Bridging Machine Learning and Embedded Systems: Edge AI Experiments for Undergraduate Education

Saipranith Oku, Stevens Institute of Technology (School of Engineering and Science)

Saipranith Oku is a Machine Learning Engineer at UBS with a background in applied artificial intelligence and data science. He received his Master of Engineering in Applied AI from Stevens Institute of Technology. His interests include building practical machine learning systems for real-world applications, with a focus on continued research and advancing knowledge in artificial intelligence

zhenglong xu, Stevens Institute of Technology (School of Engineering and Science)

Zhenglong Xu is a graduate student in Electrical Engineering at Stevens Institute of Technology. He received his undergraduate training in optics, which informs his interest in integrating machine learning with physical systems. His work focuses on Edge Artificial Intelligence and embedded systems. He aims to bridge AI algorithms with practical hardware applications for engineering education and industrial use.

Mahmoud Al-Quzwini, Stevens Institute of Technology (School of Engineering and Science)

Mahmoud Al-Quzwini, Ph.D. Mahmoud Al-Quzwini is a senior lecturer with the department of Electrical and Computer engineering at Stevens Institute of Technology. He received his Ph.D. in Communications Engineering (2008) and, M.Sc. (1998) & B.Sc. (1995) degrees in Electronic and Communications Engineering from Al-Nahrain University, Iraq. He was a postdoctoral fellow at Florida State University in 2010. His current research interests include embedded systems and engineering education.

Bridging Machine Learning and Embedded Systems: Edge AI Experiments for Undergraduate Education

Abstract

The accelerating integration of Artificial Intelligence (AI) and Edge AI into modern engineering applications highlights the urgent need to expose undergraduate students to practical, hands-on experiences in these domains. This paper presents the development of a structured laboratory framework that bridges theoretical AI principles with embedded, real-time implementation through accessible Edge AI experiments. The work builds on an instructional case study using the Arduino Nano 33 BLE Sense and Edge Impulse, an open-source machine-learning platform designed for low-power devices. Within this framework, students collect and label image data using an onboard camera, design and train lightweight convolutional models such as MobileNetV2 (0.35) with FOMO (Faster Objects, More Objects) architecture, and deploy them on-device to perform object classification and counting.

The proposed experiments emphasize the complete machine-learning life cycle from data acquisition and preprocessing to model training, evaluation, and edge deployment without the need for GPUs or cloud resources. This approach does not only reinforces core machine-learning and embedded-systems concepts but also cultivates problem-solving, iteration, and experimental design skills essential to modern engineering education.

Preliminary implementation within the undergraduate embedded-systems context demonstrates that the Edge AI experiments enhance student engagement and conceptual understanding of AI pipelines, model efficiency, and hardware–software co-design. The paper concludes by outlining best practices for integrating AI-and-Edge-AI laboratories into electrical and computer engineering curricula, illustrating how accessible hardware and cloud-free platforms can facilitate AI education and prepare students for the next generation of intelligent, data-driven embedded systems.

1 Introduction

Edge Artificial Intelligence (Edge AI) has become increasingly important in real-world applications such as industrial inspection, autonomous systems, and embedded sensing. As noted by T. Meuser et al. [1], unlike traditional cloud-based machine learning, Edge AI focuses on deploying models directly on low-power hardware, enabling real-time inference with reduced latency, bandwidth usage, and energy consumption.

Previous studies have highlighted the value of incorporating practical machine learning applications into education, particularly through embedded and Edge AI systems. Related work has

demonstrated the use of simplified platforms and resource-constrained hardware to support hands-on learning and improve students' understanding of machine learning concepts. However, many existing approaches either address isolated stages of the machine learning pipeline or rely on relatively complex configurations, which may reduce their accessibility for undergraduate students [2].

The ultimate purpose of developing machine learning theories and methods is to solve real problems in industry and daily life. As highlighted in previous research, effective instruction should begin with meaningful application scenarios rather than abstract algorithms alone. According to the characteristics of undergraduate students, the integration of application scenarios that are well suited to machine learning technologies should be strengthened. For example, real-world applications of machine learning in fields such as autonomous driving and industrial production can stimulate students' interest in course content and enhance their motivation to participate in the curriculum.

Based on these considerations, this work presents an Edge AI experiment that simulates a real industrial production line inspection task. By integrating machine learning with embedded systems, the experiment explains the principles behind parameter selection and model design while using simple methods to construct a complete system. Through this process, students gain a deeper understanding of real-world problem contexts and machine learning techniques, and become better equipped to analyze and solve practical issues.

The goal of this project is to create a simple and complete example of a real-world Edge AI system in which an object classification and counting model is deployed on an Arduino Tiny Machine Learning Kit using the Edge Impulse platform. This experiment was chosen because it represents a typical industrial inspection task while remaining intuitive and easy to understand for educational purposes. Edge AI was selected for its relevance to modern industrial applications and its ability to support real-time inference on resource-constrained devices, while the Arduino Tiny Machine Learning Kit was chosen due to its low cost, accessibility, and suitability for hands-on learning without the need for high-end hardware. The experiment illustrates the full lifecycle of a machine learning system, from data collection to final results, without the need for high-end GPUs, massive datasets, or complicated code. Beyond technical implementation, this project demonstrates how educational experiments can promote deeper learning by allowing students to directly observe, modify, and evaluate the systems they build.

To further support its use in undergraduate engineering education, this experiment is designed with the following learning objectives. After completing this experiment, students will be able to: understand the concepts of Edge AI and machine learning, design and implement an end-to-end Edge AI pipeline, including data acquisition, labeling, model training, and deployment on a microcontroller. Analyze the constraints of embedded systems (e.g., memory, computation, and power) and evaluate their impact on model selection. Apply lightweight deep learning models (e.g., MobileNetV2 and FOMO) for real-time object detection tasks.

2 Embedded Hardware Platform

Before diving into the software portion of this experiment, it is important to understand the hardware that makes the system possible. Having a clear understanding of the hardware helps students see how physical components interact to create an intelligent system. It also provides a deeper appreciation for how sensors, processors, and supporting modules come together to form a complete machine learning setup. Understanding the hardware is a critical first step before exploring how the software drives the functionality.

2.1 Arduino Nano 33 BLE Sense

At the center of our setup is the **Arduino Nano 33 BLE Sense** board, shown in Figure 1. This small but powerful microcontroller serves as the foundation for the entire experiment. It is equipped with a wide range of built-in sensors capable of detecting motion, acceleration, rotation, barometric pressure, sound, gestures, proximity, color, and light intensity [3].

The inclusion of so many sensors makes the board a very flexible and educational tool. It allows students to experiment with multiple data sources without needing to purchase or connect many different components. The board is not only energy-efficient but also specifically designed to make it easier for beginners and students to learn how embedded machine learning works. Rather than being a difficult hardware challenge [4], it serves as a well-structured learning platform that guides users through the entire process of building and deploying small-scale AI applications.

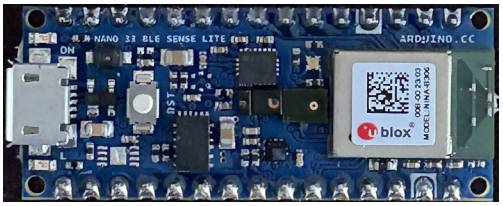

Figure 1: Arduino Nano 33 BLE Sense Board

2.2 Camera Module (OV7675)

The main sensor used for this project is an external **OV7675 camera module**, shown in Figure 2. This component acts as the main input device responsible for collecting image data. The data captured by the camera will be used for both training and testing the machine learning model that performs object counting and classification.

The OV7675 camera supports a maximum resolution of 640×480 (VGA); however, lower resolutions were used in this experiment to meet the memory and computational constraints of the embedded device [5].

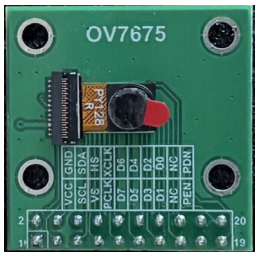

Figure 2: OV7675 External Camera Module

3 Edge AI Software Framework

Edge Impulse is used as the primary Edge AI development platform in this experiment. It provides an integrated environment for data acquisition, preprocessing, model training, and deployment on embedded devices. By offering a unified workflow within a single platform, Edge Impulse enables rapid development of machine learning models while keeping the overall system accessible for educational use [6]. The platform is officially compatible with the Arduino Nano 33 BLE Sense, which simplifies hardware integration and reduces system setup complexity.

The platform's data acquisition tools allow image samples to be collected directly from connected embedded devices. Captured data can be visualized and labeled through a web-based interface, enabling efficient dataset preparation. After data collection, Edge Impulse applies built-in signal and image processing operations, including resizing, normalization, and feature extraction, to transform raw sensor data into suitable inputs for model training [7].
The extracted features are then used to train a neural network within the same development environment. Users can configure model architecture and training parameters without relying on external machine learning frameworks. Upon completion of training, the resulting model is automatically optimized for deployment on low-power microcontrollers through techniques such as quantization and memory optimization. This ensures that the deployed model meets the performance and resource constraints of embedded systems.

