ISSN 2007-9737

Arduino Devices as A Platform for Execution of Machine Learning Algorithms: A Brief Review and Experimentation

Juan José Flores Sedano, Hugo Estrada-Esquivel*, Alicia Martínez Rebollar

TecNM/Centro Nacional de Investigación y Desarrollo Tecnológico CENIDET, Mexico

{m20ce084, hugo.ee, alicia.mr}@cenidet.tecnm.mx

Abstract. At present, the Internet of Things and Artificial Intelligence are among the most relevant transformative technologies for making a smart world a reality. In this context, this paper explores the transformative synergy between the Internet of Things (IoT) and Artificial Intelligence (AI) by integrating AI algorithms into Arduino devices. The literature review has demonstrated a current need for optimization in implementing AI algorithms on Arduino platforms. Through a empirical literature review and practical experimentation, this paper provides a comprehensive analysis of several Arduino boards, including the Portenta H7 Lite, Arduino Uno, Wemos D1 ESP8266, and Arduino Nano 33 BLE, comparing their performance for AI projects. The selection of an IoT board is emphasized based on project-specific needs and budget considerations. The research presented in this paper reveals the impact of combining IoT, AI, and Arduino on reshaping interactions with the connected world, paving the way for intelligent systems enabled for decision-making and to execute complex tasks

Keywords. Artificial intelligence, Arduino platform, IoT

1 Introduction

The convergence of Artificial Intelligence (AI) and the Internet of Things (IoT) has emerged as a pivotal field that aims to explore the synergy between IoT and AI, fundamentally reshaping how we interact with our environment. This research work focuses on unlocking a realm of possibilities for intelligent decision-making and automation [1] by seamlessly integrating AI algorithms into Arduino devices. As the IoT revolution, redefining real-time connectivity between devices, sensors, and systems, is complemented by the convergence with AI, enhancing the learning and adaptive capabilities of these machines. The primary challenge lies in the effective implementation of AI on Arduino devices, where overcoming resource limitations such as memory and processing capacity is paramount [2]. This research not only conducts an empirical examination of existing literature on AI implementation in Arduino devices but also proposes additional optimizations to address these limitations and significantly enhance performance. In order to support the theoretical review, a comparative experimental study is conducted using various Arduino boards, such as the Portenta H7 Lite, Arduino Uno, Wemos D1 ESP8266, and Arduino Nano 33 BLE [3]. This experimental approach aims to evaluate the performance of these boards in AI projects, specifically by comparing their performance using the MNIST dataset. The review emphasizes the importance of choosing a suitable IoT board based on project needs and budget constraints.

This work not only sheds light on how the combination of IoT, AI, and Arduino is transforming interactions with the connected world but also lays the groundwork for intelligent systems capable of decision-making and executing intricate tasks [4]. The paper is structured as follows: Section 2 presents the literature review of the implementation of machine learning algorithms into the Arduino infrastructure. Section 3 shows experimentation of Arduino as a platform to execute machine learning algorithms, and finally, Section 4 details the conclusions and future work.

2 Literature Review

In this section, an analysis of research works exploring the combination of Artificial Intelligence and Arduino reveals a wide range of approaches aiming to bring machine learning capabilities to these low-power, resource-constrained devices. The classic Arduino Uno has served as a popular experimentation platform, despite its limited memory and computing power. Various researchers have implemented everything from simple neural networks for pattern recognition [5] to handwritten digit classifiers [6] on the Uno. While achieving promising accuracies, they are severely bottlenecked by the 2KB of RAM and limited CPU performance, leading to impractical execution times as model complexity increases.

To push the boundaries, researchers have turned to more powerful Arduino boards, such as the Due, with its 32-bit microcontroller and higher memory capacity. This has enabled the implementation of more complex neural network architectures for applications like fish quality detection from gas sensor data, achieving 80% accuracy [7]. However, memory and computing resources remain significant constraints.

The real game-changer has been the advent of Tiny Machine Learning (TinyML) and model optimization techniques. The Arduino Nano 33 BLE Sense, for example, is well-suited for TinyML. Researchers have successfully deployed voice and gesture recognition models on this device by aggressively quantizing and pruning neural networks [11]. However, these optimizations often come at the cost of reduced model precision and accuracy.

Work has also been done on using the Nano 33 BLE for ultrasonic signal classification to detect transducer misalignment [12]. The multilayer neural network model achieved top accuracy and was implementable on this low-power edge device, showing real-world potential. Nevertheless, running intensive ML workloads can quickly drain the battery of portable Arduino devices, and energy efficiency is rarely optimized for.

Beyond just inference, pioneering work has explored training models directly on IoT devices through approaches combining federated learning and transfer learning [13].

This enables on-device training without compromising privacy or relying on the cloud, although such capabilities are still limited.

