

# Real-Time Fall Detection on an Edge Device Using Edge Impulse and Arduino Nano 33 BLE

Milan Nieuweboer and Yunus Emre Demir

## 1. Introduction

With the increasing capabilities of AI models on edge devices, new opportunities arise for developing intelligent systems that can operate efficiently without relying on cloud computing. Running machine learning models on small, energy-efficient hardware enables real-time processing, reduces latency, and ensures privacy by keeping data local. However, deploying AI on embedded systems comes with challenges such as limited computational resources, memory constraints, and power efficiency [\[1\]](#).

To explore the potential of AI on edge devices, this project focuses on developing a fall detection system as a practical use case. Fall incidents pose a significant risk, particularly for older adults and other vulnerable groups, as they can lead to severe injuries, hospitalization, and loss of independence [\[2\]\[3\]](#). Since individuals who fall are not always able to call for help themselves, an automated detection system could provide a crucial safety measure.

By leveraging the Arduino Nano 33 BLE sensor and TinyML technologies, this project investigates how machine learning models can be optimized for real-time fall detection in a resource-constrained environment. Using Edge Impulse as the development platform, various AI techniques will be explored to build a robust and practical system capable of detecting falls with high accuracy and reliability.

In this project, we focus on the question: "How can a fall detection system using the Arduino Nano 33 BLE and machine learning be implemented to enable rapid emergency response?" To support this, we also explore the following question: "How can a classification model be implemented in real time on an edge device?" These questions help guide our approach in selecting the most suitable technology and methods for developing an accurate and reliable fall detection system.

Our ultimate goal is to deploy a model that can consistently and accurately detect falls in real-world scenarios, while avoiding having more general movements trigger the device. This ensures that the system remains both sensitive to actual incidents and robust against false alarms, making it a practical solution for real-life deployment.

## 2. Literature Review

### 2.1 Fall-Related Accidents and Their Impact

Falls represent a significant public health concern, particularly among older adults. Research indicates that fall-related deaths have been rising over the past decades. A study by Van der Naald et al. [2] analyzed trends in fall-related mortality in the Netherlands from 1990 to 2045, revealing a continuous increase in fatal falls, particularly among the elderly. Their projections highlight the urgent need for effective fall prevention and detection systems to mitigate the risks associated with aging populations.

Further statistics from Statistics Netherlands [3] report that, as of 2019, an average of 13 people per day in the Netherlands died as a result of a fall. This alarming trend underscores the necessity of implementing reliable fall detection technologies to provide timely assistance and potentially reduce fatalities.

## 2.2 Sensor-Based Fall Detection

Existing research on fall detection focuses on leveraging sensor data, particularly from accelerometers and gyroscopes, to improve accuracy and reliability.

Li et al. [4] explore the integration of gyroscope and accelerometer data to derive posture information for fall detection. Their study emphasizes the importance of combining multiple sensor modalities to reduce false positives and improve real-time responsiveness. They highlight that analyzing body orientation and sudden changes in motion can significantly enhance the accuracy of fall classification.

Similarly, Rakhman et al. [5] investigate fall detection using smartphone-based accelerometers and gyroscopes. Their work demonstrates that sensor fusion techniques can effectively distinguish between falls and normal daily activities. The study also discusses the importance of selecting appropriate threshold values and classification methods to optimize detection performance.

These studies indicate that a multimodal approach—utilizing both accelerometer and gyroscope data—plays a crucial role in accurate fall detection. However, challenges remain in minimizing false detections and ensuring robustness across different users and environments. Since these studies have not used a machine learning-based solution, it shows that there still might be room for improvement for us to discover.

## 2.3 Microcontroller for Fall Detection

Embedded microcontrollers play a crucial role in real-time fall detection, enabling efficient on-device processing. The Arduino Nano 33 BLE is a compact and powerful microcontroller designed for AI and IoT applications. It features an nRF52840 SoC with Bluetooth Low Energy (BLE) and multiple built-in sensors, making it suitable for edge-based machine learning applications.