By supporting a wide range of hardware targets and enabling efficient on-device inference, Edge Impulse facilitates the deployment of trained models without requiring cloud connectivity. In this context, Edge Impulse functions not only as a development framework but also as an educational platform that bridges theoretical understanding with practical implementation. Through the use of this platform, the experiment demonstrates how Edge AI systems can be designed, trained, and deployed under real-world resource constraints, providing a foundation for future industrial applications.

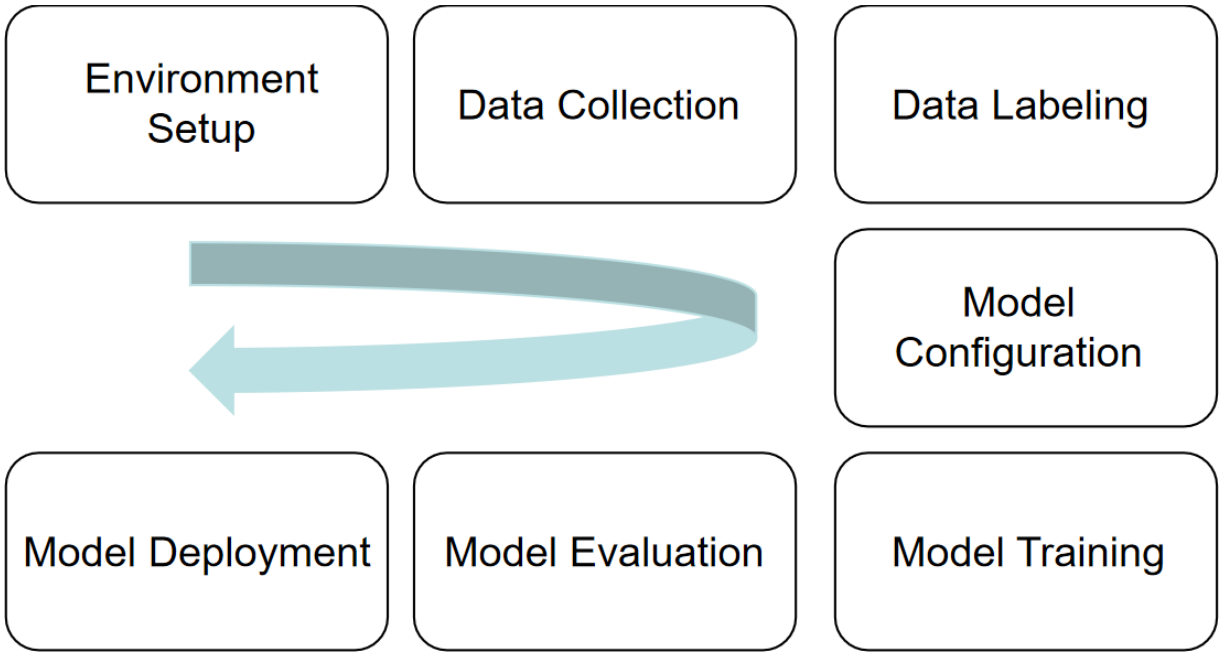

Figure 3: Overall workflow of the Edge AI experiment

4 Experimental Setup

To provide an overview of the experimental process, Figure 3 illustrates the complete Edge AI workflow used in this study.

Step 1: Creating a Test Environment

The first step is to create a simple and controlled test environment that ensures consistent and clear image capture. For this experiment, we used a plain white background so that the camera module can easily detect the nuts and bolts without distractions. A few sheets of clean white paper were placed on a flat surface, and different combinations of nuts and bolts were arranged on it for image collection, as shown in Figure 4. Then we need to open the Edge Impulse platform and create a new project. The workflow is illustrated in Figure **5** begins by creating an Edge Impulse account, followed by initializing a new project in Edge Impulse Studio, assigning a project name, and completing the project creation process.

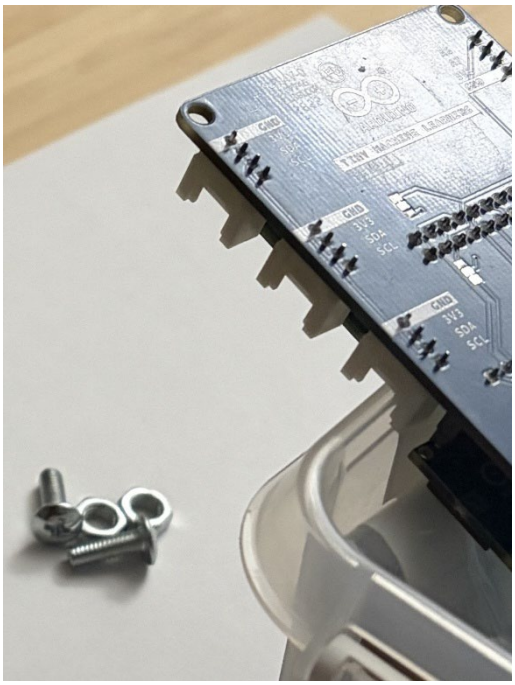

Figure 4: Hardware and camera setup for Edge AI inspection experiment

Step 2: Photo Collection

The next step is to collect data using the Arduino camera module which is mentioned in the hardware section. To do this, we must ensure that our device is properly connected to our personal computer and is also connected to the Remote Management API of Edge Impulse. This will allow for the proper communication from our device to the Edge Impulse Software. Once connected, we can simply go to the data collection section in our Edge Impulse software and select our proper camera configuration (as mentioned in the hardware section). This can be seen in Figure 6. All of this is done without writing any code, making the process seamless.

To improve model accuracy and increase robustness across real-world environments, the training dataset includes images captured under different lighting conditions. By exposing the model to variations in illumination, the system learns to generalize better and maintain reliable detection performance under changing lighting scenarios.

The dataset is organized into seven classes (class0–class6), where each class represents a different combination and quantity of nuts and bolts present in a single image. For each class, images were collected under multiple lighting conditions to ensure that the model can correctly recognize and count objects regardless of illumination changes [8].

Figure 7 illustrates four raw images captured under different lighting conditions for the class representing two nuts and one bolt.