Complementary work has also delved into hybrid approaches where heavy lifting occurs in the cloud, and simplified models are deployed on the Arduino for data capturing, preprocessing, and communication [14]. However, integrating different software components and environments can itself be challenging.

Table 1. Literature review comparison on Arduino uno devices

Reference	Aplications	Results	Limitations
[5]	C-Mantec neural network for pattern recognition	Exponential execution time, reduces precision	Memory restrictions
[7]	Real-time biosignal acquisition with MATLAB	Real-time visualization	Sampling rate 10Hz
[8]	Vibration analysis with wavelet and LabVIEW	Real-time processing	Requires multiple software, sampling rate limitation.
[9]	Vital signs telemetry	Functional and low cost	Potential temporary fluctuations
[10]	Environmental monitor and Android application	Remote viewing working	Limited sensors, limited sampling rate

Implementing AI on microcontrollers comes with its own set of challenges that existing solutions only partially address - squeezing models into limited memory, optimizing for inference speed, managing energy consumption, maintaining precision, and enabling on-device training with privacy preservation.

Overall, the research demonstrates the immense potential of taking AI to the edge, empowering intelligent sensing and decision-making capabilities in compact, low-cost IoT devices. Observations highlight key applications such as real-time vibration analysis [8], biofeedback and data acquisition [7], patient health monitoring [9], environmental sensing [10], and even face mask detection during COVID-19 [15].

Table 1 focuses on studies conducted using the Arduino Uno board, which has limited memory (2KB RAM) and computing power. The studies cover applications like pattern recognition using

Table 2. Literature review comparison on Arduino nano 33 BLE devices

Reference	Aplications	Results	Limitations
[11]	Gesture and voice recognition with TinyML	Functional models of keywords and gestures	Restricted sampling rate
[12]	ML algorithms for classifying transducer misalignment	Achieving superior classification performance metrics, including accuracy, precision, recall, and confusion matrices	Memory restrictions, Limited sensors,

Table 3. Literature review comparison of other devices

Reference	Applications	Results	Limitations
[13]	neural network for fish quality recognition	An 80% success rate was achieved in recognizing the quality of fish	Limited applicability
[14]	real-time face detection system	accuracy, with a rate of correctness of up to 97.80%	No limitations are mentioned
[15]	on-board training of ML algorithms on IoT devices.	an accuracy rate of 86.48% in classification and 0.0201 in regression.	No limitations are mentioned

neural networks [5], handwritten digit recognition [6], biosignal acquisition [7], vibration analysis [8], vital signs telemetry [9], and environmental monitoring [10]. While some promising results were achieved, such as 82% accuracy for digit recognition, the key limitations were exponential execution times, reduced precision, memory restrictions, low sampling rates, and the need for multiple software tools.

Table 2 summarizes research using the more capable Arduino Nano 33 BLE board, which is better suited for tiny machine learning (TinyML) applications. The studies cover gesture and voice recognition [11] and classifying transducer misalignment [12] using ML algorithms. Though functional models were developed, limitations included restricted sampling rates, memory constraints, and limited sensor capabilities.

Table 3 includes studies conducted on various other Arduino boards or setups. It covers applications like fish quality recognition using neural networks [13], real-time face detection [14], and on-board training of ML algorithms on IoT devices [15]. While high accuracies were achieved (up to 97.8% for face detection), the studies either had limited applicability or did not mention any specific limitations.

Analyzing existing works reveals a trend where Arduino devices are primarily utilized for data collection, with subsequent processing and analysis occurring on external devices. While a few studies showcase the direct implementation of neural networks on Arduino infrastructure, the predominant approach involves utilizing Arduino as a data collector. However, important limitations still exist- memory constraints, compute power bottlenecks, trade-offs in model optimization, energy efficiency issues, lack of scalable on-device training, systems integration challenges, and the need for more general, easily extensible solutions across different AI problems and hardware. Comprehensively addressing these challenges will be crucial for AI to become truly ubiquitous on Arduino and embedded devices. This insight prompts the exploration of whether Arduino devices alone can execute AI techniques in real-time without relying on external servers or PCs.

Arduino boards, while versatile, face significant limitations when implementing ML algorithms. Their constrained RAM (often below 1MB), limited processing power, and lack of dedicated AI accelerators make them unsuitable for complex deep learning tasks. Additionally, power consumption and real-time execution challenges limit their deployment in continuous AI-driven applications.

3 Experimentation: Arduino as a Platform for Implementing Machine Learning Algorithms

The experimentation approach proposed in this research work has the objective of comparing the performance of some of the most popular IoT development boards available on the market. In the experimentation it is tried to explain how these boards, such as the Arduino Nano 33 BLE, Portenta H7 Lite, Arduino Uno R3 and Wemos D1 ESP8266, address the challenge of classifying handwritten digits using machine learning techniques. To accomplish this task, the MNIST dataset was used, which consists of thousands of images of handwritten digits and their respective labels.