Key specifications include:

- Processor: Arm Cortex-M4F (64 MHz)
- Connectivity: Bluetooth 5.0 Low Energy
- IMU (Inertial Measurement Unit): 3-axis accelerometer and gyroscope (LSM9DS1)

- Additional Sensors: Temperature, humidity, barometric pressure, color, and microphone
- Memory: 1MB Flash, 256KB RAM
- Power Efficiency: Designed for battery-powered applications

The Arduino Nano 33 BLE is particularly well-suited for fall detection applications due to its compact size, low power consumption, and integrated Bluetooth capabilities. Its onboard LED provides visual feedback for real-time fall alerts, while its affordability makes it accessible for battery-powered implementations. Additionally, the ability to run models directly on the microcontroller enables real-time processing without requiring a cloud connection, reducing latency and improving response times.

## 2.4 TensorFlow Lite for Microcontrollers

TensorFlow Lite (TFLite) is an optimized machine learning framework designed for resource-constrained devices such as microcontrollers. It enables neural networks to run efficiently using techniques like quantization, which converts 32-bit floating-point values into 8-bit integers to reduce memory usage and computation time. Additionally, pruning and clustering remove redundant network connections and group weights, further accelerating inference [\[8\]](#).

For ultra-low-power hardware, TensorFlow Lite for Microcontrollers (TFLM) provides a minimal runtime with no operating system dependencies, allowing models to run efficiently on microcontrollers like the Arduino Nano 33 BLE. Moreover, TFLite supports integration with hardware accelerators such as CMSIS-NN, optimizing Arm Cortex-M processors for rapid matrix operations [\[9\]](#).

This will answer our sub question "How can a classification model be implemented in real time on an edge device?".

## 3. Methodology

### 3.1 Data Acquisition

To collect real-time sensor data for machine learning training, we connected the Arduino Nano 33 BLE to Edge Impulse using a 3-meter micro USB cable. Before the Arduino could be linked as a device, we first had to flash a new firmware version to ensure compatibility with Edge Impulse. Once updated, we used the command `edge-impulse-daemon` to successfully connect the Arduino.

Through the Edge Impulse platform, we were able to select labels for data collection. The tables below summarize the labels used and the number of samples gathered for both training and test sets.

Label	Train Sample	Test Sample
Walk	88	25
Fall	21	7
Idle	17	6
SittingDown	7	3
StandingUp	7	3
Total	140	44

On average, each data sample is around 3.5 seconds long. The duration of each sample was chosen to capture the full range of motion for each activity, providing the model with sufficient temporal context to accurately recognize and differentiate between movements. The data includes the following durations for each activity:

- **Walk:** 5 minutes 3 seconds
- **Fall:** 1 minute 25 seconds
- **Idle:** 43 seconds
- **Sitting Down:** 21 seconds
- **Standing Up:** 21 seconds

This selection was made to determine whether fall detection was feasible in an initial version of the product. Additionally, we included some daily activities where a person might lower their height, such as, sitting on a chair, to evaluate the robustness of the system and prevent misclassification. Adding this variation might force, for example, that sitting down would not be mistaken for a fall. First we also had included stair walking as a different class, but because in all the training results this ended up being classified as “Walk”, we decided to merge these labels.



Figure 1: Examples of how the data set was obtained

According to the official documentation [\[6\]\[7\]](#), the collected sensor data consists of gyroscope and accelerometer readings, each with X, Y, and Z components. The accelerometer measures linear acceleration, while the gyroscope captures rotational velocity. These signals provide crucial information for detecting falls by analyzing sudden changes in motion and orientation. A visualization of this can be seen in figures 2 and 3.

To collect the data, we connected the Arduino to a laptop with a USB cable and then fixed it to the person's belt in a 3D-printed case for steady positioning. We collected 7 minutes of data for the train data set and 3 minutes of data for the test data set. Data has been collected via 2 different individuals to create a more general model.