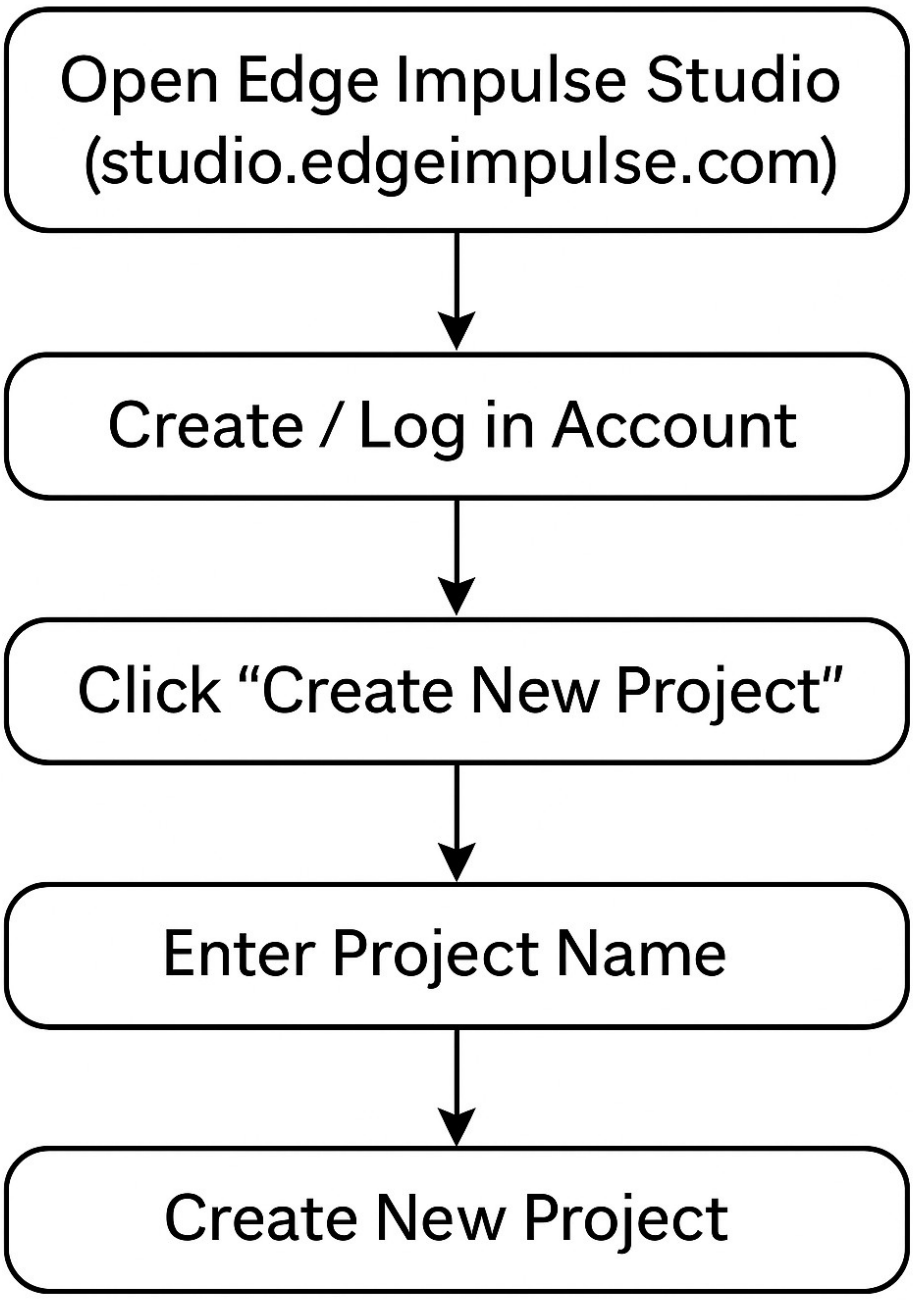

Figure 5: Workflow for creating a new project

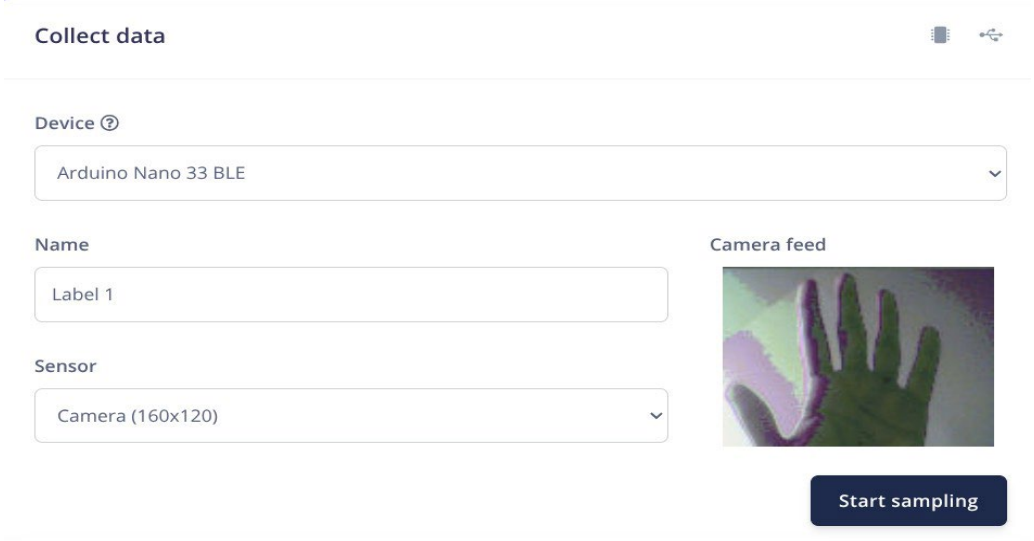

Figure 6: Data collection interface in Edge Impulse

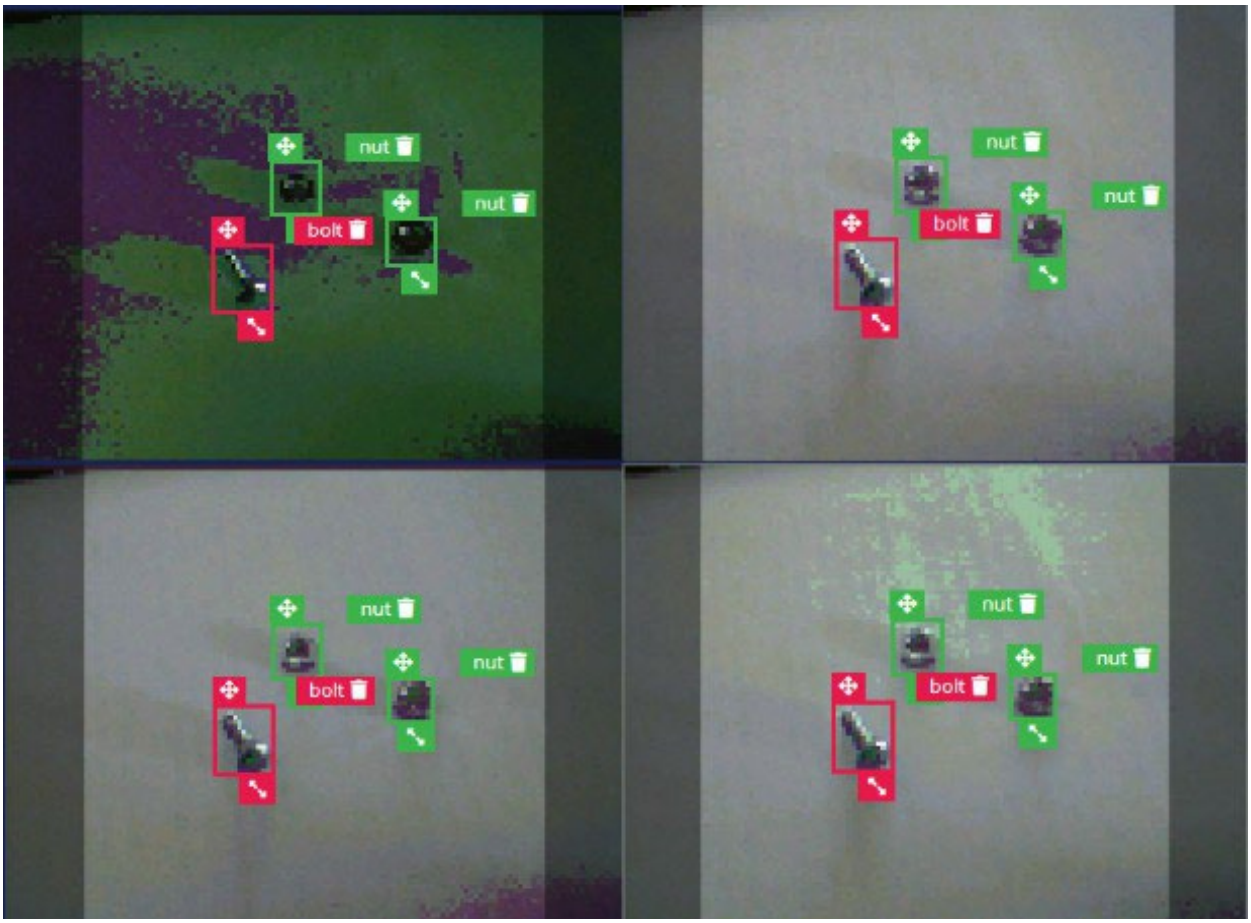

Figure 7: Raw images captured under different lighting conditions

Step 3: Labeling the Data

After collecting all the photos, the next step is to label the dataset so that the machine learning model can learn to distinguish between the different classes. In Edge Impulse, data labeling is performed directly within the Data Acquisition section, which can be found in the left-side menu of the platform. Each image is assigned to its corresponding class folder based on the number of nuts and bolts shown in the photo.

With recent updates to the Edge Impulse platform, multiple AI-based labeling models, including Gemini, Chat GPT-based models, and OWL-ViT, are available for automatic data annotation. All three approaches support text-based object recognition, allowing users to specify the target objects by entering their names into a prompt field. Based on the provided textual descriptions, the selected model automatically identifies and annotates the corresponding objects across the dataset.
For this project, we choose OWL-ViT to perform the initial automatic labeling because it fits well with embedded and educational experimentation. OWL-ViT (Open-World Vision Transformer), proposed by Google, supports zero-shot object detection and identifies arbitrary object categories based on textual prompts [9]. The model integrates directly into the Edge Impulse platform, runs without relying on any external API, and remains free to use, which makes it suitable for controlled experimental environments and classroom use.

The first step is entering the text prompt “a nut (nut, 0.2)” and “a bolt (bolt, 0.2)” into the input field to perform automatic annotation. The detailed input settings are shown in Figure 8. The confidence threshold specifies the minimum probability required for the model to generate a bounding box prediction. A bounding box is a rectangular region drawn around a detected object to indicate its position in the image. Setting a relatively low threshold reduces missed detections during the initial labeling stage and allows the system to generate more candidate annotations for subsequent manual verification and refinement [10].

OWL-ViT still produces a noticeable number of labeling errors, so manual verification is required. By selecting the Dataset in Data acquisition, we can iterate through the entire dataset, identify incorrectly annotated images, and manually adjust the bounding boxes by dragging them to the correct positions. The details are shown in Figure 9.

Step 4: Processing the Data

After the data labeling, the next step is to design an *Impulse* in Edge Impulse. An Impulse is a structured workflow that defines how raw input data are processed, features are extracted, and the model is trained [11]. In this step, students select the input type (image), set up image processing blocks, and choose a learning block such as object detection. Edge Impulse then helps extract meaningful features such as color, texture, and shape from the images, which are used during training. This stage is essential because it prepares the data in a form that the model can understand.

In our Impulse we set the image width and image height to 60px, the resize mode to Fit the longest axis. This mode can keep the original shape of the object and avoids image distortion. Then in the

image tab, we used Grayscale as the color depth to reduce the image size and make it more suitable for classification. The details are shown in Figure 10.

