Evaluating Object Detection For Various Platforms

Kyler Martinez, Ruben Gutierrez, Austin Vinh Tran, Nelson Chong, **Mohamed El-Hadedy**
California State Polytechnic University, Pomona
Department of Electrical and Computer Engineering
Pomona, California, United States of America

***Abstract*—Computer vision has become an increasingly important field for products such as self-driving cars and the greater field of electronics. As such, there has been an increase in algorithms to perform object detection of varying capabilities, with a popular being YOLOv5. These algorithms require large amounts of computing and electrical power to support these algorithms. In a desktop environment, this may not be too much of a concern, but problems arise as the computing device decreases in size. If object detection is used in embedded systems or on smaller devices, you must determine if the algorithm can function given the device's power constraints and processor resources [7]. Common microcontrollers cannot execute algorithms like YOLOv5; however, there is an increase in machine learning tasks for microcontrollers due to simplified algorithms for microcontrollers and specialized microcontrollers for machine learning tasks [6]. Despite this, systems can use microcontrollers and devices that can process an algorithm like YOLOv5, but there are additional constraints that bottleneck performance, such as internet speeds. There is no one solution for this problem, and every implementation has flaws, but machine learning on small devices is more accessible as technology develops.**

***Index Terms*—computer vision, yolov5, raspberry pi, microcontroller, esp32, esp32cam**

I. Introduction

YOLO stands for 'You Only Look Once', and is an object detection framework using a single convolutional network. YOLOv5 is the 5th version of the YOLO series that uses Focus structure with CSPdarknet53 as a backbone. It is different from other object detection algorithms because it is a regression-based algorithm that takes in the entire image as a whole as opposed to classification-based algorithms that scans the image one pixel at a time. YOLOv5 classifies images using a grid and predicts where to place bounding boxes based on the established objects, classes, and structures [8]. We tested YOLOv5 on PCs, specifically a desktop and laptop, a Raspberry Pi 4, and used it with an ESP-32 microcontroller. The PCs utilize an x86 processor, which is a CISC architecture. On the other hand, the Pi utilizes an ARM-based CPU, which is RISC architecture, and the ESP32 is a single-core RISCV-based microcontroller, but YOLO did not run on it directly. Various tests are performed to measure and capture the performance metrics.

A. Metrics

Performance metrics are important in making trade-offs from the designer's and purchasing perspective. The goal is to understand what factors contribute to the overall system performance, and the importance and costs of these factors [10]. For YOLOv5, there are many factors that affect the overall system, but only the frames per second, inference time, and power consumption are considered in this report. These factors are considered when running the YOLOv5 algorithm on different platforms. The inference time is defined as the time it takes for a forward propagation. The inference time can be computed by dividing FLOPs by FLOPS, where FLOPs is the number of floating point operations and FLOPS is the number of floating point operations per second [9]. Frames per second and inference time have an inverse relationship. A KUMAN Power Meter KW47 [5] was used to measure the power consumption of the Raspberry Pi. For the desktop and laptop, a software called Core Temp [11] was used to monitor the power consumption. The power consumption is calculated by taking the difference between the average power and idle power. The average power is calculated by dividing the sum of the power readings by the number of power readings. The total power of the Pi was considered during its evaluation. From these metrics, trade-offs can be made through performance versus power consumption. For example, the frames per second can be increased by limiting the size of the algorithm but it decreases the performance. These metrics are used to compare how well the algorithm runs on different platforms and can be used to find the most suitable platform for any application. To test the performance metrics, YOLOv5 processes a test set of 31 images and a live camera feed.

TABLE I
Unit Reference

Name	Unit
Frames Per Second (FPS)	[frame/s]
Inference Time (IT)	[ms]
Power	[W]

II. Object Detection on PCs

When testing YOLOv5 on various devices, it can be inferred that the efficiency of the process is dependent on how strong the CPU is for each machine. For PCs and Laptops, this is especially apparent when compared to smaller devices like the Raspberry Pi. As a result, whenever YOLOv5 is running, the FPS of PCs and laptops is significantly larger. The time to process images and videos from a camera is also significantly shorter. The only drawback is that PCs and Laptops require much more power compared to a Raspberry Pi.