In the experimentation, the behavior of each of the Arduino boards was analyzed to execute artificial intelligence tasks using the same reference dataset. Each board has its own strengths and limitations, and understanding how they compare in the context of AI can be instrumental in making informed decisions when selecting the right platform for each project. The Arduino Nano 33 BLE, the Portenta H7 Lite, the Arduino Uno R3 and the Wemos D1 ESP8266 are analyzed below.

In order to evaluate the performance of classification models and each Arduino board, the following techniques were implemented in the Arduino boards: Convolutional Neural Networks (CNN), Decision Trees, and Clustering with K-Means. The chosen dataset for this analysis was the well-known MNIST dataset, comprising 28x28 pixel black and white images, each labeled with a corresponding digit. The researchers allocated 80% of the dataset for training purposes, reserving the remaining 20% for testing and assessing model accuracy [16].The selection of models and hyperparameters was based on computational constraints. CNNs were chosen for their effectiveness in image recognition, while Decision Trees and K-Means were included due to their lower resource requirements. Parameters such as the number of convolutional layers, tree depth, and cluster numbers were adjusted to balance performance and efficiency on each board.

Before running the machine learning algorithms and comparing their performance on different Arduino boards, it is essential to perform a thorough exploration of the MNIST dataset that will be used for experimentation. This data exploration plays a critical role for several reasons: Deep understanding of the data set: Exploration permits to obtain a solid understanding of the characteristics, structure, distributions and patterns that can be found in the data. This is essential to formulate realistic assumptions and expectations about the performance of the models and algorithms that will be applied later.

- Early problem detection: During exploration, potential problems such as outliers, missing data, biases, noise, or data inconsistencies can be identified. Addressing these issues early prevents them from propagating and negatively affecting model performance.
- Informed selection of techniques and preprocessing: Understanding the nature of the data permits the selection of the most appropriate machine learning techniques and algorithms, as well as determine the transformations or preprocessing necessary to optimize performance.
- Establishing baselines: The results of data exploration, such as summary statistics, distributions, and visualizations, establish important baselines for later evaluating and comparing model performance.

Analyzing existing works reveals a trend where Arduino devices are primarily utilized for data collection, with subsequent processing and analysis occurring on external devices. While a few studies showcase the direct implementation of neural networks on Arduino infrastructure, the predominant approach involves utilizing Arduino as a data collector. However, important limitations still exist- memory constraints, compute power bottlenecks, trade-offs in model optimization, energy efficiency issues, lack of scalable on-device training, systems integration challenges, and the need for more general, easily extensible solutions across different AI problems and hardware. Comprehensively addressing these challenges will be crucial for AI to become truly ubiquitous on Arduino and embedded devices. This insight prompts the exploration of whether Arduino devices alone can execute AI techniques in real-time without relying on external servers or PCs

Identification of valuable insights: Visual and quantitative exploration of data can reveal non-obvious patterns, trends or relationships that could be valuable to the modeling process and interpretation of results. In the specific context of this study, exhaustive exploration of the MNIST data set prior to experimentation on Arduino boards is crucial for several reasons:

- Understand the statistical properties of handwritten digit images and their labels, which will inform the configuration and expectations of classification models.
- Identify potential challenges or limitations that could arise when implementing these models on the restricted resources of Arduino boards, such as image resolutions, pixel value ranges, etc.
- Establish initial benchmarks and metrics to evaluate the performance of algorithms in terms of accuracy, memory usage, and other relevant aspects.
- Achieve a visual understanding of the variations, styles and patterns present in digit images, which could influence the performance of the models.

Data exploration is a critical step before conducting experimentation, as it provides a solid understanding of the data set, helps identify potential problems, facilitates the selection of appropriate techniques, and establishes crucial benchmarks for subsequent evaluation of the models. in a resource-constrained environment like Arduino boards. The experimental approach was consistent with the data exploration methodology detailed in the referenced literature [17]. The goal is to thoroughly understand a dataset first through a mix of quantitative analysis and visual inspection techniques before developing models. This builds intuition, sets modeling expectations, and allows detecting data issues early. The data exploration process provided a comprehensive understanding of the MIST dataset. With no missing values and a balanced digit distribution, the dataset appears well-suited for training robust digit recognition models. Insights gained from visualizing sample images will inform subsequent analyses and model development.

The following steps can be used to perform data exploration:

- Loading the Dataset: The first step is to load the image dataset and any associated labels or outputs. For MNIST, this consists of 70,000 small 28x28 grayscale digit images, each with a corresponding digit label from 0-9 indicating which number it shows.