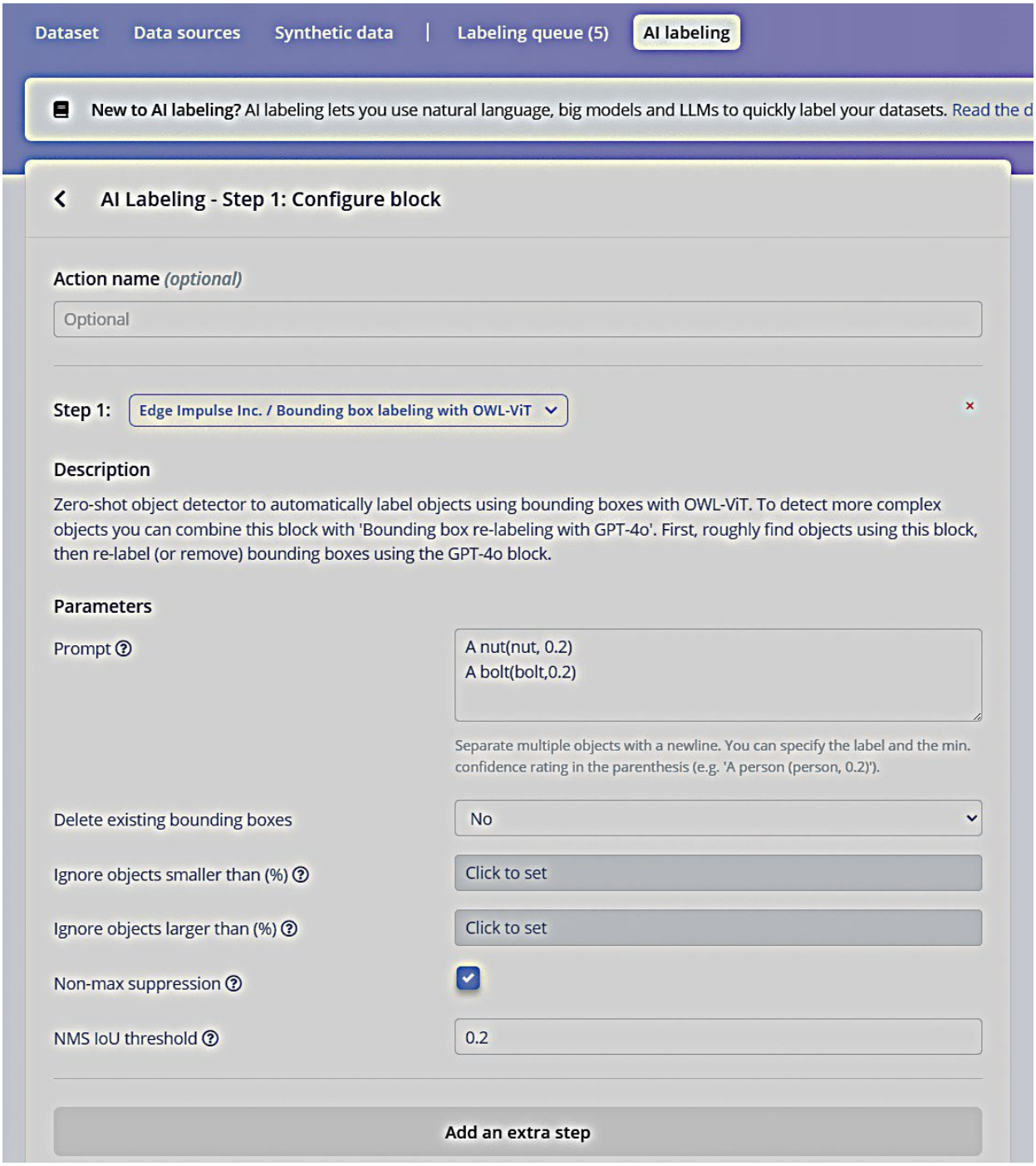

Figure 8: OWL-ViT–based AI labeling for automatic nut and bolt bounding box annotation.

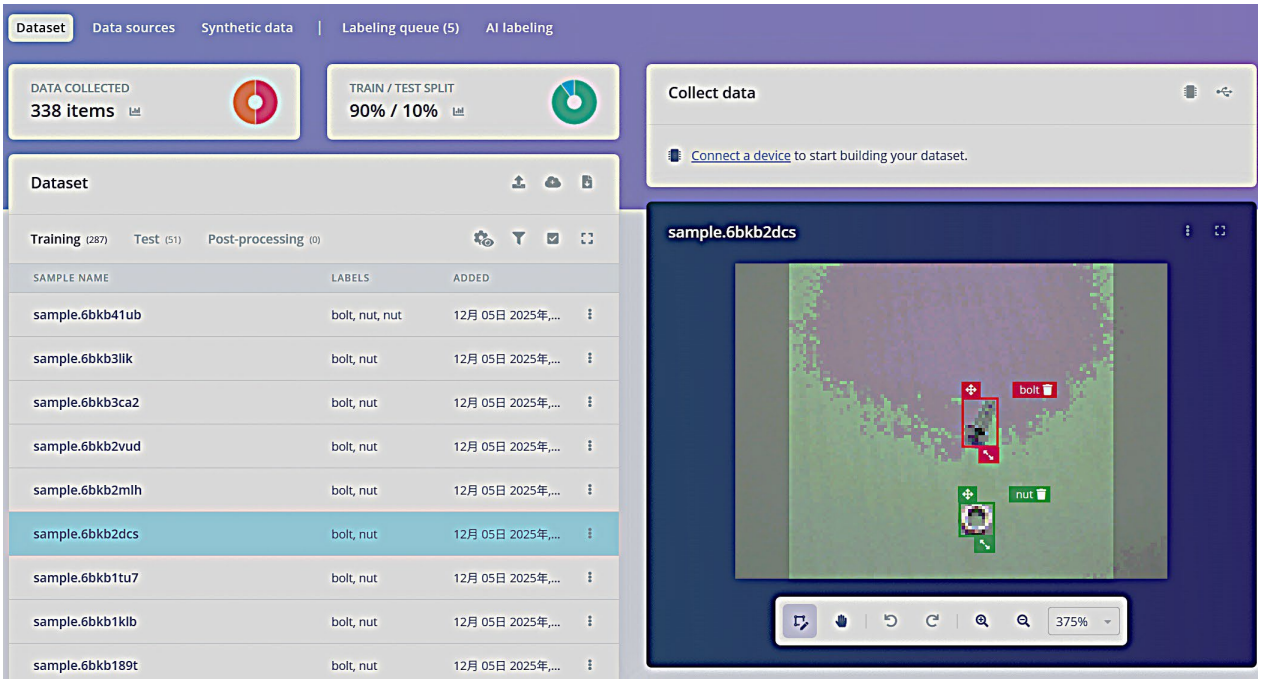

Figure 9: Manual annotation example

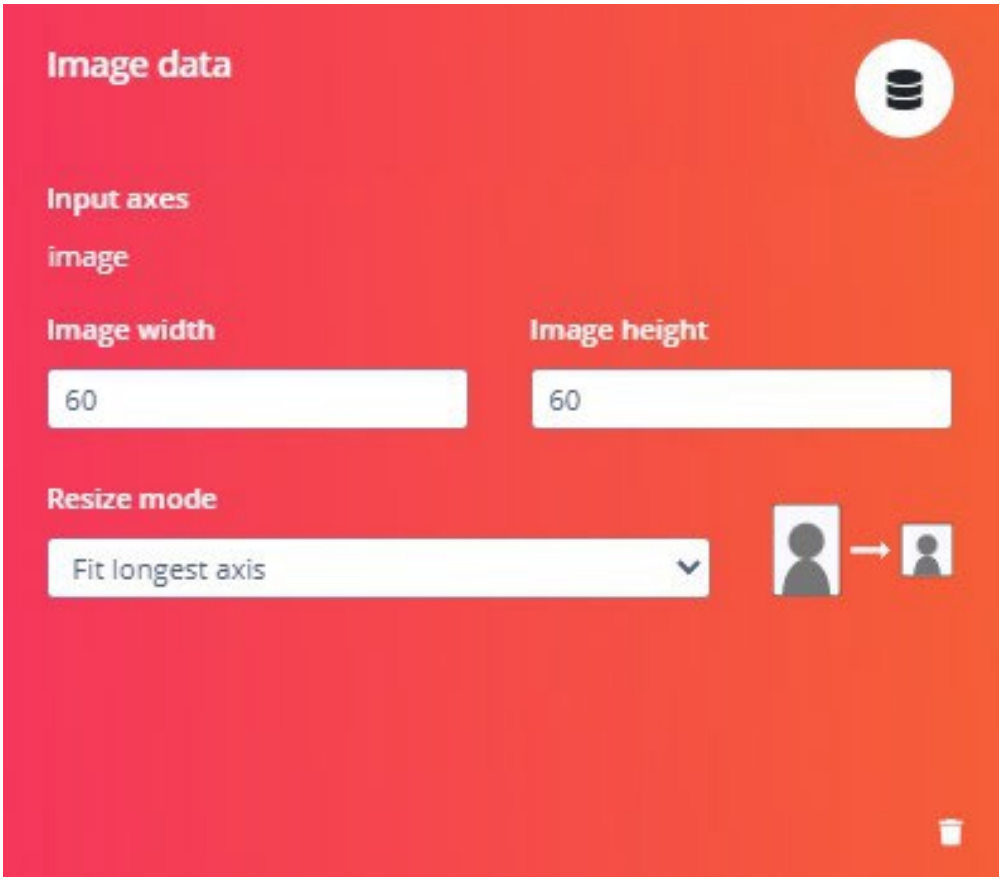

Figure 10: Image input configuration and resize settings

Step 5: Training the Model

Once the impulse design is completed, the next step is selecting and training an object detection model that can operate efficiently on a resource-constrained embedded platform. In this project, the target hardware is the Arduino Nano 33 BLE Sense, which is equipped with a 64 MHz microcontroller, 256 KB of RAM, and 1 MB of flash memory. These strict computational and memory constraints significantly limit the range of deep learning models that we can deploy.

