Cracking the Code: Achieving High-Performance Embedded Gesture Recognition with Transparent Pipelines

Nathan Li
Department of Mechatronics Engineering
University of Waterloo
Waterloo, Canada
n99li@uwaterloo.ca

Simon Gorbet
Department of Mechatronics Engineering
University of Waterloo
Waterloo, Canada
sgorbet@uwaterloo.ca

***Abstract*—Existing automated embedded ML tools lack transparency, hindering debugging and maintenance. This study compares opaque NanoEdgeAI classifiers against transparent custom Scikit-learn/emlearn pipelines to evaluate performance trade-offs on microcontrollers. We collected 1,200 gesture samples separated into six gesture classes using an MPU9250 IMU and STM32F411RE MCU to train Support Vector Machines (SVM), Random Forest (RF), and (Multilayer Perceptrons) MLP models. The custom MLP achieved the highest on-device accuracy (91.78%), surpassing the best NanoEdgeAI model (91.00%), though it required more flash memory. Conversely, the custom RF model offered the lowest latency at 15.27 microseconds, executing 27 times faster than its automated counterpart, while automated models failed completely to classify circular gestures during live testing. Transparent pipelines can rival commercial AutoML tools in performance while offering superior interpretability and optimization flexibility.**

***Index Terms*—Embedded machine learning, Gesture recognition, IMU time-series classification, NanoEdgeAI, Microcontroller inference, Edge AI**

I. Introduction

This project uses machine-learning models to classify IMU gesture data, providing a testbed for comparing NanoEdgeAI's procedurally-generated classifiers with transparent Scikit and emlearn implementations deployed on resource constrained microcontrollers. Running gesture-recognition models directly on microcontrollers enables low-latency, low-power, and privacy-preserving motion interfaces for wearables, robotics, consumer electronics, and industrial tools. On-device inference avoids communication delays, bandwidth usage, connectivity failures, and privacy concerns associated with sending raw sensor data to external processors or cloud services, issues that become critical in mobile or safety critical systems. However, tight memory constraints and limited compute power makes deploying ML models to microcontrollers uniquely challenging [1].

Industry leading microcontroller manufacturers have started to provide end-to-end tools for training, developing, validating, and deploying ML models to their products, lowering the barrier of entry for creating on-device models. One such example is STMicroelectronics' NanoEdgeAI tool, which generates and benchmarks thousands of candidate models, outputting performance metrics such as accuracy, latency, memory usage, and confusion matrices [2]. However, these auto-generated models are not fully transparent, and their internal details such as features and hyperparameters choices are not fully disclosed. NanoEdgeAI does not expose its model transparently, it is unclear whether its top performing models can be reproduced or approximated using open, transparent machine-learning tools. Developers who deploy models on microcontrollers might require full visibility into the model for debugging, certification or long-term maintenance requirements [3]. If comparable performance can be achieved using standard frameworks such as Scikit and emlearn, then developers gain the flexibility to tune, modify and extend the models produced by automated EdgeAI development tools. Thus, attempting to replicate NanoEdgeAI's top SVM, Random Forest, and MLP classifiers using a fully transparent workflow allows us to evaluate the reproducibility, interpretability and deployment trade-offs between automated embedded-AI tools and custom-engineered pipelines.

Gesture recognition provides a well-studied, accessible testbed for evaluating embedded machine-learning workflows. Accelerometer-based gestures are simple to record, require no specialized hardware beyond an IMU, and have been repeatedly shown to be learnable using lightweight classical ML models such as kNNs, SVMs, Random Forests, and small neural networks [4], [5]. Because gestures like circles and swipes produce clean, structured motion patterns, they allow controlled experimentation without the complexity, noise, or domain-specific feature engineering required by more advanced signal-processing tasks. This makes gesture classification an ideal benchmark for exploring the differences between automated embedded-AI tools and custom model-development pipelines. The gestures to be classified in this experiment are shown below in Fig. 1.