A. Desktop: Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz

1) Testing: The testing was done through YOLOv5's object detection by processing thirty-three different images of various

TABLE II
YOLOv5 Results Summary: PC

	Process from File	Process from Camera
Average FPS	**100.1**	**125**
Average IT	**9.99**	**8.0**
Average Power	**31.425**	**12.075**

objects, people, and things. Testing with the camera also works the same way. When the camera is processed with YOLOv5, the detection analyzes every frame of the video. In the case of this test set, the camera recorded for about 12 seconds. In terms of the tabular data, the inference was recorded from the terminal output and FPS can be calculated with the equation 1/Inference. The average power was obtained by recording the power change as YOLOv5 processed each image. This applies to both the image set and the camera set. The average values taken for the calculation of average power increase are only from the time when YOLOv5 processes the images.

2) Image Testing: The results from the image test show that the average FPS of a desktop is approximately 100.1FPS and the average inference is 9.99ms. In comparison to the camera test set, the inference has about a 2ms difference and as a result, the average FPS for the image test set is a little bit lower. The power required to test the images however, is much higher than the camera test set requiring an average power increase of 31.425W. Fig 1 shows the power increase over time for the image test set.

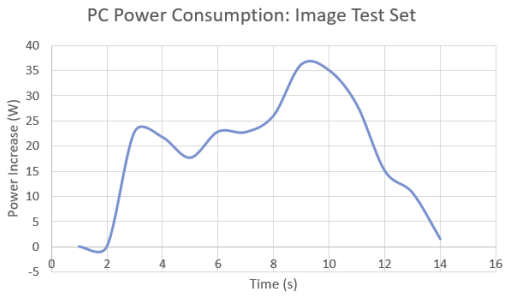

Fig. 1. PC Image Set: Average Power Increase over Time

3) Camera Testing: The results show that the average FPS of a desktop when testing the camera is 125 FPS, which is slightly higher than the image test set. The inference is also consistently around 8.0 which about 2ms faster than the image test set. The most significant difference shown from the tests is the amount of power each test set requires. In the case of the camera, the average power increase was 12.075W which is significantly less power compared to the image test set. Fig 2 shows the power increase over time for the camera test set.

B. Laptop: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz

1) Testing: Testing was done through object detection from images on a file, and a live camera feed. YOLOv5 was tested through different models or weights. The four models are small, medium, large, and extra large. Smaller models are lighter on processing requirements but are less accurate. On the other hand, larger models require more processing power but are more accurate. There is a trade-off between power and performance.

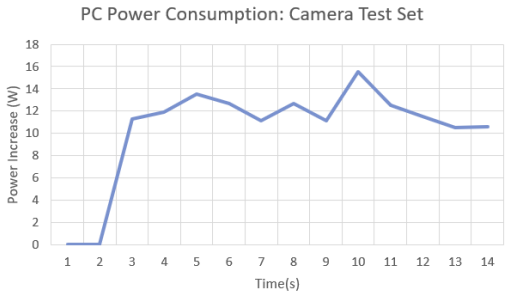

Fig. 2. PC Camera Set: Average Power Increase over Time

TABLE III
YOLOv5 Results Summary: Small Weight

	Process from File	Process from Camera
Average FPS	**4.81**	**4.11**
Average IT	**208.02**	**243.18**
Average Power	**16.3**	**14.13**

2) Process From File: The results when processing images from a file using the small weight is faster than processing images from a camera, but it draws more power. On average, it draws 16.3W. It takes 208.02ms to process each image and has an average FPS of 4.81.

3) Process From Camera: The results when processing images through a camera using the small weight is slower than processing images from a file, but it consumes less power. On average, it draws 14.13W and takes 243.18ms to process each image with an average FPS of 4.11. The performance is not much of a difference compared with processing images from a file in terms of FPS and IT but consumes less power, so this is more effective than the first test.

III. Object Detection on The Raspberry Pi

The Raspberry Pi is a microprocessor popular for IoT projects due to its small size and relatively low price point. Additionally, the Pi is used to incorporate machine learning on smaller devices with applications in computer vision.

YOLOv5 was tested on the Raspberry Pi to evaluate its performance and the practicality of object detection on small

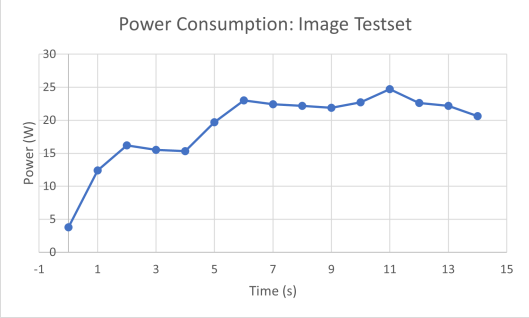

Fig. 3. Laptop: Power from Test Images.