- Understanding Structure: Investigate general metadata like the number of images, resolution per image, data formats, and how labels are encoded. This orients to key structural aspects.

- Summarizing Statistics: Calculate summary statistics per image and per class label when available. These include basics like mean or average pixel intensity, standard deviations from the means, min and max values, quantiles showing value distributions, etc. The summarizing statistics of the MNIST dataset can be seen in figures 1 and 2. 1 and 2. Figure 1 shows various statistics calculated for the full set of 70,000 handwritten digit images from the MNIST dataset. The first row with all values set to zero corresponds to the statistical summaries for missing data or null values. Since there is no missing data in this set, all of these cells are zeros. The following rows show multiple summaries per variable or characteristic:

  - The "count" row indicates the total count of non-null values per column, in this case 70,000 for all columns.

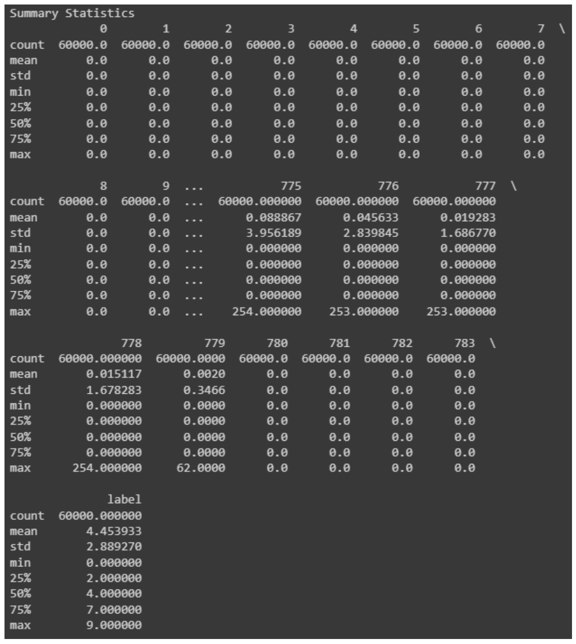
  - "mean" shows the average intensity of pixels per column over all images.

  - "std" is the standard deviation of pixel intensities, a measure of how much the values vary from the mean.

  - "min" and "max" show the minimum and maximum intensity values in the entire data set.

  - Rows like "25%", "50%", etc. are the quantiles of the distribution, indicating that 25% of the values are below that threshold.

  - The last row with only the total count serves to verify that there is no missing data.



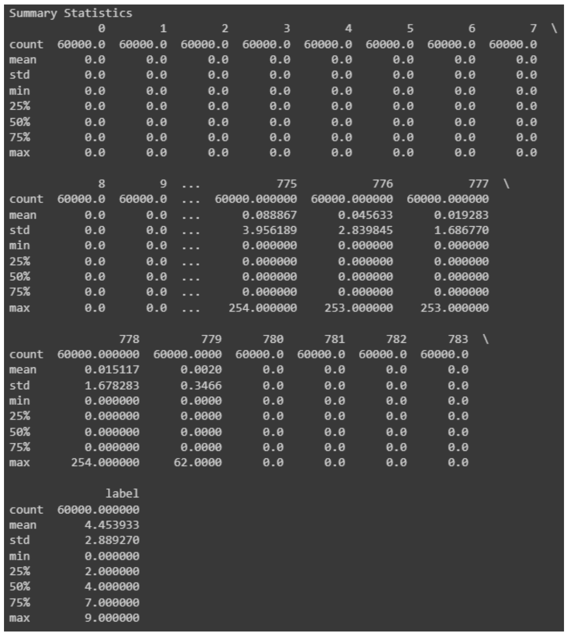
```
Summary Statistics
             0        1        2        3        4        5        6        7  \
count  60000.0  60000.0  60000.0  60000.0  60000.0  60000.0  60000.0  60000.0
mean       0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
std        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
min        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
25%        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
50%        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
75%        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
max        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0

             8        9  ...           775           776           777  \
count  60000.0  60000.0  ...  60000.000000  60000.000000  60000.000000
mean       0.0      0.0  ...      0.088867      0.045633      0.019283
std        0.0      0.0  ...      3.956189      2.839845      1.686770
min        0.0      0.0  ...      0.000000      0.000000      0.000000
25%        0.0      0.0  ...      0.000000      0.000000      0.000000
50%        0.0      0.0  ...      0.000000      0.000000      0.000000
75%        0.0      0.0  ...      0.000000      0.000000      0.000000
max        0.0      0.0  ...    254.000000    253.000000    253.000000

                778         779      780      781      782      783  \
count  60000.000000  60000.0000  60000.0  60000.0  60000.0  60000.0
mean       0.015117      0.0020      0.0      0.0      0.0      0.0
std        1.678283      0.3466      0.0      0.0      0.0      0.0
min        0.000000      0.0000      0.0      0.0      0.0      0.0
25%        0.000000      0.0000      0.0      0.0      0.0      0.0
50%        0.000000      0.0000      0.0      0.0      0.0      0.0
75%        0.000000      0.0000      0.0      0.0      0.0      0.0
max      254.000000     62.0000      0.0      0.0      0.0      0.0

              label
count  60000.000000
mean       4.453933
std        2.889270
min        0.000000
25%        2.000000
50%        4.000000
75%        7.000000
max        9.000000
```

