Benchmarking EdgeAI platforms using Arduino Nano 33 BLE

Rohitashva S. Jhala and Jan Cordes

[1] University of Oldenburg, Oldenburg, Lower Saxony, Germany
rohitashva.jhala@uni-oldenburg.de
[2] Deutsches Zentrum für Luft- und Raumfahrt (DLR), Oldenburg, Lower Saxony, Germany
jan.cordes@dlr.de

Abstract

Edge AI enables machine learning models to operate directly on resource-constrained devices such as microcontrollers, allowing for real-time data processing and enhanced data privacy. This paper benchmarks multiple Edge AI frameworks, the evaluation spans five key metrics: accuracy, memory footprint, inference time, power consumption and usability. The platforms are tested using a human-motion classification dataset and deployed on an Arduino Nano 33 BLE board. Results indicate significant variations in framework performance across these metrics, providing insights into selecting the most suitable frameworks based on specific application requirements. This research highlights the strengths and weaknesses of each framework, helping developers make more informed decisions when choosing Edge AI solutions.

1 Introduction

In an increasingly interconnected world, the ability to process information at the source, namely, the device itself has become a critical challenge [13]. The concept of Edge AI is to decentralize data processing and machine learning from cloud-based infrastructures and high-performance computers to the very source, where the data is actually generated i.e. the sensor module itself [14]. This has the potential to extend the scope of Artificial Intelligence (AI) applications to areas previously considered limited, from wearable health monitors to smart sensors in remote locations and farming, the implications for daily life are vast and varied. These devices typically have limited computational resources and are under significant constraints in terms of memory and power supply as they are usually running on batteries or power harvesting.

Different frameworks have come up over the last years to make developing such Edge AI devices easier. They typically include the process of development, training, deployment, and optimization of machine learning models on these edge devices. In this work, we are comparing four Edge AI frameworks in five domains: accuracy, memory footprint, inference time, power consumption and usability. We therefore set up all four frameworks with an identical dataset and nearly identical configurations for the model to be generated. As a target edge device, we chose the Arduino Nano 33 BLE board because of its widespread adoption and inbuilt inertial measurement unit (IMU) sensor [1].

This work is structured in the sort review of related work following the explanation of the concept and the methodology to establish comparability between the frameworks. In the Concept chapter also the dataset used and the common data preprocessing as well as the gathering of metrics to compare the frameworks are explained. The implementation chapter describes how the different frameworks have to be set up and what hurdles have to be overcome to make them comparable. The Evaluation chapter gives the results of the measured metrics and gives some additional reasoning for the determined usability score. Finally, in the Conclusion

we sum up our results and give some recommendations on which framework is suitable for which use case.

2 Related Work

Osman et al. [11] did some similar work by comparing two popular frameworks: TensorFlow Lite Micro and CUBE AI. The aim was to compare the two for different TinyML applications and determine their use cases. They concluded CUBE AI is better suited for memory-limited and power-intensive TinyML applications, while TFLM offers wide availability and support for devices. Similarly, there is a need for comparing/benchmarking various other frameworks too to assess a suitable fit w.r.t. the applications as mentioned by Osman et al. in the future work.

Likewise, the article by Pratim Ray [12] presented current understanding of TinyML, the existing toolset, key enablers of the TinyML paradigm, state-of-the-art frameworks use cases and challenges. They too emphasized the need for more researchers to come forward, and develop new benchmarks with new datasets to test these frameworks and understand the use cases better.

Although Osman and Ray's work is in the field of EdgeAI, our study differentiates itself in several key ways. Unlike Osman's work which compares only two EdgeAI frameworks with both tested on different hardware platforms lacking a common reference point for benchmarking, we created a standardized benchmarking approach by using a common hardware platform, ensuring a consistent reference point for all frameworks tested. In contrast to Ray's work which is more focused on theoretical enquiry and challenges instead of practical benchmarking of frameworks with an edge device, we combine both the theoretical and practical aspects. We use the System Usability Scale (SUS) [9] based on established usability principles to evaluate the usability of the frameworks, and also benchmark them across multiple metrics like accuracy, memory consumption, inference time, and power consumption for comparison.