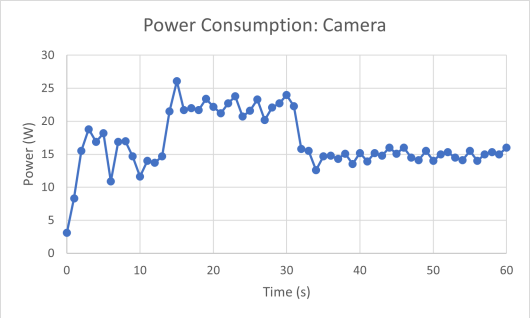

Fig. 4. Laptop: Power from Camera.

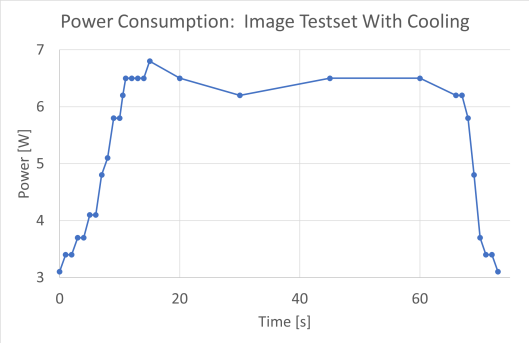

Fig. 5. Pi 4: Power when cooling is applied.

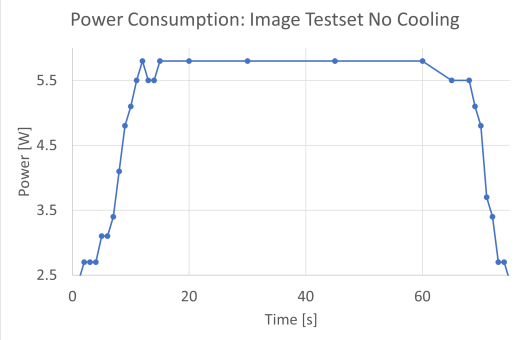

Fig. 6. Pi 4: Power when no cooling is applied.

devices like it. Throughout our tests, we used a Raspberry Pi 4 4GB model. In addition to evaluating the algorithm's performance, we tested the effects of cooling and image retrieval methods to evaluate the use cases for the Pi and YOLO.

A. Testing

For the inference time, we used two general tests with two configurations. The two tests were detection from a file and detection from a USB webcam, and the two configurations were cooling from a 5V fan and no cooling from the fan, which resulted in a total of eight tests.

The file tests used the image dataset and finished in approximately 70 seconds. The camera tests have a duration of 1200 seconds to test the performance after a long period of use. The power test measured the power throughout these tests. For the power test, we the test period is 70 seconds for the camera with cooling and 5 minutes for the camera without cooling.

B. Results

TABLE IV
PI 4 YOLOV5 RESULTS SUMMARY

	Process from file		Process from camera	
	Cooling	No Cooling	Cooling	No Cooling
Average FPS	0.65	0.47	0.47	0.33
Average IT	1520	2600	2137	2743
Average Power	6.0	5.2	7.2	6.7

1) Results: Process From File: The Pi performed best when it processed images from disk. When the fan was on, the Pi processed images at an average rate of 0.65 frames per second, while without cooling, it performed at 0.47 frames per second.

When processing images from disk, the Pi draws less power; the cooling test used an average of 6.0W, and the no-cooling test used an average of 5.2W. For both tests, the drawn power was constant during most of the operation.

2) Results: Process From Camera: The camera tests performed worse than the image tests with and without cooling. When cooling was applied, the inference time was relatively stable, varying between 2120 and 2180 milliseconds after the first 30 seconds of run time.

Without cooling, the frame rate did not stabilize to a range of values. The frame rate changed logarithmically, with performance degrading over time. After 20 minutes of use, the inference time was 2985ms per frame, with no signs of stabilizing.

The power usage of the Pi reflects behavior similar to the inference time. For the cooling test, the power drawn was consistent for most of the test's duration. The power drawn by the Pi is greater with the fan cooling than without cooling. The increase is mostly due to powering the fan, camera, and greater processing speeds.