There are four object detection model we can choose on Edge Impulse: YOLO Pro, MobileNetV2 SSD FPN-Lite (320x320), MobileNetV2 0.1 and MobileNetV2 0.35. Each model presents different trade-offs in terms of accuracy, model size, and computational cost.

YOLO Pro(You Only Look Once): YOLO is a highly efficient object detection model known for its impressive accuracy and performance in terms of Mean Average Precision (mAP). It offers fast inference speeds, exceeding 230 frames per second (FPS) on NVIDIA RTX 4090 [12]. But the board we used Adriano BLE only has a 64 MHz MCU, 256KB RAM and 1 MB Flash, while YOLO typically needs at least a 1 GHz CPU and 1-2 GBofRAM, thus, we can't choose YOLO.

MobileNetV2 SSD FPN-Lite (320×320): MobileNet is a Convolutional Neural Network (CNN) architecture model that explicitly focuses on Image Classification for mobile applications, what makes this model stand out is that its architecture reduces computational cost and computing power is very low. Feature Pyramid Network (FPN) can handle the problem of scale mismatch in image processing. Combine MobileNetV2 with SSD and FPN generate multi-scale features that can represent objects at various scales with different resolutions [13].Our board has only 256 KB RAM and 1 MB Flash, while MobileNetV2 SSD FPN-Lite requires several megabytes of storage and tens of megabytes of RAM, making it impossible to load or execute.

MobileNetV2 is the backbone of the MobileNetV2 SSD FPN-Lite. By removing the SSD detection head and the FPN-Lite feature pyramid, and retaining only the classification feature extraction components, the model size is reduced dramatically—less than 500KB. This lightweight structure allows the model to run efficiently on small embedded devices, including Arduino Nano 33 BLE Sense. MobileNetV2 (0.35) has wider channel capacity and produces richer feature maps than MobileNetV2 (0.1). As a result, it captures more detailed visual information and achieves much stronger detection accuracy while remaining lightweight. Therefore, MobileNetV2 (0.35) has been selected as the feature extractor.

FOMO (Faster Objects, More Objects), is a lightweight object detection method developed by Edge Impulse. The system can detect and track multiple objects on small devices such as smartphones and various types of IoT sensors [14]. FOMO is designed to operate efficiently in environments with limited processing power and memory capacity. It is extremely small—less than 100KB, making it ideal for deployment on microcontrollers and other low-resource hardware platforms.

MobileNetV2 (0.35) provides visual features, while FOMO uses these features to perform ultra-lightweight object detection, enabling high accuracy on resource-constrained edge devices. The overall architecture and dataflow of the MobileNetV2 (0.35) and FOMO model are illustrated in Figure 11. Moreover, Edge Impulse has already integrated both FOMO and MobileNetV2 (0.35), allowing us to build and deploy the model on the platform with ease.

In this project, we select "Object Detection" from the menu to begin training the model. On the model selection page, we choose "FOMO (Faster Objects, More Objects) with MobileNetV2 0.35".

A total of 304 images with labeled bounding boxes are used for training. Training cycles determine how many times the model learns the dataset; too many cause overfitting, too few lead to underfitting. The learning rate controls the step size of each update; too high causes instability and too low slows learning [15]. To balance training time and accuracy, we set the number of training cycles to 100 and the learning rate to 0.004. The details are shown in Figure 12 . After saving the settings, Edge Impulse automatically uses the provided dataset to train the FOMO model.

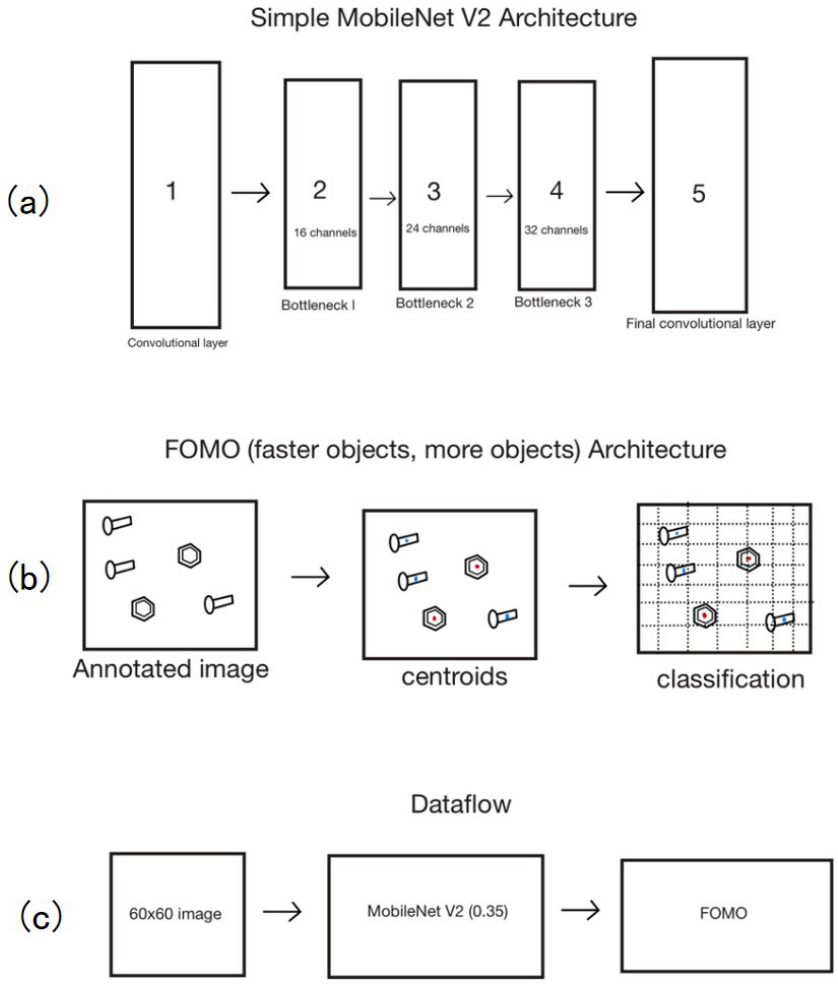

Figure 11: Architecture and dataflow of the proposed system: (a) MobileNetV2 (0.35) backbone, (b) FOMO detection mechanism, (c) overall dataflow from input image to detection output.