Fig. 1. Summary statistics Dataset MNIST



```
Summary Statistics
             0        1        2        3        4        5        6        7  \
count  60000.0  60000.0  60000.0  60000.0  60000.0  60000.0  60000.0  60000.0
mean       0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
std        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
min        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
25%        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
50%        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
75%        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
max        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0

             8        9  ...           775           776           777  \
count  60000.0  60000.0  ...  60000.000000  60000.000000  60000.000000
mean       0.0      0.0  ...      0.088867      0.045633      0.019283
std        0.0      0.0  ...      3.956189      2.839845      1.686770
min        0.0      0.0  ...      0.000000      0.000000      0.000000
25%        0.0      0.0  ...      0.000000      0.000000      0.000000
50%        0.0      0.0  ...      0.000000      0.000000      0.000000
75%        0.0      0.0  ...      0.000000      0.000000      0.000000
max        0.0      0.0  ...    254.000000    253.000000    253.000000

                778         779      780      781      782      783  \
count  60000.000000  60000.0000  60000.0  60000.0  60000.0  60000.0
mean       0.015117      0.0020      0.0      0.0      0.0      0.0
std        1.678283      0.3466      0.0      0.0      0.0      0.0
min        0.000000      0.0000      0.0      0.0      0.0      0.0
25%        0.000000      0.0000      0.0      0.0      0.0      0.0
50%        0.000000      0.0000      0.0      0.0      0.0      0.0
75%        0.000000      0.0000      0.0      0.0      0.0      0.0
max      254.000000     62.0000      0.0      0.0      0.0      0.0

              label
count  60000.000000
mean       4.453933
std        2.889270
min        0.000000
25%        2.000000
50%        4.000000
75%        7.000000
max        9.000000
```

Fig. 2. Continue Summary statistics Dataset MNIST

Figure 2 complements the previous table by showing additional columns with more statistics, such as skewness, kurtosis for each variable, and limits such as "range" which gives the interval between minimum and maximum values.

Visualizing Samples: Plot a sample of original images to get a visual sense of variability in styles, rotations, scales, deformations and noise in the data. This could expose limitations or challenges. See figure 3.

Analyzing Distributions: Visualize the distribution of pixel intensities across images via plots like histograms and box plots, to see the full range and typical values. Break this analysis down by class label too. Look for imbalanced data or odd distributions. See figure 4.

Detecting Outliers: Based on the distributions, quantitative metrics to detect and flag outlier images that are anomalous or especially noisy. This could affect model training.

In the pursuit of comprehensively evaluating the performance of classification models on diverse Arduino boards, a systematic methodology was diligently crafted, aligning with principles of data exploration [18-19]. Each step of the experimentation carried specific significance, contributing to the robustness and depth of the assessment. The following steps were carried out to perform the experimentation on each Arduino board and each of the machine learning algorithms.

- Dataset Partitioning: The initial step involved the meticulous partitioning of the MNIST dataset, dedicating 80% for training and reserving 20% for testing. This partitioning strategy was pivotal to furnish a robust evaluation of model performance, ensuring an effective gauge of their generalization capabilities.
- Model selection: the chosen models, Convolutional Neural Networks (CNN), Decision Trees, and K-Means Clustering Algorithm, were selected due to their relevance in classification tasks and distinct approaches to pattern recognition within the MNIST dataset. This step aimed at deploying models representing diverse methodologies to garner a comprehensive understanding of their effectiveness.
- Arduino Board Implementation: Tailoring the selected models for execution on specific Arduino boards, including Arduino Uno R3, Wemos D1 ESP8266, Arduino Nano 33 BLE,