The objective of this project is to explore the complete workflow for developing an embedded-AI solution within an industrial application. A crucial element of this endeavor is data collection; consequently, a novel dataset was generated

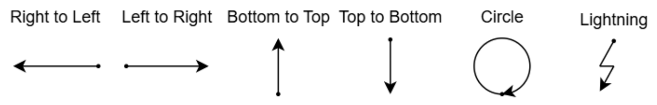


Fig. 1: Visualization of gestures to be classified.

utilizing custom data. This was accomplished utilizing the MPU9250 IMU, which was affixed to a pointing stick situated 10 centimeters from the grip. The sensor incorporates a 3-axis accelerometer, gyroscope, and magnetometer. Gesture classification was restricted such that the pointing stick consistently begins and concludes in the same orientation relative to the user, although the direction relative to Earth's magnetic field may vary. Given these constraints, only the accelerometer and gyroscope data were collected, yielding 6 degrees of freedom per sample. The dataset is comprised of six distinct gestures produced by four individuals in different orientations, resulting in a total of 200 samples for each gesture. This data was then partitioned 80:20 to a training and test set, respectively.

This project evaluates three lightweight algorithms suitable for microcontroller-class gesture recognition and covered in lecture: SVMs, RFs, and MLPs. These models were selected because they align with the types of classifiers NanoEdgeAI identifies as top performers and are feasible to deploy on embedded hardware. For the Random Forest model, the explored parameters will be number of trees and maximum tree depth as these control ensemble stability and model complexity. For the MLP, we will vary the number of neurons, and number of hidden layers since these determine model capacity and memory footprint. Secondary parameters such as batch size or number of epochs will be handled using standard early-stopping criteria rather than explicit tuning. Performance will be measured using the same metrics produced by NanoEdgeAI—balanced accuracy, confusion matrices, latency, RAM usage, and flash size—allowing for a direct comparison between auto-generated models and their transparent counterparts.

II. Materials and Methods

A. Hardware & Software Configuration

For practical experimentation, we procured a hardware system to collect and evaluate the models. We used the NUCLEO-F411RE developer board as the MCU because it is supported by NanoEdgeAI Studio, has 512kB of flash and 128kB of SRAM memory which eases memory constraints of our models. Another benefit of the STM32F411RE is its Cortex-M4F architecture, which is widely supported by open-source embedded ML frameworks and has a dedicated Floating Point Unit. The sensor used for data collection was the MPU9250 IMU. The sensor provides three-axis accelerometer and gyroscope measurements, resulting in six degrees of freedom per sample. accelerometer and gyroscope full-scale ranges were set to ±16 g and ±2000 dps respectively, both with 16-bit resolution, to avoid signal saturation during dynamic gestures. The hardware components were then taped to a ruler, with the developer board on the base and IMU at the tip to capture the "wand" movements as seen in Fig. 2. The sensor communicated with the MCU using the I2C protocol, and all signals were sampled at a rate of 200 Hz. Development utilized the C programming language for the MCU firmware, with compilation managed through the PlatformIO IDE. Real-time data readouts were recorded via UART with a laptop connected to the developer board. The post-processing and visualization scripts were created using Python, with 95% confidence intervals being implemented as error bars for all graphs.

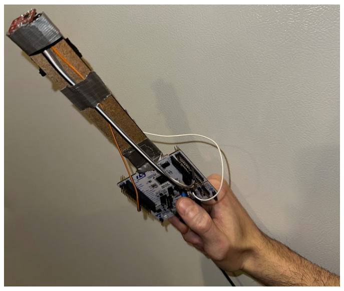


Fig. 2: Photograph of wand device.