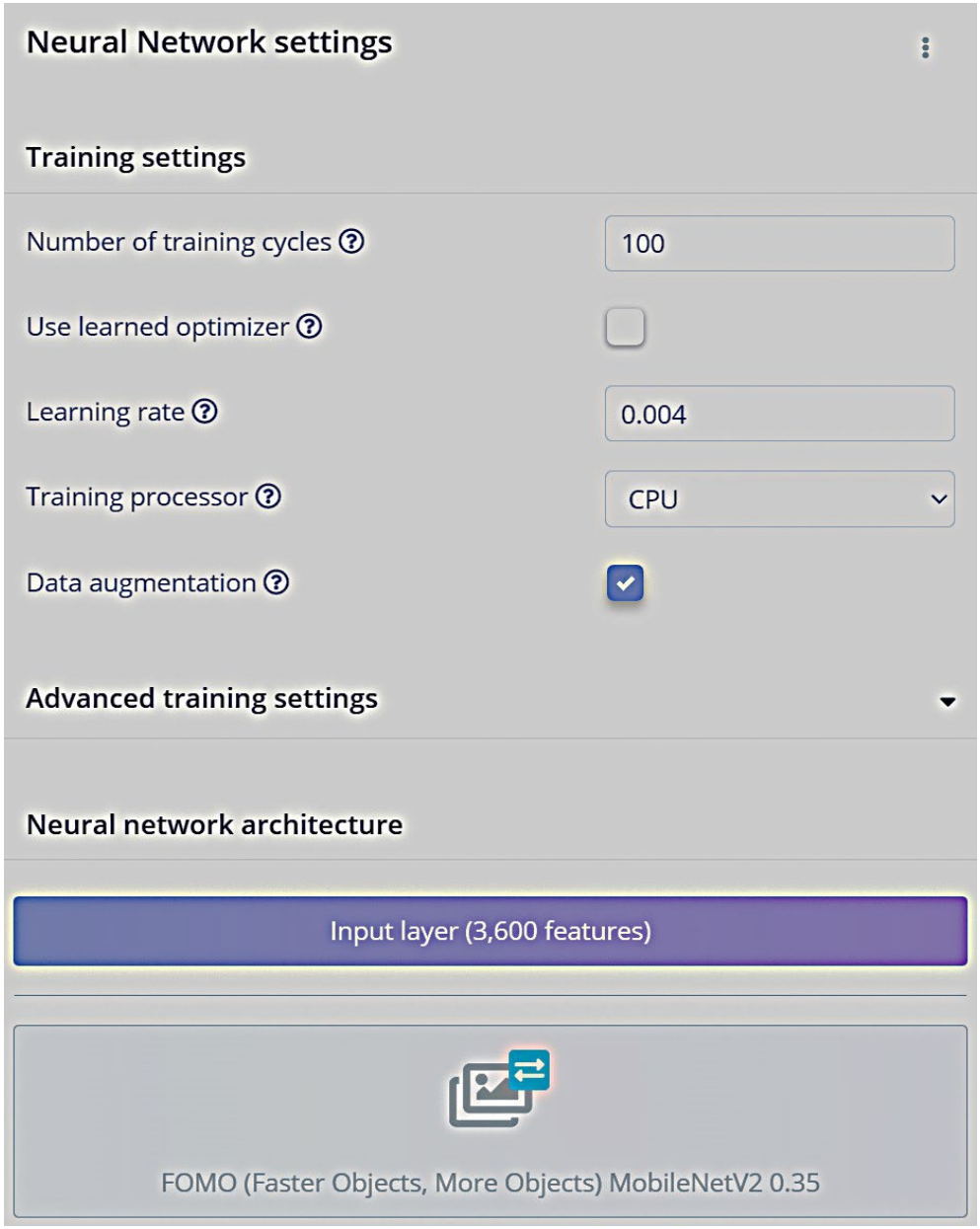

Figure 12: Neural network training settings and architecture configuration in Edge Impulse

Step 6: Evaluating the Model

After training, the model is tested and evaluated using the built-in testing tools in Edge Impulse. Specifically, evaluation is performed through the Model Testing section accessible from the left-side menu of the Edge Impulse interface.

The performance can be measured by looking at accuracy, latency, and reliability. If the accuracy is low, students can collect more data or retrain the model with improved image variety.

This iterative process helps the system reach higher performance standards suitable for industrial

applications. This also gives students the opportunity to gain exposure to real world problem solving techniques.

The images we used for testing were also correctly labeled with “nail” and “nut” as shown in Figure 13. The training results show that we achieved an accuracy of 88.24%, as shown in Figure 14.

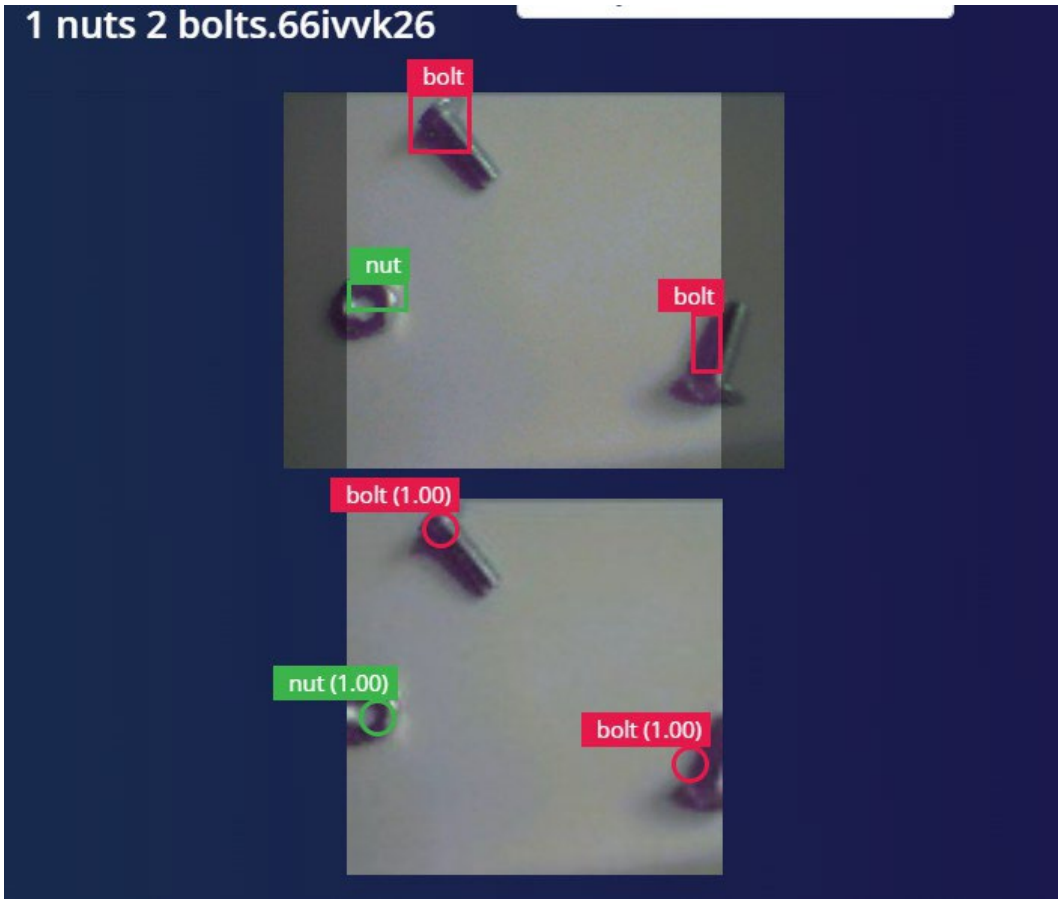

Figure 13: Testing Example

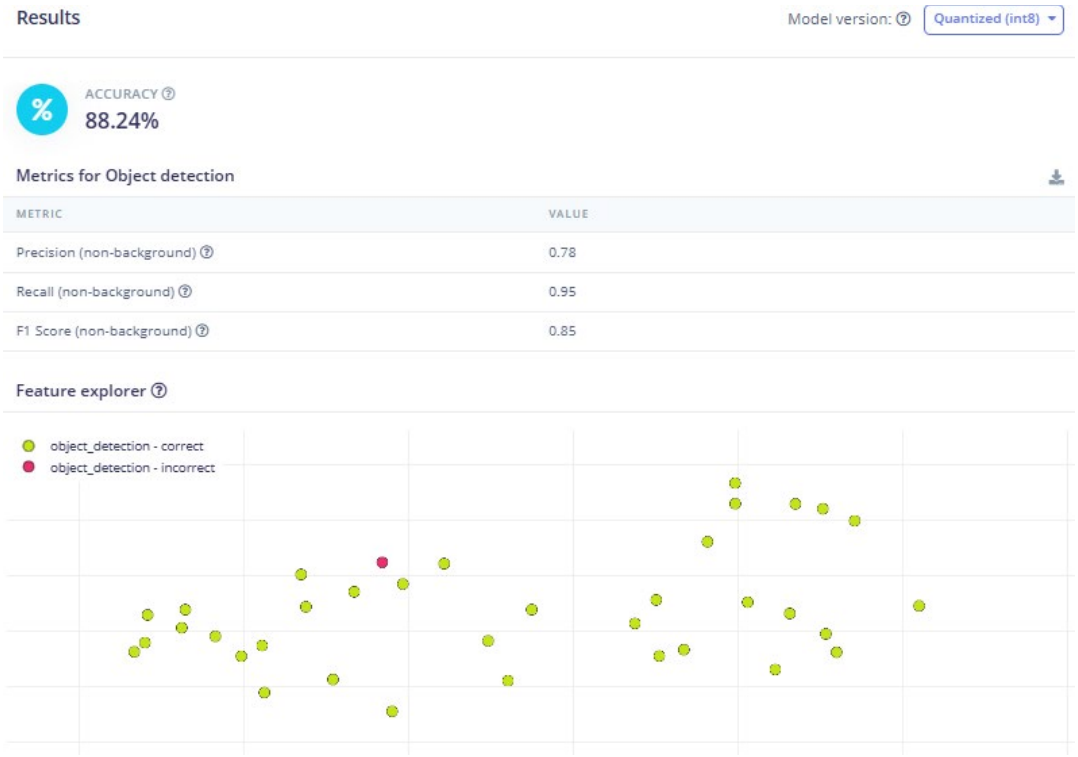

Figure 14: Training performance and object detection accuracy of the proposed model

Step 7: Deploying the Model to the Edge Device

The Arduino Nano 33 BLE Sense (nRF52840) has only 256 kB of SRAM and 1 MB of Flash, so memory efficiency becomes a key limitation for on-device inference. To deploy trained neural network models on such resource-limited edge devices, we need compilers to convert the models into executable code that can run efficiently on embedded hardware. Edge Impulse provides TensorFlow Lite and the EON Compiler as two deployment options.