3 Concept

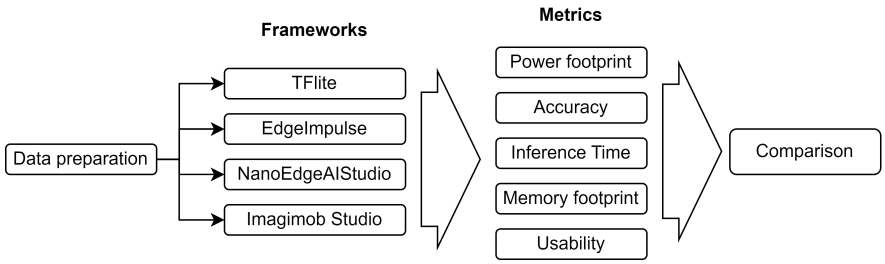

Figure 1: Concept behind the benchmarking of the frameworks.

Figure 1 shows the workflow and methodology underpinning the research. The data is used by the frameworks to train the model and once trained, the comparison is laid out across the array of metrics. In our case, the work begins with data processing where the dataset is divided into testing and training sets. Following this, the sliding window technique is employed to transform the raw inertial measurement unit (IMU) data into a series of windows [8]. Each window represents a single sample on which the neural network will be trained. The trained model is

then deployed onto the Arduino Nano 33 BLE board using the Arduino integrated development environment (IDE). The inference process also involves configuring the necessary input and output functionalities to enable the BLE board to perform real-time inference. The input functionality captures new data from the sensors, while the output functionality manages the prediction results generated by the model. Accuracy and memory consumption are measured before the deployment while the inference time, current consumption, and energy are measured after the deployment. While working with the frameworks we score their usability on a system usability scale so a comparison between the usability of the frameworks can be drawn.

3.1 Dataset and Data Preprocessing

As a typical dataset for this study we used the MotionSense Dataset [3], consisting of vibration data of human motion classification recorded by an IMU. The data, recorded at 50 Hz, offers 6 different activities: Walking, Sitting, Jogging, Standing, Upstairs, and Downstairs.

The data consists of time-series signals that capture the dynamic movements of the human body along three spatial axes (x, y, z) at 50 Hz. When trying to classify human activity, such as walking, jogging, sitting etc., it's beneficial to look at segments of data over a period of time rather than individual data points. For this, we use the sliding window technique where the data is divided into smaller segments called windows. By segmenting data into overlapping windows, we can capture the temporal dependencies which are crucial in sequential data [2]. In our work, we use the window length of 2.5 seconds or 125 data points as it has proven to be reliable [2]. Before creating sliding windows, each file containing the data is first divided into test and train sets and then windows are generated for each set. This methodology also ensures that there is no overlapping of the data between test and training sets.

3.2 Metrics

We evaluated various Edge AI frameworks by assessing their performance across six critical metrics: energy consumption, inference time, model accuracy, current consumption, memory usage, and usability.

To measure the **current consumption**, we use a digital multimeter to record the current observed when the model is making predictions (active) and when the device is idle.

To measure **inference time**, we recorded the execution period per classification i.e. from the moment an input is provided to the model until an output is received. This is done by using the Profiler library [4] which will output the time taken in milliseconds for one classification.

The **energy consumption** for every classification can simply be calculated by multiplying the current consumption value and inference value of the respective framework with the voltage, which in our case is 3.3 volts.

Accuracy measures the model's performance in correctly predicting the labels compared to the true labels. To measure accuracy, we consider the accuracy results were obtained by validating the trained model created within the corresponding framework using a separate test dataset.

Whereas for **memory consumption**, the Arduino IDE itself provides the information on the space occupied by the files used for performing inference. During the sketch upload process, the console output of the application reports the amount of flash memory and the RAM used when uploading the files to the Arduino device.

In our evaluation, we also wanted to assess the usability of each framework based on a set of established usability principles. For that, we chose the System Usability Scale (SUS) to incorporate a score-based assessment. SUS is a 10-question 5-response Likert scale which we

used for a subjective assessment of our opinions about the usability of these frameworks. These questions ask the user if the given framework can fulfil each aspect of the usability principles. Wherein, the response options vary from "Strongly Disagree" to "Strongly Agree", with 0 = Strongly Disagree to 4 = Strongly Agree. To convert the responses into a range of 0 to 100, for each framework, the responses were summed and multiplied by 2.5. This resulted in overall usability score of that framework.