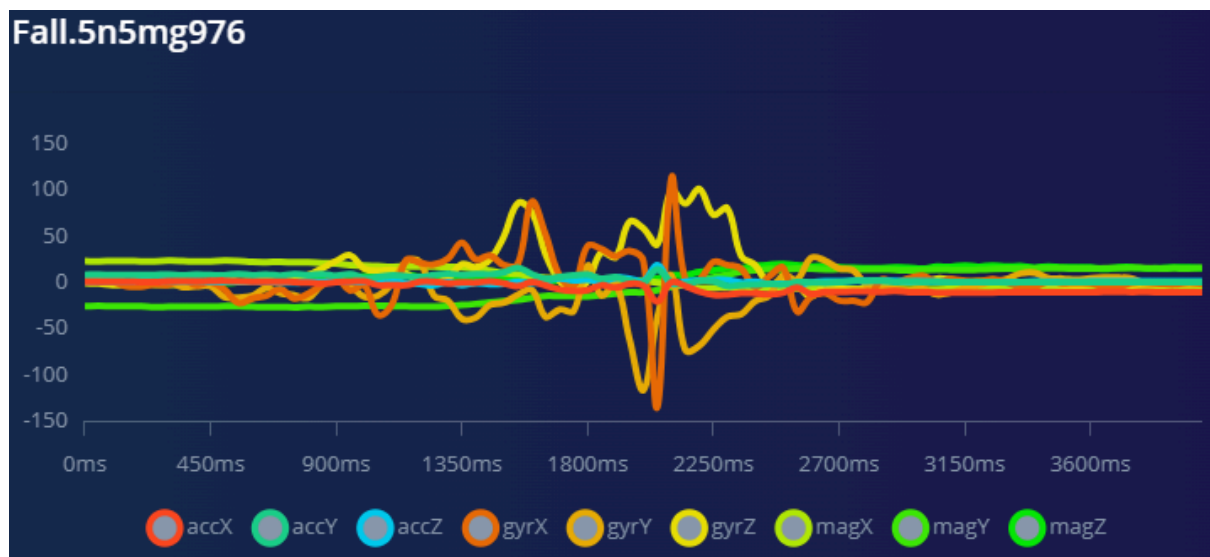


Figure 2: Example of a fall

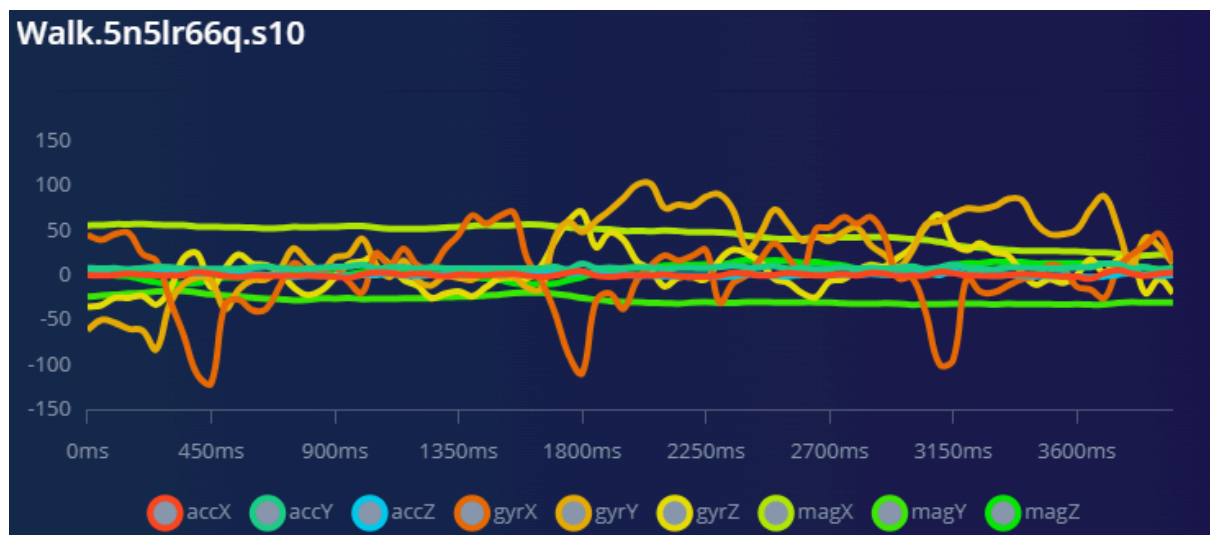


Figure 3: Example of walking

## 3.2 Data Preprocessing

During data collection, the duration of each recorded action varied. Continuous movements, such as walking, were recorded over multiple minutes, whereas discrete actions, such as falling, sitting down, and standing up, were captured per instance. To ensure consistency across all actions, we standardized the collected data using a fixed time window, where the continuous data was split into multiple 3 second sections.