B. Feature Vector Definition

The input to the embedded classifier is structured as a feature matrix where each row (m) represents a single gesture instance. The temporal dimension is flattened horizontally: for every time step n, the six sensor channels (accelerometer and gyroscope) are concatenated sequentially. This results in a feature vector of length $6 \times n$ for each run.

$$\begin{bmatrix} \underbrace{Ax_{1,1}\ \ Ay_{1,1}\ \ Az_{1,1}\ \ Gx_{1,1}\ \ Gy_{1,1}\ \ Gz_{1,1}}_{t=1} & \cdots & \underbrace{Ax_{1,n}\ \ \cdots\ \ Gz_{1,n}}_{t=n} \\ \vdots\quad\ \ \vdots\quad\ \ \vdots\quad\ \ \vdots\quad\ \ \vdots\quad\ \ \vdots & \ddots & \vdots\quad\ \ \ddots\quad\ \ \vdots \\ Ax_{m,1}\ \ Ay_{m,1}\ \ Az_{m,1}\ \ Gx_{m,1}\ \ Gy_{m,1}\ \ Gz_{m,1} & \cdots & Ax_{m,n}\ \ \cdots\ \ Gz_{m,n} \end{bmatrix} \tag{1}$$

C. Data Collection

A custom gesture dataset was collected for this study. The dataset consists of six unique gestures performed by five individuals. Each gesture was performed 45 times by participants, resulting in 270 samples per gesture and a total of 1620 labeled gesture instances. Participants record their gestures by holding the user button of the developer board. The samples are composed of time-series synchronized three-axis accelerometer and gyroscope data. The number of data points per sample varied depending on gesture type and the user creating data.

The dataset consists of six unique gestures performed by five individuals. Each gesture was performed 45 times by participants, resulting in 270 samples per gesture and a total of 1620 labeled gesture instances. Participants record their gestures by holding a button at the base of the recording device. The samples are composed of time-series synchronized three-axis accelerometer and gyroscope data. The number of

TABLE I: Gesture Data Statistics

Gesture Type	Mean Data Points per Gesture	Standard Deviation σ
Circle	60.34	11.78
Lightning	55.64	4.29
Swipe Up	24.42	5.89
Swipe Down	38.64	13.18
Swipe Left	38.68	13.63
Swipe Right	40.40	11.99

data points per sample varied depending on gesture type and the user creating data.

After collection, the datasets were preprocessed to prevent erroneous data from being used to train the models. Firstly, we noticed that some samples had very few or no data points due to the button being released during a run. To avoid this data being used for training, any samples that had data points beyond $\pm 1.5\sigma$ of the mean data points per gesture were dropped. To maintain a consistent number of data points, each sample was temporally resampled to 100 data points each using interpolation. Following data collection, the datasets underwent pre-processing to mitigate the inclusion of erroneous data in the model training. Furthermore, to ensure a uniform number of data points across all samples, each axis was temporally resampled to exactly 100 data points using interpolation. The samples were then validated to have distinct features from each other by plotting each accelerometer and gyroscope axis per gesture with a 5% confidence interval as seen in Fig. 3 and 4. After pruning failed samples and ensuring all gestures have the same number of valid samples, we are left with 200 samples per gesture for a total of 1200 labeled gesture instances.

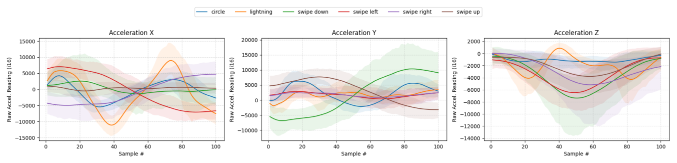


Fig. 3: Acceleration readings per axis by gesture.

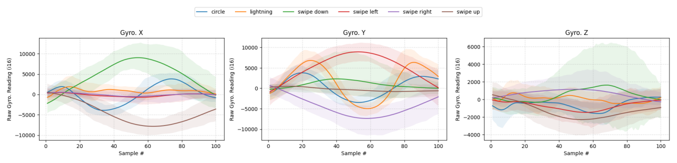


Fig. 4: Gyroscope readings per axis by gesture.