When no cooling is applied, the power drawn is initially high but gradually decreases over time before stabilizing at 6.5W.

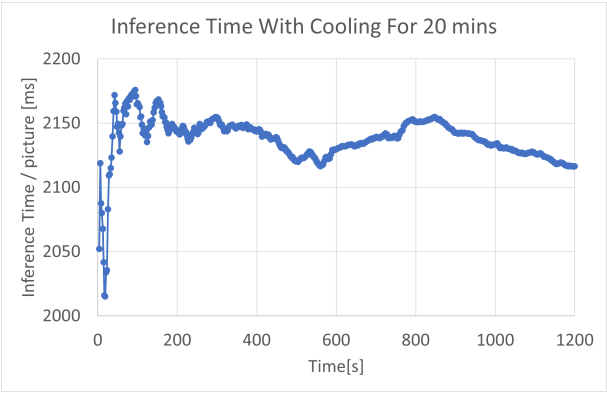

Fig. 7. Pi 4: Inference time when cooling is applied.

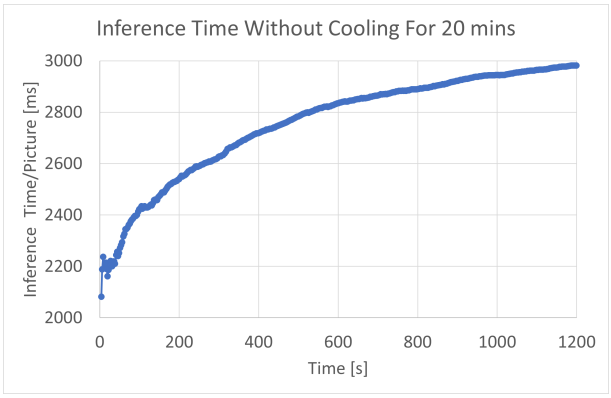

Fig. 8. Pi 4: Inference time when no cooling is applied.

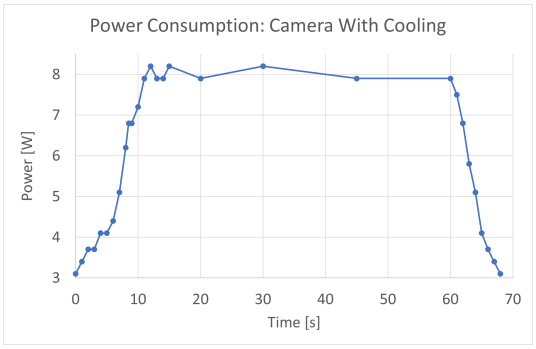

Fig. 9. Pi 4: Power cooling is applied.

C. YOLOv5: Evaluation

From our results, the best performance is when images are retrieved from the external SD card. However, many systems require a live camera feed and identify objects using that data.

When the system is cooled, the required power increases but with improvements in performance and power stability. Without cooling, the power requirements are lower, but this comes at the cost of degrading performance and inconsistent power consumption.

Overall, the YOLOv5 algorithm performs poorly on the Raspberry Pi 4. The Pi 4 does not have the hardware required to execute the algorithm with the performance necessary for most practical applications. As a result, a less computationally intensive algorithm is necessary for object detection applications using a Raspberry Pi.

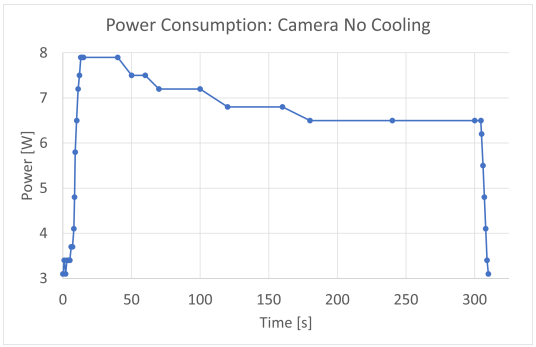

Fig. 10. Pi 4: Power when no cooling is applied.

IV. Object Detection on Microcontrollers

Microcontrollers (MCUs or also known as micros) are simple processors that perform only one task, it can be a simple task like listening to sensors or detecting objects on images [15].

Many limitations exist with using MCUs; for instance, MCUs are limited in memory. A microcontroller typically contains anywhere from hundreds and thousands of bytes [14], limiting the micro's ability to perform a simple task. An additional constraint for MCUs is power, microcontrollers are low-powered and therefore can not handle high-power devices or output high wattage [12].