We selected a 3 second time window based on the discrete actions in our dataset, which are measured over 3 seconds. This ensures that all movement patterns, including short-duration events like falls and longer activities like walking, fit within the same analysis window. Additionally, we implemented a sliding window with a 500ms overlap to improve detection accuracy. This allows the model to recognize actions that slightly exceed the predefined time window, preventing misclassification if an action takes slightly longer than expected.

Although data has been captured among 9 axes, the Edge Impulse Arduino library only gave the ability to use the accelerometer. Because of that we weren't able to use the gyroscope and magnetometer in this version, but it later showed that with this approach feasible results were still possible.

## 3.3 Model Training and Testing

After collecting and preprocessing the sensor data, we trained a machine learning model for fall detection using Edge Impulse. Given that the model needed to distinguish between different poses, we opted for a classification model designed to handle time-series data within Edge Impulse.

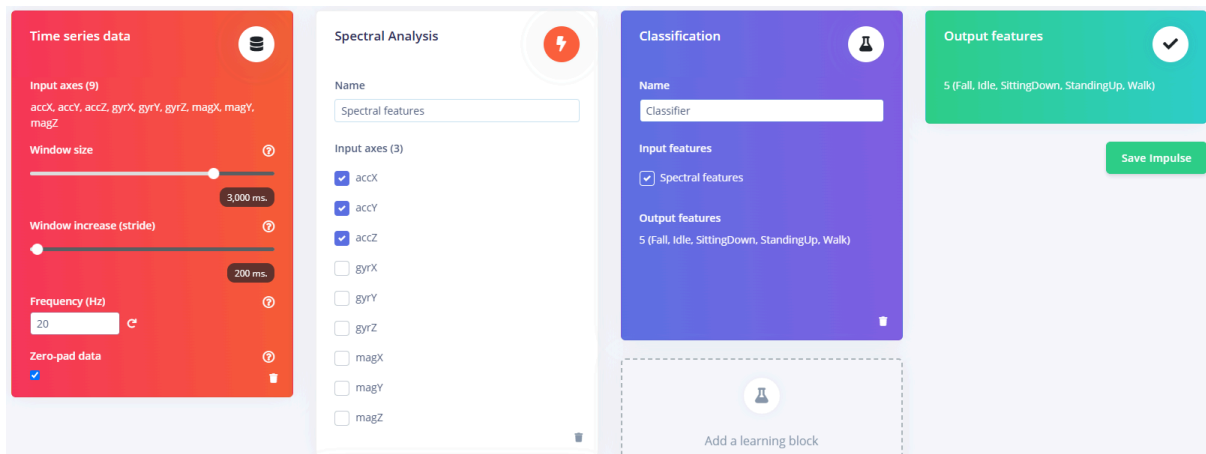


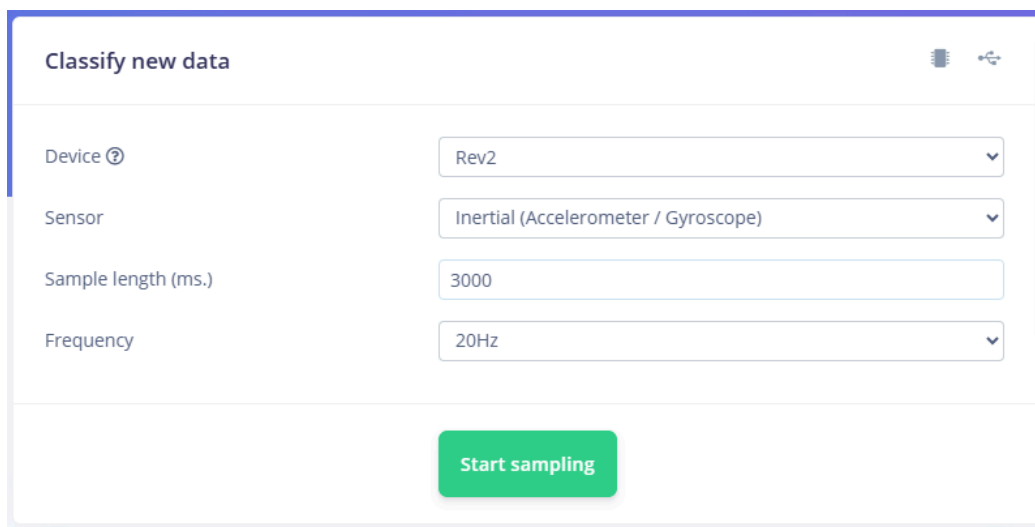
Figure 4: Setup of the “Impulse” in the Edge Impulse environment

The iterative training process consisted of the following steps:

- **Training and Validation:** The model was trained using labeled movement data, while hyperparameters such as the learning rate, number of epochs, and layer size were fine-tuned to optimize performance. After training the baseline model, we focused on fine-tuning the hyperparameters that Edge Impulse gave us control over. These were the number of epochs (where we chose 20, 50, 75, and 100), the

learning rate (where we chose 0.005, 0.001, and 0.0005) and the amount of neurons and layers in the architecture. These values were chosen experimentally as a kind of grid search, as they typically lead to significant variations in performance, allowing us to identify clear improvements. For the model we wanted to keep the architecture as small as possible because it has to perform in real time on an edge device, which is why we tried to keep the neurons in the dense layers as low as possible.