D. NanoEdgeAI Model Generation

NanoEdgeAI (NEAI) was used to automatically generate and benchmark candidate gesture-classification models. NEAI uses an 80:20 train:test split for evaluating the thousands of models it tests. Each candidate model is ranked using NEAI's quality index, a composite model ranking system based on balanced accuracy , F1-score, Matthews correlation coefficient (MCC), custom measurement that estimates the degree of certainty of a classification inference, and lastly RAM & flash memory usage. The top ranking SVM, MLP, and RF models according to quality index were selected for further evaluation. These models were exported and deployed directly to the target embedded platform using NEAI's generated code.

E. Custom Model Development

In parallel, transparent custom-built Random Forest and MLP classifiers were developed using an open machine-learning pipeline. Feature extraction and model development were performed in Python using standard Scikit-learn tools. Trained models were then converted to embedded-compatible C implementation using the emlearn framework, which provides methods to convert Python based models from Scikit into C based microcontroller inference functions. Emlearn also provides methods for estimates of memory usage, and computational cost of tree-based models.

For both model types, multiple model configurations were trained by varying key hyper-parameters. For RF models, the number of estimators and max depth were varied and benchmarked. For MLP models, the number of hidden layers, and neurons per layer were varied and benchmarked. Each model variation was benchmarked on their average performance across 10 trials on an 80:20 train:test split. The top-performing custom models were selected using a weighted composite score consisting of balanced accuracy (weight 0.9), emlearn estimated memory usage (weight 0.05), and emlearn estimated compute cost (0.05). This selection criterion prioritizes classification robustness while accounting for embedded resource constraints.

F. Final Evaluation

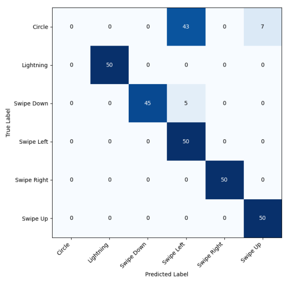
The selected models were deployed to the embedded gesture-capture device ("wand") for the final evaluation. Inference was performed on the embedded device for each gesture performed by a new sixth participant who did not appear in the data collection set. Each gesture was performed 50 times by the new user where the predicted class and inference time were recorded. Final compiled flash and ram usage was also measured for each selected model.

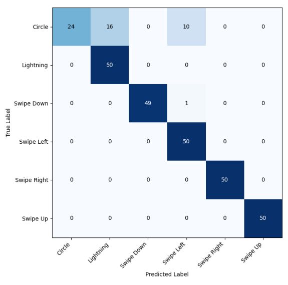
III. Results and Discussion

A. NanoEdgeAI Models

Training yielded models employing RF, SVM, and MLP methodologies, all demonstrating estimated accuracies surpassing 99.9%. Nano Edge Studio subsequently identified the SVM model as the highest-performing due to its minimal utilization of both RAM and flash memory. We then tested these models on the STM32F411 with 50 samples per gesture by comparing the estimated and actual accuracy and inference time.

The on-device testing demonstrated that the MLP model achieved the highest classification accuracy, with the SVM and RF models exhibiting lower performance, as illustrated in Fig. 5. It was observed that accuracy was highly contingent upon the specific gesture executed. While the model generally classified swipe gestures correctly, occasional misclassification between 'swipe down' and 'swipe left' occurred. Furthermore, the model was entirely unable to classify the 'circle' gesture, which was consistently misclassified as either 'swipe left' or 'swipe up'. The circular motion commences with a diagonal component, comprising elements of both 'swipe up' and 'swipe left', which is the probable cause of this misclassification. The MLP model similarly classified most gestures accurately, but registered only a 48% accuracy for the 'circle' gesture, most frequently confusing it with 'lightning' or 'swipe left'. The RF model yielded the lowest accuracy overall, with low classification accuracy for the horizontal swipes, 'lightning', and 'circle' gestures.