TensorFlow Lite runs neural network models using a general computation graph. It executes operations layer by layer and supports FP32 and INT8 quantized models. The EON Compiler analyzes the neural network and the target hardware together, then generates hardware-aware and highly optimized inference code. TensorFlow Lite focuses on portability, while the EON Compiler focuses on efficiency for resource-constrained devices.

Compared to FP32 representations, INT8 quantization uses 8-bit integers for weights and activations, significantly reducing RAM and Flash consumption with only minimal impact on accuracy, making it better suited for resource-constrained microcontrollers [16].

Using standard TensorFlow Lite with INT8 quantization, the deployed model achieved an accuracy of 88.24%, but required approximately 123.1 kB of RAM and 110.5 kB of Flash, with an inference latency of about 359 ms. These results are approaching the practical limits of the Arduino Nano 33 BLE Sense. In contrast, deployment using the EON Compiler maintained the same accuracy of 88.24% while reducing RAM usage by approximately 17% and Flash usage by

approximately 36% compared to TensorFlow Lite. The EON compiled model required around 106 kB of RAM and 70.8 kB of Flash, making it significantly more suitable for the Arduino Nano 33 BLE Sense. These reductions are achieved through hardware-aware compiler optimizations that generate highly efficient inference code tailored to the target device [17].

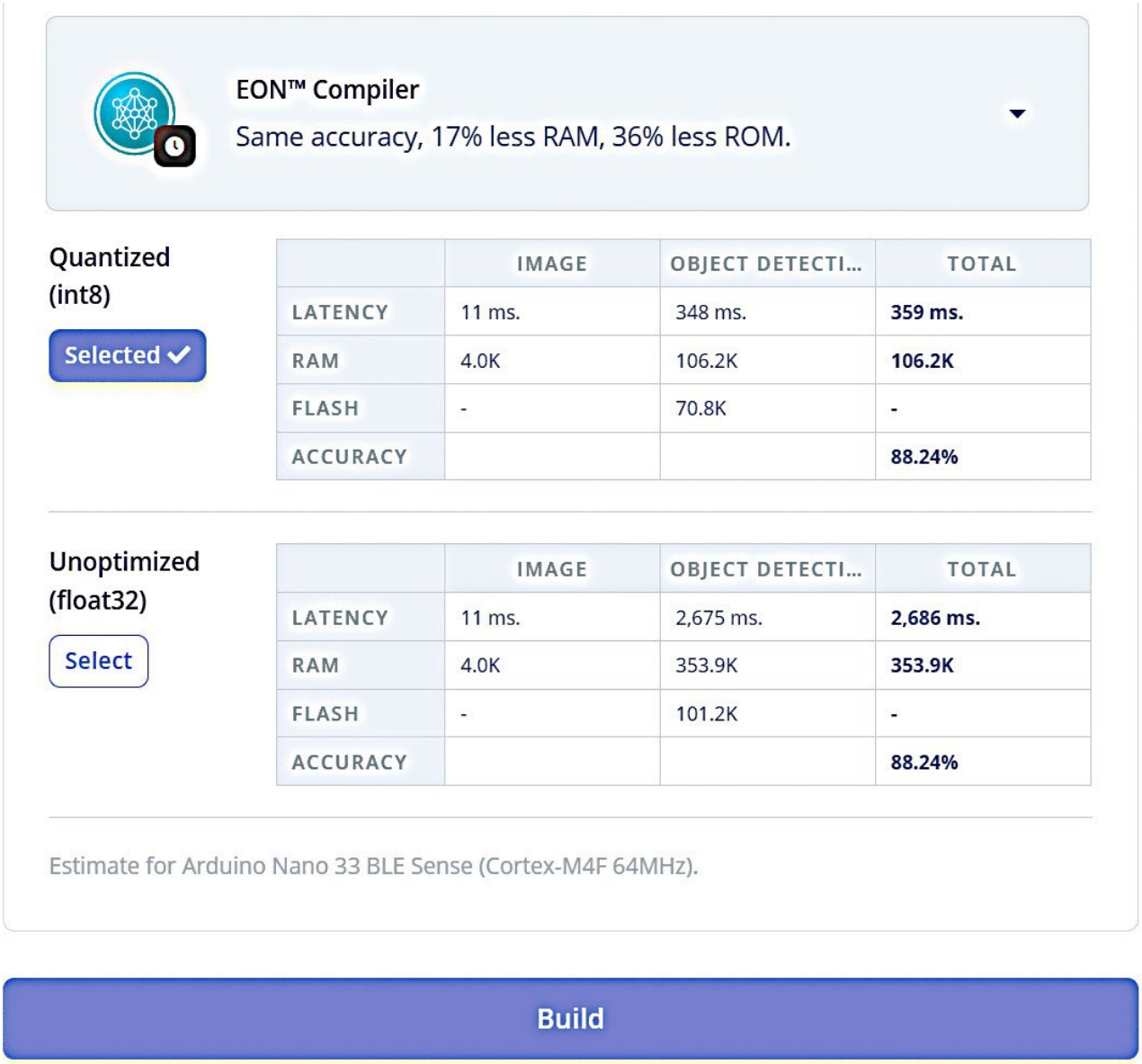

Figure 15: EON Compiler deployment settings and on-device performance metrics

The practical deployment workflow begins by selecting the Deployment option from the left menu in the Edge Impulse interface. The EON Compiler is chosen, for the deployment target we select our board (Nano 33 BLE Sense) and the INT8 quantized model is chosen for optimization. After configuration, Edge Impulse generates an Arduino library containing the compiled model along with all necessary inference code and dependencies.

The deployment configuration and on-device performance comparison using the EON Compiler are shown in Figure 15, which illustrates the significant memory savings achieved through compiler optimization while maintaining detection accuracy.

To flash the optimized firmware onto the Arduino Nano 33 BLE Sense, the Arduino Command Line Interface (Arduino-CLI) is installed on the development machine. This tool provides programmatic access to board management, library installation, and firmware uploading capabilities. The deployment process involves connecting the Arduino board via USB, and flashing the Arduino board using the generated Edge Impulse library. The firmware is now uploaded on the to board and running the model is as simple as running edge-impulse-run-impulse command on the terminal. This streamlined workflow eliminates the need for manual configuration and ensures that the model runs correctly within the resource constraints of the target hardware.

Figure 16 demonstrates the real-time object detection performance of the deployed model on the Arduino Nano 33 BLE Sense.

```
Starting inferencing in 2 seconds...
Taking photo...
Timing: DSP 9 ms, inference 325 ms, anomaly 0 ms
Object detection bounding boxes:
  nut (0.996094) [ x: 35, y: 7, width: 7, height: 14 ]
  bolt (0.992187) [ x: 7, y: 14, width: 14, height: 7 ]
  bolt (0.996094) [ x: 21, y: 28, width: 14, height: 7 ]
  nut (0.996094) [ x: 42, y: 28, width: 7, height: 7 ]
Starting inferencing in 2 seconds...
Taking photo...
Timing: DSP 9 ms, inference 325 ms, anomaly 0 ms
Object detection bounding boxes:
  nut (0.996094) [ x: 35, y: 7, width: 7, height: 14 ]
  bolt (0.992187) [ x: 7, y: 14, width: 14, height: 7 ]
  bolt (0.996094) [ x: 21, y: 28, width: 14, height: 7 ]
  nut (0.996094) [ x: 42, y: 28, width: 7, height: 7 ]
Starting inferencing in 2 seconds...
Taking photo...
Timing: DSP 9 ms, inference 325 ms, anomaly 0 ms
Object detection bounding boxes:
  nut (0.996094) [ x: 35, y: 7, width: 7, height: 14 ]
  bolt (0.976562) [ x: 7, y: 14, width: 14, height: 7 ]
  bolt (0.996094) [ x: 21, y: 28, width: 14, height: 7 ]
  nut (0.996094) [ x: 42, y: 28, width: 7, height: 7 ]
```

Figure 16: Real-time object detection output showing three consecutive inference cycles

5 Results and Discussion

The proposed Edge AI system demonstrates remarkable performance in resource-constrained environments, achieving an F1 score of 89.1. The F1 score is automatically computed and reported by the Edge Impulse and represents the harmonic mean of precision and recall. This performance is particularly impressive given that the system operates on approximately a quarter of a megabyte of memory, highlighting the effectiveness of aggressive optimization strategies for edge deployment. This achievement has significant implications for the broader field of embedded machine learning, as it demonstrates that sophisticated computer vision tasks can be accomplished on hardware with severely limited resources, opening pathways for widespread deployment in cost-sensitive industrial and IoT applications.