Specific aspects of the usability principles to be examined include: **User Control and Freedom** [10] which includes the ability to configure the training process, neural network architecture, data preprocessing and resolution, **Clarity** [10] which includes clear description of the functionalities, **Help and Documentation** [5] including diversity in tutorials/examples and response time of community forum, and **Compatibility** [7] for the hardware device.

4 Implementation

Some frameworks lack the option for full control over the network design. Consequently, we sometimes had to opt for different network designs, for example: in the case of NanoEdge AI Studio as discussed in the section 4.2.3. We tried to keep these changes minimal and stick to a standard design. For that, we set up a baseline neural network architecture with Keras and TensorFlow. The deployment methodologies for the trained network also differ among the frameworks. Some, like Edge Impulse, produce a ready-to-use package library, while others generate C++ source code that we then add to an Arduino project. This code then must follow specific guidelines to interface the model on the hardware board.

4.1 Baseline architecture of Neural Network

In order to best compare the frameworks, a fixed architecture of a neural network must be followed across all the frameworks which we call the baseline model. This model serves as a foundational template ensuring consistency and comparability.

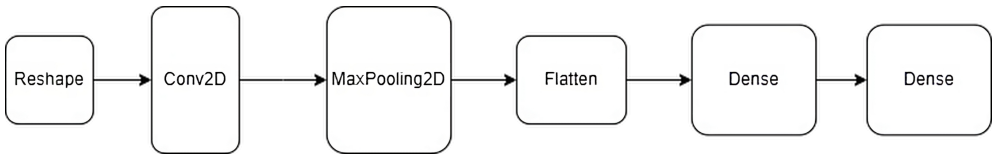

Figure 2: The baseline architecture of the Keras model

Figure 2 shows the architecture of the baseline model which was built using the Keras API in TensorFlow. The model is designed to operate on input data shaped as (125, 3), where 125 represents the number of time steps in a sliding window, and 3 corresponds to the three axes (X, Y, Z) of the IMU data.

This architecture and configuration of the neural network is the baseline model for every framework to follow.

4.2 Frameworks

4.2.1 TensorFlow Lite for Microcontrollers

TFLM being an open-source library with Python as an interfacing language, had the capability to follow the same neural network architecture design as mentioned in the section 4.1. Once the model was trained over the data, it was validated using the test dataset. Quantization is possible in TLFM, so the model was quantized using full integer quantization i.e. by converting the weights and activations from floating-point to 8-bit integer values. After this, the quantized model was converted into the .tflite format using TensorFlow's TFLite Converter. We used the "Hello World" example from TensorFlow Lite as a reference to incorporate the IO functionalities. The .tflite model then was converted into a C array with the xdd tool [1] to be used for performing inference. Once the C file of the model was created, we included it in our Arduino program for deployment on the device.

4.2.2 Edge Impulse

Edge Impulse is a web-based developing platform that provides the interface to upload the data, build and train the model, and deploy it onto an edge device. Edge Impulse follows a block-based design, beginning with uploading the dataset and splitting it randomly between test and training sets. After this, the sliding windows are generated and a classification model is chosen in the learning block with six features representing the categories of the human-motion dataset. Within the preprocessing block, we opted for the raw data option, choosing to process the data without any additional preprocessing steps following the guidelines from the baseline architecture. After training and testing in the framework, the trained model is quantized (int8) converting the weights and activations to 8-bit integer values. The framework built the quantized model into an Arduino library, which is then downloaded as a zip file. Once unzipped, It contains the trained model's parameters and an Arduino program for our BLE board, which can directly be deployed.