(a) SVM


(b) MLP

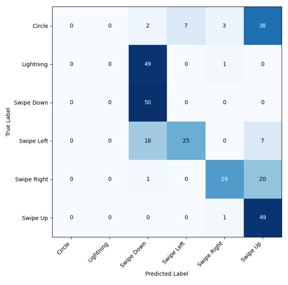

(c) RF

Fig. 5: NanoEdgeAI Model Embedded Inference Performance.

The model performance during deployment on the embedded device was found to be lower than the estimated metrics provided by the NanoEdgeAI Studio as seen in Tab. II. Several factors may contribute to this observed discrepancy. The deployed models incorporated fixed-size enumeration types, which allocate only the necessary number of bytes for the declared range of possible values. This practice significantly optimizes memory utilization but consequently leads to a reduction in performance. The difference in execution time is potentially attributable to variations in the compilers used. As the NanoEdgeAI Studio abstracts the compiler employed for accuracy estimation, it is plausible that it utilizes a more highly optimized version than the compiler implemented within the NUCLEO-F411RE via PlatformIO.

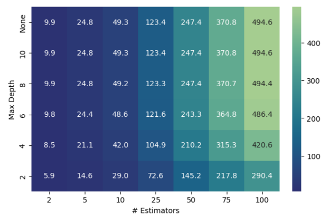
B. Custom Models

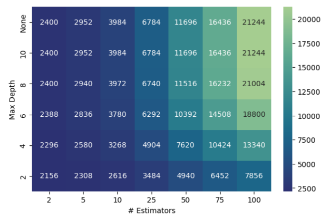
1) Random Forest: Scikit's random forest classifier was tested with the number of estimators parameter set within

TABLE II: NanoEdgeAI Model Performance Comparison.

Model	Estimated Inference Time (μs)	On-Device Inference Time (μs)	Estimated Accuracy (%)	On-Device Accuracy (%)
SVM	300	453.62	100	81.67
MLP	500	824.16	100	91.00
RF	300	411.79	99.95	51.00

the range of 2-100 and max depth parameter set within the range of 2-10 as well as with no depth limit. All other model parameters are set to Scikit's default. The resulting model benchmarks are seen below in Fig. 6. All benchmarked RF models have identical RAM usage estimates of 1100 bytes.


(a) Est. Compute (Lower Better)


(b) Est. Flash In Bytes

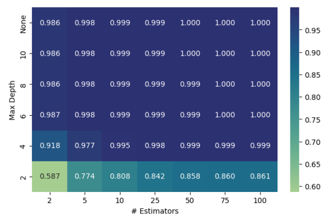

(c) Mean Accuracy on Test

Fig. 6: Custom-built RF Models Benchmarks.

Using the weighted model selection criterion, the resulting selected model has 25 estimators and a max depth of 6.

Deploying the model to the wand and having the gestures performed by the test subject shows an average inference time of 15.27 micro seconds, with a mean accuracy of 76.32%, and produces the confusion matrix seen below in Fig. 7. The model's estimated and deployed performance are compared in table III below. The final deployed model along with necessary implementation code has a memory footprint 30292 bytes in flash and 26996 bytes in RAM.

2) Multilayer Perceptron: Scikit's multilayer perceptron classifier was tested with the following hidden layer configurations [(8,), (16,), (64,), (16, 16), (32, 32)] and a regularization value of 1e-5. All other model parameters are set to Scikit's default. The resulting model benchmarks are seen below in 8. All benchmarked MLP models had identical RAM usage estimates of 4800 bytes.

Using the weighted model selection criterion, the resulting selected model has a single 32 neuron hidden layer.