Following the constraints, microcontrollers are limited in the ability to do machine learning (ML) due to the complexity of the process. ML is defined as computer systems that are able to learn and adapt without any clear instructions but simply by following algorithms and models to analyze and draw inferences from data patterns [13]. Machine Learning is not possible on micros because of complex requirements that are necessary from the models. However, ML on microcontrollers is feasible through other possible solutions.

A. TensorFlow Lite and Microcontroller

The microcontroller tested was the ESP32. For this part, Raspberry Pi, ESP32 microcontroller, and ESP32 Cam were wired and connected together to perform object detection. This process is only possible if TensorFlow Lite is detecting objects because machine learning can not be done on the microcontroller directly. TensorFlow Lite analyzes the images from the microcontroller and completes the object detection. This idea is different and unique because it allows for object detection to be completed on a tiny device with such a complex build and many limitations.

B. YOLO and Microcontroller

An alternative to using the YOLOv5 algorithm was to stream the data from the ESP32 micro and the ESP32 cam to a device that could run YOLOv5. For this section, a laptop would run the YOLOv5 algorithm as it receives the feed from the cam and micro in order to do the object detection and machine learning similar to using TensorFlow.

C. Results on the Microcontroller

For the ESP32 and TensorFlow, the average FPS rate is 1.28 and the average inference time is 783.43 ms with no cooling. On the other hand, with cooling applied the average FPS rate is 1.90 and the average inference time is 525.67 ms. When cooling is applied to the ESP32 and TensorFlow, the performance time seems to be slower than when no cooling is applied. As seen in the graphs, many spikes occur as a result of instability. The instability is a result of the network latency that occurs due to the Wi-Fi connection.

Finally, the analysis for YOLO and Windows has an average FPS rate of 6.04 and the inference time is 165.61 ms. Similar

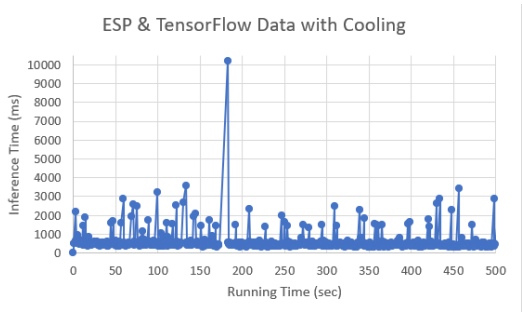

Fig. 11. Inference time of ESP32 and TensorFlow with cooling.

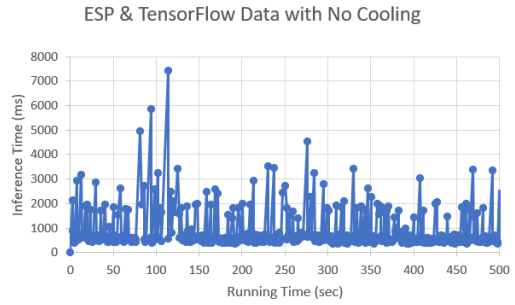

Fig. 12. Inference time of ESP32 and TensorFlow with no cooling.

to the ESP32 microcontroller and TensorFlow graphs, this graph contains sudden spikes because of the unstable network.

For power consumption, the ESP32 micro and the ESP32Cam are negligible. However, when the microcontroller and the camera are connected together with the Raspberry Pi, the power used increases by 0.7W. While idle the Raspberry Pi has a measured power usage of 2.7W, but when connected to the microcontroller, the power increases to 3.4W.

V. Conclusion

Object detection algorithms such as YOLOv5 are incredibly powerful when applied to problems requiring computer vision, but are computationally intensive. Modern desktop PCs have the capabilities to process frames and perform inferences fast enough to be useful in many applications. However, if we wish to use object detection on smaller devices or in embedded systems, computational resources and power constraints must be accounted for. If object detection were to be implemented on a microprocessor like a Raspberry Pi, an algorithm like YOLOv5 underperforms, but a less intensive algorithm, like TensorFlow Lite, can run with much greater success. Running YOLOv5 on a microcontroller is infeasible for many of the common microcontrollers on the market, but there are frameworks for running some algorithms like TensorFlow Lite. If an algorithm like YOLOv5 is used with a microcontroller, it is possible to stream the video feed from the microcontroller, process it on another machine, and have the microcontroller respond accordingly. However, this method accrues additional costs and forces internet constraints that bottleneck the overall system. Despite this, it is feasible to process the feed from the ESP32CAM and achieve results in an acceptable time frame despite the bottlenecks of the internet connection.