- **Evaluation Metrics:** To assess the model's accuracy, we measured the F1-score across all movement classes. This will be our main metric as well, since we think for this initial product version the balance in successfully detecting falls without having false positives is in balance. Note that in Edge Impulse, a default confidence threshold of 0.4 is applied to validation predictions, meaning that any output below this threshold is classified as "Uncertain." We only discovered in the later iterations that this setting could be disabled, which is why some confusion matrices still contain this column.
- **Live classification:** With Edge Impulse it was possible to record a pose while having the Arduino attached to the body, which made it possible to test the model performance in a real test scenario. Here we performed multiple poses to conduct if the model would predict the *Fall* class correctly and hopefully wouldn't predict *Fall* when the person was sitting down or walking. This extra step of testing is necessary to test how the model performs in a real life scenario.



The screenshot shows a web interface titled "Classify new data". It contains four configuration fields: "Device" with a dropdown menu showing "Rev2", "Sensor" with a dropdown menu showing "Inertial (Accelerometer / Gyroscope)", "Sample length (ms.)" with a text input field containing "3000", and "Frequency" with a dropdown menu showing "20Hz". At the bottom of the form is a green button labeled "Start sampling".

Figure 5: *Live classification with the Arduino*

### 3.3.1. Baseline Model

Our initial two hidden layer model with 20 and 10 neurons was the default setup provided by Edge Impulse, with a learning rate of 0.0005 and 20 epochs. We used this as a baseline to assess performance and determine the necessary adjustments for further tuning.