Deploying the model to the wand and having the gestures performed by the test subject shows an average inference time of 3178.29 micro seconds, with a mean accuracy of 91.78%,

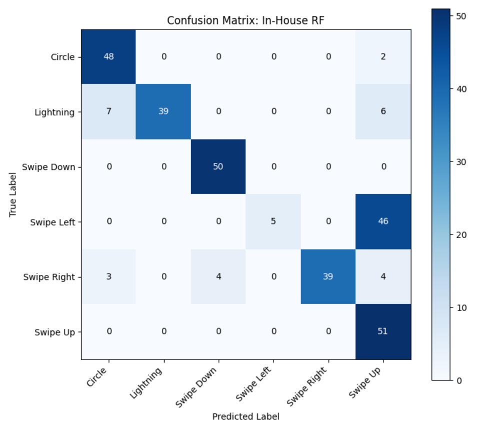


Fig. 7: Custom-built RF Model Embedded Inference Performance.

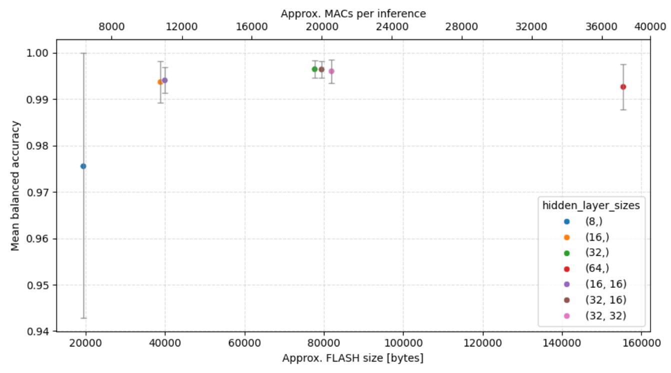


Fig. 8: Custom-built MLP Models Benchmarks.

and produces the confusion matrix seen below in Fig. 9. The model's estimated and deployed performance are compared in table III below. The final deployed model along with necessary implementation code has a memory footprint 115836 bytes in flash and 33020 bytes in RAM.

TABLE III: Custom-Built Model Performance Comparison.

Model	Estimated Accuracy (%)	On-Device Accuracy (%)
MLP	99.67	91.78
RF	99.95	76.32

C. Model Comparisons

Overall, the custom-built models had a better on-device accuracy compared to the NanoEdgeAI models as seen in Fig 10. Interestingly, that relationship is opposite with their respective estimated accuracies. The MLP models had the highest accuracy, followed by SVM and RF. Of the five models tested, the custom-built MLP model achieved the highest accuracy. During testing, the only models to exhibit "blind spots" (0% correct prediction) for a specific gesture were the

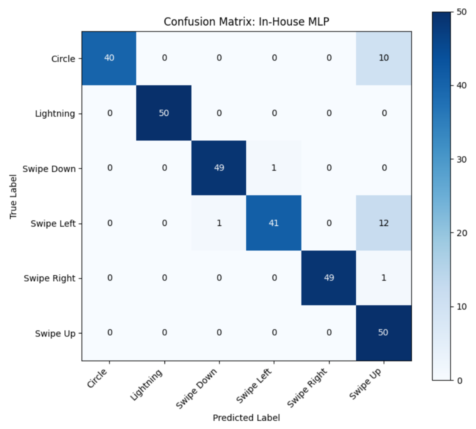


Fig. 9: Custom-built MLP Model Embedded Inference Performance.

NanoEdgeAI RF and SVM models for the 'circle' gesture. Furthermore, the NanoEdgeAI RF model showed an additional blind spot for the 'lightning' gesture. A common observation was that all models performed significantly worse in deployed environments compared to their predicted performance, which is likely due to the limited number of data samples used for training and testing, as well as the users having inconsistent motions per each gesture.