ISSN 2007-9737

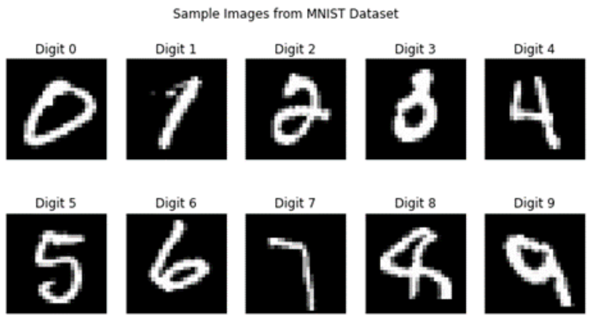


Fig. 3. Continue Summary statistics Dataset MNIST

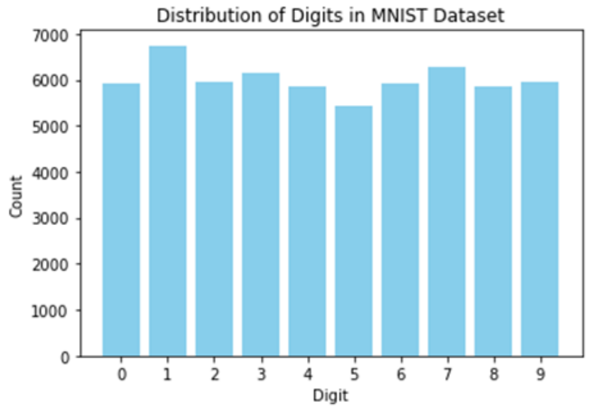


Fig. 4 . Distribution of digits MNIST dataset.

and Portenta H7 Lite, was imperative. Adapting the models to the unique constraints and capabilities of each hardware configuration allowed for a nuanced assessment in real-world embedded systems scenarios.

– Model Execution: The execution of models, including Convolutional Neural Networks (CNN), Decision Trees, and K-Means Clustering Algorithm, was executed meticulously on the designated Arduino boards. This step provided practical insights into the models' behavior, strengths, and potential limitations when applied to embedded systems.

– Accuracy Assessment: The accuracy assessment, performed using the 20% reserved test dataset, offered critical insights into the models' ability to generalize and make accurate predictions on previously unseen data. This step was fundamental in understanding the practical utility of the models in real-world applications.

– Resource Utilization Analysis: Beyond accuracy metrics, a detailed analysis of resource utilization, encompassing RAM and flash memory usage percentages on each Arduino board, was conducted. This granular examination aimed to ascertain how efficiently each model harnessed available resources, crucial for practical considerations in embedded systems.

– Result Compilation and Analysis: The results, consolidating accuracy metrics and resource utilization percentages, were meticulously compiled and analyzed. This final step facilitated a comparative evaluation of the models' effectiveness across different Arduino boards, providing nuanced insights for deploying AI applications in embedded systems.

This structured and systematic approach ensured not only a thorough exploration of the models' performance but also provided valuable context for their applicability in resource-constrained environments.

4 Results of the Experimentation

This section aims to present and analyze the key findings obtained from the comprehensive experimental evaluation performed on various Arduino boards. This section systematically examines the performance metrics and resource utilization characteristics exhibited by the machine learning models (Convolutional Neural Networks, Decision Trees, and K-Means Clustering) across different Arduino platforms.

Model accuracy: The accuracy metric quantifies the capability of a machine learning model to correctly classify or predict instances within a dataset. In this study, model accuracy serves as a critical measure to assess the reliability and effectiveness of the implemented algorithms on the constrained Arduino environments.

The results indicate that the Portenta H7 Lite board consistently outperforms other boards in terms of model accuracy, making it a robust choice

Table 4. Results

Metric	Model	Arduino Uno	Wemos d1	nano 33 ble	H7 lite
Accuracy (%)	CNN	87.3	89.8	92.5	94.2
	K-means	65.5	68.4	72.2	78
	Decision tree	86.5	88.7	91	92.3
Ram usage (%)	CNN	79.6	65.3	58.2	71.8
	K-means	85.6	69.8	63.5	78.2
	Decision tree	38.9	42.3	45.8	51.4
Flash usage (%)	CNN	92.3	68.9	72.4	82.1
	K-means	73.8	52.1	57.2	64.5
	Decision tree	48.7	54.8	58.6	61.2

for deploying machine learning applications. Among the evaluated models, the Convolutional Neural Network (CNN) exhibits promising accuracy, ranging from 87.3% on the Arduino Uno R3 to an impressive 94.2% on the Portenta H7 Lite. This trend underscores the efficacy of CNN models in image classification tasks, even on resource-constrained devices.

Resource utilization: Efficient resource utilization is paramount when deploying machine learning models on embedded systems with limited computational capabilities and memory constraints. This study meticulously analyses the RAM and flash memory usage percentages incurred by each model across the evaluated Arduino boards. The results reveal varying degrees of resource consumption among the models and boards. For instance, the CNN model exhibits higher RAM usage, ranging from 79.6% on the Arduino Uno R3 to 71.8% on the Portenta H7 Lite. Conversely, the Decision Tree model demonstrates relatively lower RAM requirements, spanning from 38.9% to 51.4% across the boards.