4.2.3 NanoEdge AI Studio

NanoEdge AI Studio is a free software developed by STMicroelectronics for Windows and Linux platforms. We fed the training dataset in the form of sliding windows as the Studio itself does not provide a tool for creating these. Once imported, the benchmark section runs an array of machine learning models online from its servers on the dataset to find the best performing one. The selection is done based on accuracy and memory consumption. Hence, the Studio does not provide any tool to access or alter the architecture of the model and the configurations of the training process. In our case, out of the best-performing models we chose Multi-Layer Perceptron (MLP) due to its similar architecture of interconnected multilayered neurons. Following the benchmark, the model was tested with the test dataset to ensure accuracy and then packed into a zip file for deployment. The zip file contains the Arduino program and the C source files. These C files contain the model's weights, function declarations, enumerations, macros, and other necessary definitions required for proper model integration. The main Arduino program includes the generated C source files as headers. Additionally, we add input/output functionalities to the Arduino program for sensor reading, pre-processing inputs, and printing the outputs of the prediction.

[1] xxd -i quantized_TFmodel.tflite > model.cc

4.2.4 Imagimob Studio

Imagimob Studio is a free software for the Windows platform where the training of the models is done over the cloud. The framework accepts the data in a very specific format [6] with two CSV files, one for the sensor data and the other for the label containing the "start time", "end time", and the "label name". This requires further pre-processing of the dataset. After uploading, the data is split into training, testing, and validation sets. For the model architecture and training configurations, we followed the same as specified in the baseline architecture 4.1. Imagimob performs the training process on its cloud servers which was speeding up the training process. Once trained, the Studio runs the model on validation and test sets and displays the accuracy and F1 scores. Following this, the trained model was converted into C/C++ source files using the inbuilt code generation tool. To deploy the generated files on the Nano BLE, we built an Arduino program performing IO functionalities for the model, similar to that in the NanoEdge AI Studio.

5 Evaluation

	Edge-Impulse (Enterprise)	TFLite for Microcontrollers	NanoEdge AI Studio	Imagimob Studio
Accuracy	88.58%	97.35%	94.80%	96.52%
Memory Usage-RAM (KB)	50.41	70.82	48.40	96.32
Memory Usage-Flash (KB)	156.15	359.20	128.21	845.85
Inference Time (ms)	21	20	36	56
Current Consumption (mA)	7.95	7.98	8.03	8.01
Energy Consumption (mJ)	0.50	0.47	0.86	1.34
Usability[2]	95	85	37.5	47.5

Table 1: Resulting metrics from the implementation of the frameworks.

5.1 Quantitative Results

As shown in the table 1, across the six key metrics, TFLM performs the best in terms of accuracy, inference time (per classification), and energy consumption (per classification). Although there is a similar current consumption figure across all the frameworks, the different inference times result in different figures for the energy consumption for a single classification. Hence, TFLM consumes the least energy per classification. As for the memory consumption, NanoEdge AI Studio utilizes the least Flash and RAM as compared to the rest of the frameworks.

5.2 Experimental results of Usability

In terms of usability scores as per system usability scale, as shown in the table 1, NanoEdge AI Studio follows the least of the usability principles mentioned in the section 3.2 while Edge Impulse adheres the most to these principles.

[2]The scores are derived from a system usability scale explained in detail in the section 3.2

TensorFlow Lite for Microcontrollers, in terms of User Control and Freedom, offers complete control to configure the training process, neural network architecture, and data preprocessing. With the support of numerous Python IDEs like VSCode, PyCharm, and Jupyter Notebook, the resolution of the application is highly configurable. It offers a wide range of tutorials and examples with a large and active community forum. It also offers support for a wide array of edge devices. However, in terms of clarity, the text describing the features and the deployment process is quite complex to understand at first.

Edge Impulse too provides control over the training process and the neural network architecture. However, it can only be accessed through Keras Expert Mode and is dependent on the output from its preceding layers. For data preprocessing, the framework provides an extensive array of tools and options like CSV Wizard. In terms of clarity, it offers a clear description of the features and options provided with the step-by-step navigation through the application interface. It also offers a wide range of examples/tutorials with deployment support for numerous edge devices. It has a large and active community offering a response time of 2 to 3 days, as observed.

NanoEdge AI Studio does not provide control over the training process and neither does it over the neural network architecture. It also does not support generating sliding windows from the dataset and does not allow any change in the resolution of the application. NanoEdge AI Studio offers limited clarity about the features and options, particularly in the benchmark section where no information on the configuration of the library is provided. It offers limited tutorials and no support was observed from the community on the posted queries. However, the range of officially supported devices is quite extensive and varied.