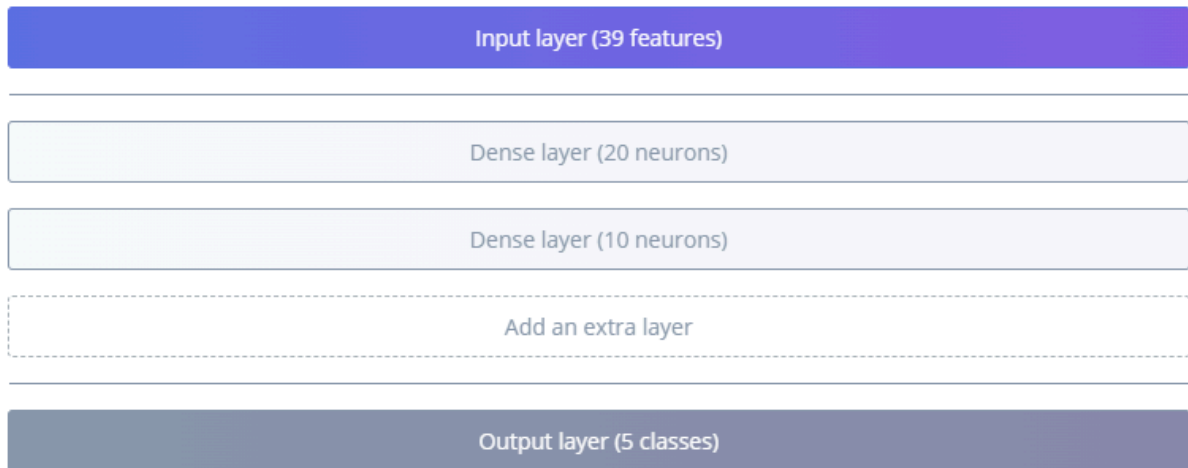


Figure 6: Baseline model architecture

	FALL	IDLE	SITTINGDOWN	STANDINGUP	WALK
FALL	0%	66.7%	0%	0%	33.3%
IDLE	0%	100%	0%	0%	0%
SITTINGDOWN	0%	0%	0%	66.7%	33.3%
STANDINGUP	100%	0%	0%	0%	0%
WALK	0%	100%	0%	0%	0%
F1 SCORE	0.00	0.57	0.00	0.00	0.00

Figure 7: Baseline model results: 2 hidden dense layers (20, 10), with learning rate 0.0005 and 20 epochs

For this small model, it shows that only the “Idle” pose has been detected correctly. Since this pose is very minimal in its data variety because of the standing still, we think the model wasn’t able yet to tell the differences in the more complex poses.

### 3.3.2. Model Optimizations

By adding multiple neurons to the layers, we were hoping to add more complexity to the model. For these layers, a learning rate (LR) of 0.0005 and 100 epochs performed best:

	FALL	IDLE	SITTINGDOWN	STANDINGUP	WALK
FALL	85.7%	14.3%	0%	0%	0%
IDLE	0%	100%	0%	0%	0%
SITTINGDOWN	100%	0%	0%	0%	0%
STANDINGUP	66.7%	0%	0%	33.3%	0%
WALK	77.8%	0%	0%	0%	22.2%
F1 SCORE	0.48	0.92	0.00	0.50	0.36

Figure 8: 2 Hidden dense layers (20, 30), learning rate 0.0005 and 100 epochs

To further optimize the model, we experimented with adding an additional dense layer, as we suspected that the initial architecture lacked the complexity needed to differentiate between



similar movement patterns effectively. During live testing, we observed that the model frequently misclassified various movements as falls, confirming our concerns.

At this stage, we decided to introduce a third hidden layer, as we believed that the two-layer architecture might be too simplistic for this task. While adjusting the number of neurons in the existing layers was also a possibility, we anticipated that increasing model depth would yield better results. Building on the baseline model, which used 20 and 10 neurons in its hidden layers, we added an additional layer with 20 neurons to enhance the model's ability to distinguish between different types of movements. The best outcome with the standard learning rate of 0.0005 with 75 epochs gave the results shown in Figure 9.

	FALL	IDLE	SITTINGDOWN	STANDINGUP	WALK	UNCERTAIN
FALL	71.4%	0%	0%	0%	14.3%	14.3%
IDLE	0%	100%	0%	0%	0%	0%
SITTINGDOWN	66.7%	0%	0%	33.3%	0%	0%
STANDINGUP	0%	0%	0%	100%	0%	0%
WALK	0%	0%	0%	0%	100%	0%
F1 SCORE	0.71	1.00	0.00	0.86	0.95	

Figure 9: 3 Hidden dense layers (20, 10, 20), learning rate 0.0005 and 75 epochs

By adding a third dense layer, it shows that the model is better at detecting the different poses to their own class. This results in only “*SittingDown*” still being classified as “*Fall*”, which might have to do with both classes having a vertical acceleration downwards.

