28 March 2022].
- [5] J. Geerling, “Power Consumption Benchmarks,” Jeff Geerling, n.d.. [Online]. Available: <http://www.pidramble.com/wiki/benchmarks/power-consumption>. [Accessed 2 April 2022].
- [6] A. Kono, “How to use MPU9250 with Raspberry Pi,” 29 April 2018. [Online]. Available: <https://asukiaaa.blogspot.com/2018/04/raspberry-pi-mpu9250.html>. [Accessed 1 April 2022].
- [7] ElectroPeak Inc., “ENC28J60 Ethernet Network Module,” ElectroPeak Inc., n.d.. [Online]. Available: <https://electropeak.com/ethernet-module-enc28j60>. [Accessed 2 April 2022].
- [8] Matt, “Adding Ethernet to a Pi Zero,” 20 May 2020. [Online]. Available: <https://www.raspberrypi-spy.co.uk/2020/05/adding-ethernet-to-a-pi-zero/>. [Accessed 2 April 2022].