Furthermore, an analysis of flash memory usage uncovers the efficiency with which models can be deployed on these embedded platforms. The CNN model, while utilizing 72.4% of flash memory on the Arduino Nano 33 BLE, showcases optimal efficiency on the Portenta H7 Lite, consuming only 82.1% of the available flash storage. Table 4 provides a comprehensive overview. Running ML models on low-power devices like Arduino introduces significant energy challenges. AI inference tasks increase power draw, which can be a critical factor in battery-operated IoT applications. Optimizations such as quantization and model pruning can help reduce power consumption but often come at the cost of accuracy.

One notable observation is the trade-off between model complexity and resource utilization. While the Convolutional Neural Network (CNN) model demonstrates superior accuracy, it comes at the cost of higher RAM and flash memory consumption across all Arduino boards. This trade-off becomes particularly evident when comparing the CNN model's resource demands to the more lightweight Decision Tree and K-Means Clustering models.

Interestingly, the Arduino Uno R3, despite being one of the more budget-friendly and resource-constrained boards, exhibited relatively efficient resource utilization for specific models. For instance, the Decision Tree model consumed only 38.9% of RAM and 48.7% of flash memory on this board, highlighting its potential for deploying less computationally intensive algorithms in resource-limited scenarios.

Another noteworthy observation is the discrepancy in resource utilization patterns between the Arduino Nano 33 BLE and the more powerful Portenta H7 Lite board. While the Nano 33 BLE showcased respectable accuracy levels, its resource consumption, particularly for the CNN model, was significantly higher compared to the Portenta H7 Lite. This discrepancy underscores the impact of hardware specifications on model performance and resource efficiency.

Furthermore, it is essential to consider the specific application requirements and resource constraints when selecting an Arduino board and machine learning model combination. For example, in scenarios where memory footprint is a critical concern, the Decision Tree or K-Means Clustering models may be more suitable choices, even if they sacrifice some accuracy compared to the CNN model.

Interestingly, the results also revealed potential optimization opportunities. For instance, the K-Means Clustering model exhibited relatively low resource utilization across all boards, suggesting that there might be room for further optimization or model compression techniques to reduce its memory footprint further.

These observations highlight the intricate interplay between model complexity, hardware capabilities, and resource constraints in the context of embedded machine learning systems. Striking the right balance between performance, accuracy, and resource efficiency requires careful consideration of these trade-offs and a thorough understanding of the application requirements and hardware limitations.

The observed results can be attributed to the hardware capabilities of each Arduino board. The Portenta H7 Lite, featuring a more powerful processor and greater memory, was able to efficiently execute AI models with higher accuracy and lower resource constraints. Meanwhile, the Arduino Uno R3, with its limited computational power, struggled to support complex models, leading to reduced performance. The Nano 33 BLE and Wemos D1 ESP8266 positioned themselves as balanced alternatives, providing decent accuracy with moderate resource usage. These findings emphasize the importance of selecting hardware that aligns with the computational demands of AI applications in embedded systems.

5 Conclusions and Future Work

In conclusion, this work underscores the pivotal role of integrating AI with IoT using Arduino devices. Through a comprehensive survey and experimentation, we have provided valuable insights into the performance of different Arduino boards in executing AI tasks. The comparative analysis reveals a spectrum of options, from high-performance but expensive boards like the Portenta H7 Lite to budget-friendly alternatives like the Arduino Uno. The Arduino Nano 33 BLE, with its balance of affordability and features, emerges as an ideal choice for AI enthusiasts. Future work includes settling whether or not the Arduino foundation is adequate to execute computer-based intelligence procedures without the need to utilize a PC or server, simply turning to the restricted assets that the Arduino boards have. The aim is to use AI techniques that allow the behavior of an Arduino device to be analyzed in real time and make decisions based on that behavior, such as determining what data it is collecting, encapsulating it and sending it to an IoT platform such as FIWARE or Amazon Web Services, facilitating the integration of Arduino devices to the IoT. Also, as future work, we plan to add more devices to the study, such as other Arduino devices, ESP boards, or even Raspberry Pi, to broaden the perspective and obtain a better comparison.

Acknowledgments

This work was supported by TecNM (México) project 21640.25-P. The first author received a CONACYT scholarship (1086749) for this research.

References

1. **Kasera, R. K., Gour, S., Acharjee, T. (2024).** A comprehensive survey on IoT and AI based applications in different pre-harvest, during-harvest and post-harvest activities of smart agriculture. Computers and Electronics in Agriculture, 216, 108522.https://doi.org/10.1016/j.compag.2023.108 522.
2. **Mansoor, S., Wani, O. A., Kumar, S. S., Popescu, S., Sharma, V., Sharma, A., ... Chung, Y. S. (2024)**. Artificial intelligence and IoT driven technologies for environmental pollution monitoring and management. Frontiers in Environmental Science, 12, 1336088.https://doi.org/10.3389/fenvs.2024.13360 88.