### 3.3.3. Final Model

From this point onward, we began experimenting with different neuron configurations to further refine the model. While not all variations are documented here due to redundancy, one configuration that yielded satisfactory results consisted of 64, 32, and 16 neurons in the hidden layers.

	FALL	IDLE	SITTINGDOWN	STANDINGUP	WALK
FALL	100%	0%	0%	0%	0%
IDLE	0%	100%	0%	0%	0%
SITTINGDOWN	33.3%	0%	0%	66.7%	0%
STANDINGUP	0%	0%	0%	100%	0%
WALK	0%	0%	0%	0%	100%
F1 SCORE	0.86	1.00	0.00	0.50	1.00

Figure 10: 3 Hidden dense layers (64, 32, 16), learning rate 0.0005 and 100 epochs

With a F1 score of 0.86 for the *Fall* class it performed well for the validation data, but it really stood out when performing live classifications, where the model was able to identify (almost) all of our falls. For now we decided to stop improving our model, since our set goal of having a device that is able to classify our fallings most of the time without having misclassifications has been accomplished.

Of course there is room for further work, this will be discussed in the Discussion section.

### 3.4 Model Deployment

After training and validation, the model was deployed to the Arduino Nano 33 BLE using Edge Impulse's deployment tools. The process included model optimization, where the trained model is converted into a TensorFlow Lite model optimized for microcontrollers, and on-device integration, where the model is flashed onto the Arduino, enabling real-time inference in a test scenario. It should be noted that the differences as shown in figure 11 are minimal and that there are still resources left, which might suggest that model architecture could be more complex. With the quantized model taking 20 kB of flash memory, there is still 980kB left according to the official documentation. Also, from the 1.4kB of RAM being in use, there is still use for another 254.6kB being left for further improvements [\[6\]](#).

Quantized (int8)			
Selected ✓			
	SPECTRAL FEATURES	CLASSIFIER	TOTAL
LATENCY	13 ms.	1 ms.	14 ms.
RAM	1.7K	1.4K	1.7K
FLASH	-	20.0K	-
ACCURACY			-

Unoptimized (float32)			
Select			
	SPECTRAL FEATURES	CLASSIFIER	TOTAL
LATENCY	13 ms.	2 ms.	15 ms.
RAM	1.7K	1.7K	1.7K
FLASH	-	30.3K	-
ACCURACY			-

Figure 11: Difference between optimized and unoptimized model

Just as during the live classification process, the Arduino was connected with the USB cable to the person. With the USB connection we could read out the predictions from the model, which gave us a better insight into the performances of the model in a real scenario. In a real setting it would be possible to have the Arduino operating on a battery without the need of a wired laptop connection, which eliminates the need for a cord. This, however, wasn't part of our scope, and having a wired connection made it less difficult to set up a serial connection with the device.

The best results were achieved when using a fixed time window of 3 seconds for predictions. Based on the set window from the train data, this ensured that enough sequential sensor data was analyzed to improve accuracy while maintaining a robust distinction between falls and normal activities. The second option was to use a moving time window, but these predictions lead to fewer "Fall" poses being predicted correctly.

## 4. Results

The final selected model, as discussed in the methodology, achieved an F1-score of 0.92 for the "fall" class on our test data, with a recall of 1. In theory this is an impressive result, indicating that the model is well-optimized for detecting falls. However, since the variety in the training data was somewhat limited, it was essential to conduct real-world testing to verify the model's actual performance.