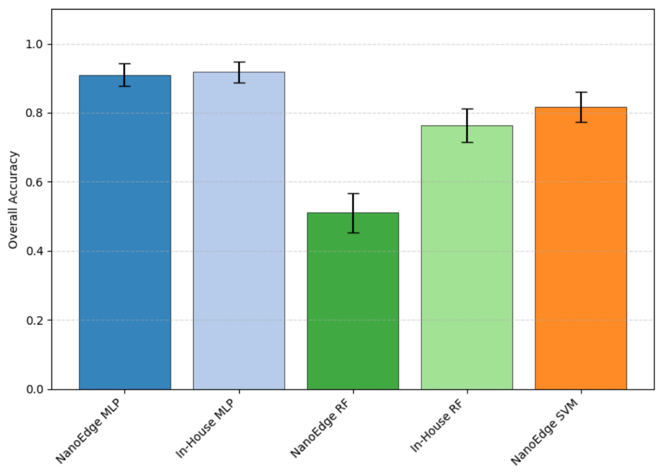


Fig. 10: Overall Accuracy per Model.

There was high variance between inference times of the Nano Edge and custom-built models as seen in Fig 11. The custom-built RF model was significantly faster than the other models, with inference being completed almost 27 times faster than its NanoEdgeAI equivalent. The custom-built MLP model had the longest absolute inference time, and took over 3.85 times longer than the NanoEdgeAI MLP model. The NanoEdgeAI models had slower on-device inference times compared to the time estimated by NanoEdgeAI Studio.

The size of the overall programs are separated into RAM and flash memory as seen in Tab. IV. RAM usage was consistent across all five models, with a standard deviation

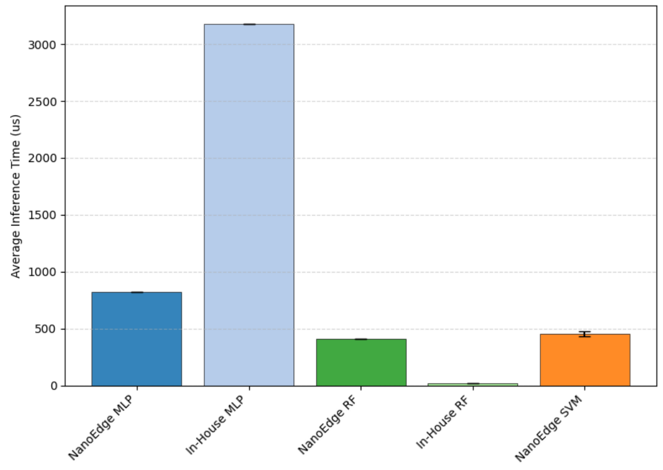


Fig. 11: Overall Inference Time per Model.

of less than 2.5kB. This is because the RAM memory requirements are dictated by the hardware and data processing overheads that are shared between models. This primarily consists of the buffer that stores the 600 normalized data points that are used for inference and the system overhead for the peripheral drivers and inference functions that are common for all models. However, flash usage varied more significantly; this is because flash stores the models structure, weights, and biases themselves. Excluding the custom-built MLP model, the standard deviation was approximately 4.2kB, and the custom-built MLP model was 78.4kB larger than the next largest model. This discrepancy in the size of the custom-built model is because it stores its weights and biases as 4-byte floats, which inflates the flash storage size.

TABLE IV: Compiled Resource Usage Comparison

Source	Model	Compiled RAM Usage (KB)	Compiled Flash Usage (KB)
NanoEdgeAI	SVM	28.724	29.004
	MLP	31.232	35.880
	RF	28.704	37.484
Custom-built	MLP	33.020	115.836
	RF	26.996	30.292

D. *Deployment-driven model selection*

No single model dominates across accuracy, latency, and flash. For highest accuracy on-device, the custom MLP reached 91.78%, but required 3178 µs per inference and 115.836 KB flash, making it the most resource-intensive option. For lowest latency, the custom RF achieved 15.27 µs inference with 30.292 KB flash, but with reduced accuracy (76.32%) relative to the best MLPs. In contrast, NanoEdgeAI's MLP provides a strong middle ground (91.00%, 824 µs, 35.880 KB flash), while NanoEdgeAI's SVM yields the smallest flash footprint (29.004 KB) at moderate accuracy (81.67%). These trade-offs suggest a simple deployment heuristic: choose custom RF for tight real-time constraints, NanoEdgeAI MLP for balanced performance/size, and custom MLP only when flash budget allows and top accuracy is the priority.