3. **Grzesik, P., Mrozek, D. (2024).** Combining Machine Learning and Edge Computing: Opportunities,Challenges,Platforms,Frameworks, andUseCases.Electronics,13(3),640.
4. **Ortega-Zamorano, F., Subirats, J. L., Jerez, J. M., Molina, I., Franco, L. (2013).** Implementation of theC-Mantec neural network constructive algorithm in an arduino UNO microcontroller. In Advances in Computational Intelligence: 12th International WorkConference onArtificialNeuralNetworks, IWANN2013, Puerto de la Cruz, Tenerife, Spain, June 12-14, 2013, Proceedings,Part I12(pp.80-87).SpringerBerlin Heidelberg.
5. **Izotov, Y. A., Velichko, A. A., Ivshin, A. A., Novitskiy, R. E. (2021).** Recognition of handwrittenMNIST digits on low-memory 2 Kb RAMArduino board using LogNNet reservoir neural network. In IOP Conference Series: MaterialsScienceandEngineering (Vol.1155,No. 1,p.012056). IOPPublishing.
6. **Kasera, R. K., Gour, S., Acharjee, T. (2024).** A comprehensive survey on IoT and AI based applications indifferentpre-harvest,during-harvest and post-harvest activities of smart agriculture. Computers and Electronics in Agriculture, 216, 108522.
7. **Rivai, M. ,Attamimi, M., Firdaus, M. H. (2019, November)**. Fish quality recognition using electrochemical gas sensor array and neural network. 2019 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM) pp. 1-5. IEEE.
8. **Jaber, A. A., Bicker, R. (2015).** Real-time wavelet analysis of vibration signals based on Arduino-UNO and LabVIEW. International Journal of Materials Science and Engineering, 3(1), 66-70.
9. **Parihar, V. R., Tonge, A.Y. ,Ganorkar, P.D. (2017)**. Heartbeat and temperature monitoring system for remote patients using Arduino. International Journal of Advanced Engineering Research and Science, 4(5), 55-58.
10. **Zafar, S., Miraj, G. ,Baloch ,R. ,Murtaza, D., Arshad, K. (2018).** An IoT based real-time environmental monitoring system using Arduino and cloud service. Engineering, Technology & Applied Science Research, 8(4), 3238-3242.
11. **Prasanna, R., Kakarla, P. … Mohan, N. (2022).** Implementation of Tiny Machine Learning Models On Arduino 33BLE For Gesture And Speech Recognition. arXiv preprint arXiv:2207.12866.
12. **Brennan, D., Galvin, P. (2024).** Evaluationofa MachineLearningAlgorithmtoClassifyUltrasonic TransducerMisalignment andDeployment Using TinyML.Sensors,24(2),560.
13. **Simanjuntak, J. E. S., Khodra, M. L., Manullang, M. C. T. (2020, July).** Design Methods of detecting atrial fibrillation using the recurrent neural network algorithm the Arduino AD8232 ECGmodule. Iop conference series:Earth And environmental science (Vol. 537, No. 1, p. 012022). IOPPublishing.
14. .**Ficco, M. ,Guerriero ,A., Milite, E., Palmieri,F., Pietrantuono, R., Russo, S. (2024).** Federated learning for IoTdevices: EnhancingTinyMLwith on-board training. Information Fusion, 104, 102189.
15. **Parker, G., Khan, M. (2016, July).**Distributed neural network: Dynamic learning via backpropagation with hardware neurons using arduino chips. In 2016 International Joint Conference on Neural Networks (IJCNN) (pp. 206-212). IEEE.
16. **Almufti, S.M.,Marqas, R. B., Nayef, Z.A., Mohamed, T. S. (2021).** Real TimeFacemask Detection with Arduino to Prevent COVID-19 Spreading. Qubahan Academic Journal, 1(2), 39-46.
17. **Bruce, P., Bruce, A., Gedeck, P. (2022).** Estadística práctica para la ciencia de datos con R y Python. Marcombo.
18. **Sánchez, C. C., Sepúlveda, F. H. (2015).** Estadística descriptiva: exploración de datos con R.
19. **Tauzin, G., Lupo,U., Tunstall, L.,Pérez, J.B., Caorsi,M., Medina-Mardones,A.M., ... Hess, K. (2021).** Giotto-Tda: A topological data analysis toolkit for machine learning data exploration. The Journal of Machine Learning Research, 22(1), 1834-1839.

Article received on 31/01/2025; accepted on 16/06/2025.
**Corresponding author is Hugo Estrada-Esquivel.*