Imagimob Studio provides control to configure the training process, neural network architecture and generating sliding windows. The resolution of the application is configurable and can be resized according to preferences. Imagimob Studio offers limited clarity about the features and the options available, particularly for data upload and management, where the procedure to select the data and label files separately is not clearly described. However, the processes after the data upload are adequately explained. Imagimob Studio also offers limited tutorials and no support was observed from the community on the posted queries. It offers a wide range of supported edge devices.

6 Conclusion

The four frameworks were evaluated for bringing the neural networks on resource-constrained microcontroller devices, in our case Arduino Nano 33 BLE board. Out of them, TFLM performed the best in terms of accuracy, inference time, and energy consumption. While NanoEdge AI Studio showed the minimum overall memory usage, flash and RAM combined. In terms of current consumption, all of the frameworks showed nearly identical figures. However, the energy consumption showed which framework used the least versus the most energy per classification. From the usability perspective, Edge Impulse showed the best result followed by TFLM, Imagimob Studio, and NanoEdge AI Studio. As different frameworks excel at different metrics, the selection of the EdgeAI framework for a specific application should be guided by the priority of the metrics for that application.

7 Future Work

Expanding the range of evaluation metrics by incorporating additional metrics such as F1 scores for each label and performing live classification tests can provide deeper insights into the model's performance before and after deployment. Evaluation of MATLAB and uTensor could not be completed due to version compatibility issues and incomplete support for our BLE hardware. Resolving these challenges can provide more frameworks for comparison.

References

[1] Arduino nano 33 ble docs. https://docs.arduino.cc/hardware/nano-33-ble/. [Accessed 13-08-2024].

[2] Effects of sliding window variation in the performance of acceleration-based human activity recognition using deep learning models - PubMed — pubmed.ncbi.nlm.nih.gov. https://pubmed.ncbi.nlm.nih.gov/36091986/. [Accessed 13-08-2024].

[3] GitHub - mmalekzadeh/motion-sense: MotionSense Dataset for Human Activity and Attribute Recognition (time-series data generated by smartphone's sensors: accelerometer and gyroscope) (PMC Journal) (IoTDI'19) — github.com. https://github.com/mmalekzadeh/motion-sense. [Accessed 13-08-2024].

[4] GitHub - ripred/Profiler: Easily profile your Arduino functions to see how much time they take. The output can be disabled and enabled at runtime. Very lightweight. — github.com. https://github.com/ripred/Profiler. [Accessed 13-08-2024].

[5] Help and Documentation (Usability Heuristic 10) — nngroup.com. https://www.nngroup.com/articles/help-and-documentation/. [Accessed 13-08-2024].

[6] Imagimob Studio documentation — developer.imagimob.com. https://developer.imagimob.com/data-preparation/data-collection/bring-your-own-data. [Accessed 02-09-2024].

[7] ISO 25010 — iso25000.com. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010. [Accessed 13-08-2024].

[8] Sliding Window and Two Pointer Technique — linkedin.com. https://www.linkedin.com/pulse/sliding-window-two-pointer-technique-saiful-islam-rasel. [Accessed 27-08-2024].

[9] Usability form (SUS) for EdgeAI frameworks — docs.google.com. https://docs.google.com/forms/d/e/1FAIpQLScEr2S0LUd9lRLmFrMUZfQYUxy5fCXgV_1o1vdOCAbLDpNaag/viewform?vc=0&c=0&w=1&flr=0&usp=mail_form_link. [Accessed 28-08-2024].

[10] Usability Principles — improvement.stanford.edu. https://improvement.stanford.edu/resources/usability-principles. [Accessed 13-08-2024].

[11] Anas Osman, Usman Abid, Luca Gemma, Matteo Perotto, and Davide Brunelli. Tinyml platforms benchmarking, 2021.

[12] Partha Pratim Ray. A review on tinyml: State-of-the-art and prospects. *Journal of King Saud University - Computer and Information Sciences*, 34(4):1595–1623, 2022.

[13] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3:1–1, 10 2016.

[14] Raghubir Singh and Sukhpal Singh Gill. Edge ai: A survey. *Internet of Things and Cyber-Physical Systems*, 3:71–92, 2023.