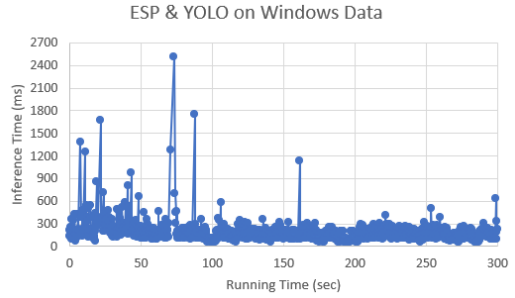

Fig. 13. Inference time of ESP32 and YOLO on Windows.

Acknowledgment

We thank Dr.Mohamed El-Hadedy for their guidance and feedback throughout the duration of our studies and work.

References

[1] Ultralytics (2022) YOLOv5 (Version 7.0) [Source Code] https://github.com/ultralytics/yolov5
[2] FreedoomTechWeb (2022) YOLOv5Raspberry Pi 4 [Source Code] https://github.com/freedomwebtech/yolov5raspberry-pi4
[3] TensorFlow (2022) TensorFlow (Version 2.11.0) [Source Code] https://github.com/tensorflow/tensorflow
[4] FreedoomTechWeb (2022) ESP32Cam Object Detection [Source Code] https://github.com/freedomwebtech/esp32camobjectdetection
[5] KUMAN Power Meter KW47 User Manual, KUMAN. Accessed: Sep. 30, 2022. [Online]. Available: https://manuals.plus/kuman/kw47-power-meter-manual
[6] Lauren Hinkel — MIT-IBM Watson AI Lab, "Tiny Machine Learning Design alleviates a bottleneck in memory usage on internet-of-things devices," MIT News — Massachusetts Institute of Technology. [Online]. Available: https://news.mit.edu/2021/tiny-machine-learning-design-alleviates-bottleneck-memory-usage-iot-devices-1208. [Accessed: 26-Nov-2022].
[7] M. Stewart, "Tiny Machine Learning: The next ai revolution," Medium, 03-Oct-2020. [Online]. Available: https://towardsdatascience.com/tiny-machine-learning-the-next-ai-revolution-495c26463868. [Accessed: 26-Nov-2022].
[8] D. Davies, "Yolov5 object detection on windows (step-by-step tutorial)," W&B, 20-Sep-2021. [Online]. Available: https://wandb.ai/onlineinference/YOLO/reports/YOLOv5-Object-Detection-on-Windows-Step-By-Step-Tutorial—VmlldzoxMDQwNzk4. [Accessed: 26-Nov-2022].
[9] J. Cohen, "How to optimize a deep learning model for faster inference?," Think Autonomous, 19-Oct-2022. [Online]. Available: https://www.thinkautonomous.ai/blog/deep-learning-optimization/. [Accessed: 26-Nov-2022].
[10] D. R. Parthasarathi, "Computer Architecture," Performance Metrics – Computer Architecture, 24-Jul-2018. [Online]. Available: https://www.cs.umd.edu/ meesh/411/CA-online/chapter/performance-metrics/index.html. [Accessed: 26-Nov-2022].
[11] Core Temp. [Online]. Available: https://www.alcpu.com/CoreTemp/. [Accessed: 26-Nov-2022].
[12] "Advantages and disadvantages of microcontroller," Polytechnic Hub, Apr. 13, 2017. https://www.polytechnichub.com/advantages-disadvantages-microcontroller/ [Accessed Nov. 27, 2022].
[13] IBM Cloud Education, "What is Machine Learning?," IBM, Jul. 15, 2020. https://www.ibm.com/cloud/learn/machine-learning [Accessed Nov. 27, 2022].
[14] M. Lab, "Microcontroller Memory Organization and Types - Explained with Memory Segments," Microcontrollers Lab, Aug. 15, 2020. https://microcontrollerslab.com/microcontroller-memory-organization-types-memory-segments/ [Accessed Nov. 27, 2022].
[15] "Microcontrollers: The Basics – ITP Physical Computing," Nyu.edu, 2021. https://itp.nyu.edu/physcomp/lessons/microcontrollers-the-basics/.