When we inspect Figure 16 further, output shows three consecutive detection cycles, each inference cycle takes approximately 2.33 seconds, including a two-second initialization delay and about 334 ms for on-device inference. The timing breakdown reveals that digital signal processing (DSP) operations such as image preprocessing and feature extraction require only 9 ms, while the model inference itself takes 325 ms per frame. Notably, the anomaly detection time is reported as 0 ms, indicating that no anomaly detection layer was active during these detections. This means the system only performs object detection and does not include any additional anomaly analysis.

The three consecutive inference cycles take approximately 7 seconds in total, the model consistently detects four objects with remarkably high confidence scores. Two nuts are identified at positions (35, 7) and (42, 28) with confidence levels of 0.996094, while two bolts are detected at positions (7, 14) and (21, 28) with confidence scores of 0.992187 and 0.996094 respectively. The minor variation in the second bolt's confidence score during the third cycle (0.976562) demonstrates that while the model maintains stable detection across frames, natural variations in lighting or minor camera movements can slightly affect confidence levels without compromising detection reliability.

The bounding box coordinates provide precise localization information for each detected object. For example, the first nut occupies a region with width 7 pixels and height 14 pixels starting at coordinate (35, 7), while the bolts are detected with bounding boxes of 14 by 7 pixels. These dimensions reflect the actual size and aspect ratio differences between nuts and bolts in the captured images. The reported coordinates represent the top-left corner of each bounding box, while the width and height values define the spatial extent of the detected region. These precise positional outputs enable downstream applications such as robotic manipulation or automated quality control to accurately locate and interact with the detected hardware components.

The consistency of detections across multiple inference cycles, with identical object positions and near-identical confidence scores, confirms that the model exhibits stable performance when observing a static scene. This repeatability is essential for industrial applications where reliable detection under consistent conditions is required. The total inference time of approximately 334 ms per frame (9 ms DSP plus 325 ms inference) translates to a throughput of roughly 3 frames per

second, which is adequate for many embedded vision applications involving stationary or slow-moving objects in manufacturing or inspection scenarios.

Despite these achievements, the system faces notable hardware limitations that constrain its potential performance. The primary bottleneck is RAM availability, which restricts the complexity of models that can be deployed and the quality of the input images. Additionally, the camera sensor exhibits some latency when adapting to lighting changes, requiring several seconds to adjust. This delay can impact real-time detection in dynamic environments where illumination varies rapidly. These constraints underscore the fundamental trade-offs inherent in ultra-low-power edge computing. Importantly, these findings suggest that even modest hardware improvements, such as doubling available RAM or upgrading to a more responsive camera sensor, could yield substantial performance gains, potentially pushing detection accuracy well beyond 95% while maintaining the low-power profile essential for edge deployment.

Model optimization techniques, including INT8 quantization and the EON Compiler, were essential in reducing memory footprint while maintaining detection accuracy. However, this compression inevitably involves trade-offs. While increased model complexity could potentially improve performance, the corresponding rise in RAM and Flash requirements would exceed the capabilities of the microcontrollers at hand. The result represents near-optimal performance given these hard hardware constraints, demonstrating that the system is operating at the theoretical limits of the current platform.

Environmental factors also significantly influence system robustness. Training data was deliberately collected under varied lighting conditions to enhance generalization. Nevertheless, other variables such as background complexity, object occlusion, spatial arrangement, and camera viewing angles remain potential sources of performance degradation. The consistency of detection results across test scenarios suggests the model has achieved reasonable robustness, though future work could explore active learning approaches to continuously adapt to new environmental conditions.

These findings demonstrate that despite severe resource limitations in both memory capacity and sensor responsiveness, carefully optimized Edge AI systems can achieve production-grade performance for industrial applications. The success depends critically on the synergy between model architecture design, quantization strategies, diverse dataset curation, and realistic assessment of hardware capabilities.

From an educational perspective, the primary strength of the proposed system lies in its simplicity and efficiency. The entire Edge AI pipeline is implemented using a compact and accessible framework, allowing key concepts to be demonstrated without the need for high-end hardware or complex configurations. The use of lightweight models, together with aggressive optimization techniques, enables a clear and practical illustration of the trade-offs between model accuracy and computational constraints.

Overall, the proposed system provides a simple, efficient, and practical framework that can be used to explain real-world Edge AI deployment in undergraduate engineering education.

References

[1]T. Meuser, L. Love´n, M. Bhuyan, S. G. Patil, S. Dustdar, A. Aral, S. Bayhan, C. Becker, E. d. Lara, A. Y. Ding, J. Edinger, J. Gross, N. Mohan, A. D. Pimentel, E. Rivie`re, H. Schulzrinne, P. Simoens, G. Solmaz, and M. Welzl, "Revisiting edge ai: Opportunities and challenges," IEEE Internet Computing, vol. 28, no. 4, pp. 49–59, 2024.

[2]W. Sun and X. Gao, "The construction of undergraduate machine learning course in the artificial intelligence era," in 2018 13th International Conference on Computer Science & Education (ICCSE), Colombo, Sri Lanka, 2018, pp. 1–5.

[3]D. M. Waqar, T. S. Gunawan, M. A. Morshidi, and M. Kartiwi, "Design of a speech anger recognition system on arduino nano 33 ble sense," in 2021 IEEE 7th International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA), Bandung, Indonesia, 2021, pp. 64–69.

[4]V. A. Wardhany, Subono, A. Hidayat, S. W. Utami, and D. S. Bastiana, "Arduino nano 33 ble sense performance for cough detection using a neural network classifier," in Proceedings of the 6th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE), Yogyakarta, Indonesia, 2022, pp. 455–458.

[5]D. Abhinay, S. V. Vighnesh, L. K. Durgam, and R. K. Jatoth, "Real-time classification of vehicle logos on arduino nano ble using edge impulse," in 2023 4th International Conference on Signal Processing and Communication (ICSPC). Coimbatore, India: IEEE, 2023, pp. 316–320.

[6]J. P. B. Lima, "Real-time fault detection in induction motors using tinyml: An evaluation of the Edge Impulse platform," in 2025 IEEE Latin Conference on IoT (LCIoT). Fortaleza, Brazil: IEEE, 2025, pp. 31–34.

[7]E. A. E. Arthur, F. A. Wulnye, D. A. N. Gookyi, K. O.-B. O. Agyekum, P. Danquah, and R. Gyaang, "Edge Impulse vs TensorFlow: A comparative analysis of TinyML platforms for maize leaf disease identification," in 2024 Conference on Information Communications Technology and Society (ICTAS). Durban, South Africa: IEEE, 2024, pp. 1–6.

[8]H. Badave and M. Kuber, "Performance analysis of light illuminations and image quality variations and its effects on face recognition," in 2021 International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT). Bangalore, India: IEEE, 2021, pp. 28–33.

[9]S. Wahwah and Y. Chen, "My Eye AI: Object Detection System Using YOLO, OWL-ViT, and BLIP," in Proceedings of the 16th IEEE International Symposium on Autonomous Decentralized Systems (ISADS). Tucson, AZ, USA: IEEE, 2025, pp. 1–8.

[10] T. Gerdprasert and S. Mabu, "Object detection for chest x-ray image diagnosis using deep learning with pseudo labeling," in 2021 IEEE 12th International Workshop on Computational Intelligence and Applications (IWCIA). Hiroshima, Japan: IEEE, 2021, pp. 1–5.

[11] M. Patil et al., “Edge impulse: Tinyml language classification model,” in 2024 4th Asian Conference on Innovation in Technology (ASIANCON), Pimari Chinchwad, India, 2024, pp. 1–6.

[12] T. Devanshi and R. Gajjar, “Detection and classification of mangoes using machine learning on various edge ai devices in real-time through drone,” in 2025 International Conference on Advancements in Power, Communication and Intelligent Systems (APCI), Kannur, India, 2025, pp. 1–6.

[13] R. Jamiah, I. Nurtanio, and A. Achmad, “Implementation of mobilenetv2 ssd fpn-lite cnn model for real-time detection spinach leaf diseases,” in 2023 International Conference on Artificial Intelligence Robotics, Signal and Image Processing (AIRoSIP), Yogyakarta, Indonesia, 2023, pp. 310–315.

[14] K. Omer and A. Monteriu`, “Edge ai for object recognition in digital twins: Enhancing indoor navigation and bim systems for living environments,” in 2025 IEEE International Workshop on Metrology for Living Environment (MetroLivEnv), Venezia, Italy, 2025, pp. 490–494.

[15] Y. S. H. Annadata, V. Moganarengam, and T. Nikoubin, “Tinyml powered drone in agriculture application,” in Proceedings of the 2024 IEEE 17th Dallas Circuits and Systems Conference (DCAS), Richardson, TX, USA, 2024, pp. 1–6.

[16] M. Z. M. Shamim, “Hardware deployable edge-AI solution for prescreening of oral tongue lesions using TinyML on embedded devices,” IEEE Embedded Systems Letters, vol. 14, no. 4, pp. 183–186, Dec. 2022.

[17] M. S. Diab and E. Rodriguez-Villegas, “A tinyml motion-based embedded cough detection system,” in Proc. 46th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Orlando, FL, USA, 2024, pp. 1–4.