IV. Conclusion

In conclusion, this project demonstrates that high-performing embedded gesture-recognition models can be constructed using fully transparent machine-learning pipelines that rival or exceed the real-world performance of automated Edge-AI tools. While NanoEdgeAI offers substantial productivity benefits, its opaque model generation and optimistic performance estimates limit its suitability for deployment-critical applications. Future work could explore hybrid workflows, where NanoEdgeAI is used for initial architecture discovery followed by transparent reimplementation and optimization. Additionally, expanding the dataset to include greater inter-user variability and investigating quantized or fixed-point custom models could further improve robustness and reduce memory overhead on microcontroller platforms.

References

[1] C. R. Banbury, V. J. Reddi, M. Lam, W. Fu, A. Fazel, J. Holleman, X. Huang, R. Hurtado, D. Kanter, A. Lokhmotov, D. Patterson, D. Pau, J. sun Seo, J. Sieracki, U. Thakker, M. Verhelst, and P. Yadav, "Benchmarking tinyml systems: Challenges and direction," 2021.

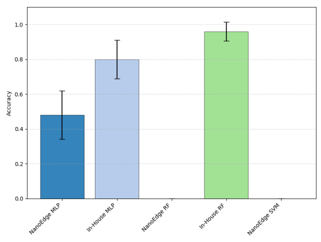
[2] "Nanoedge ai studio," Jul 2025.

[3] R. Sanchez-Iborra and A. F. Skarmeta, "Tinyml-enabled frugal smart objects: Challenges and opportunities," *IEEE Circuits and Systems Magazine*, vol. 20, no. 3, pp. 4–18, 2020.

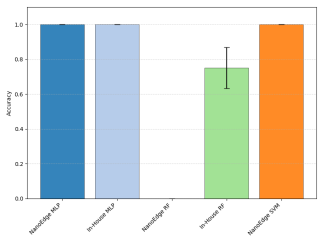
[4] T. Marasović and V. Papić, "Accelerometer based gesture recognition system using distance metric learning for nearest neighbour classification," in *2012 IEEE International Workshop on Machine Learning for Signal Processing*, pp. 1–6, 2012.

[5] P. Warden and D. Situnayake, *TinyML: Machine learning with tensorflow lite on Arduino and ultra-low Power Microcontrollers.* O'Reilly Media Inc, 2020.

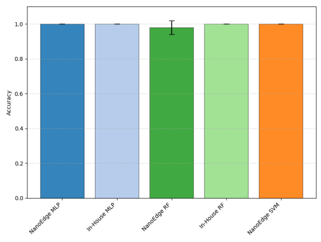
Appendix A
Supplementary Gesture Metrics



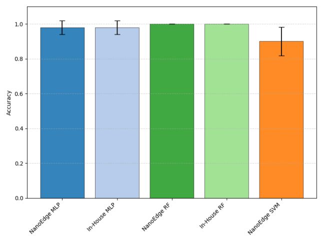
(a) Circle Gesture Accuracy



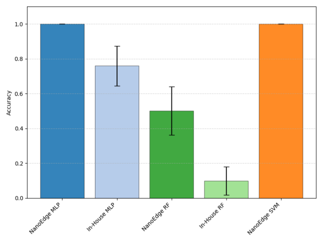
(b) Lightning Accuracy



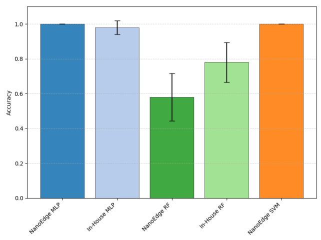
(c) Swipe Up Gesture Accuracy



(d) Swipe Down Accuracy



(e) Swipe Left Accuracy



(f) Swipe Right Accuracy

Fig. A1: Breakdown of accuracy metrics by gesture type, comparing NanoEdgeAI models against custom-built models.