	FALL	IDLE	SITTINGDOWN	STANDINGUP	WALK	UNCERTAIN
FALL	85.7%	0%	0%	0%	14.3%	0%
IDLE	0%	100%	0%	0%	0%	0%
SITTINGDOWN	0%	0%	0%	33.3%	0%	66.7%
STANDINGUP	0%	0%	0%	100%	0%	0%
WALK	0%	0%	0%	0%	100%	0%
F1 SCORE	0.92	1.00	0.00	0.86	1.00	

Figure 12: Confusion matrix for the test data

As noted in section 3.3, the live classifications of the model while attaching the device to the same person responsible for the training data, made the model correctly predict almost all of the fall incidents while performed by us. Additionally, when performing actions such as quickly sitting down on a chair or couch, no falls were falsely detected. This suggests that the model in practice still has a high precision and recall, which aligns with the strong F1-score observed during training. The classes “*StandingUp*” and “*SittingDown*” were classified as “*Walk*” all the time, but since the focus in this project was to detect falls, we didn’t want to focus too much on those results.

## 5. Conclusion

This study demonstrates that it is possible to distinguish falls from daily activities using a simple edge device with limited resources. The Arduino Nano 33 BLE is capable of running a machine learning model for fall detection in real time, and there are still computational resources available to increase the model's complexity if needed.

However, due to the limited variation in the training data, the model lacks robustness. While it performs well for the individuals whose data was used for training, its ability to generalize to a broader population remains uncertain. Expanding the dataset with more participants, diverse body types, and different sensor placements would likely improve the system’s reliability and reduce misclassifications.

The results show that a lightweight fall detection system on an embedded device is feasible. Future improvements should focus on enhancing dataset diversity and optimizing the model to ensure accurate and consistent fall detection across different real-world scenarios.

## 6. Discussion

While the results of this study are promising, there are several aspects in which the experiment could have been more comprehensive to improve the model’s robustness and reliability. Future research should focus on collecting a more extensive and diverse dataset and conducting real-world testing with multiple users. This would enhance the model’s generalizability and help develop a more reliable system that performs effectively across

different practical scenarios, since having just 2 people, with limited poses, isn't covering a wide spectrum of possibilities.

One potential improvement is increasing the variation in the dataset. Since the collected data came from a limited number of individuals, and the fall detection model was tested on the same person who contributed to the training data, the model may have partially learned person-specific movement patterns. To enhance generalizability, future work could involve collecting data from individuals with different heights, weights, and falling behaviors. Additionally, incorporating variable sensor placements (e.g., back, chest, or hip) could improve robustness by making the model less dependent on a single mounting position.

Additionally, the model's performance metrics were sometimes misleading. The confusion matrix suggested that the model performed exceptionally well, with high F1-scores and accurate classifications. However, during real-world testing, misclassifications occurred more frequently than expected. This indicates that the model may have overfit to the training data or struggled to distinguish subtle differences between movements. For example, while the model successfully detected falls in controlled test environments, it sometimes failed in real-world scenarios where fall patterns slightly deviated from the training data.

Moreover, the computational capacity of the Arduino Nano 33 BLE has not yet been fully utilized. This means there is room to implement a more complex model without exceeding hardware constraints. By increasing the number of layers or neurons, it may be possible to improve classification performance without significantly impacting real-time processing capabilities. Future research could explore more sophisticated architectures while maintaining a balance between accuracy and efficiency.

## References

1. Singh, R., & Gill, S. S. (2023). Edge AI: a survey. *Internet of Things and Cyber-Physical Systems*, 3, 71-92. <https://doi.org/10.1016/j.iotcps.2023.02.004>
2. Van Der Naald, N., Verbeek, F., Baden, D. N., Verbeek, A. J. M., Ham, W. H. W., Verbeek, J., Brummelkamp, E., Groenewoud, H., Niekerk, C. S., & Verbeek, A. (2024). Trends and projections in fall death in the Netherlands from 1990 to 2045. *Emergency Medicine Journal*, 41(7), 404–408. <https://doi.org/10.1136/emered-2023-213073>
3. Netherlands, S. (2019, December 5). 13 people a day die after a fall. Statistics Netherlands. <https://www.cbs.nl/en-gb/news/2019/49/13-people-a-day-die-after-a-fall> 2024, pp. 1-6, <https://doi.org/10.1109/apcit62007.2024.10673501>
4. Qiang Li, John A. Stankovic, Mark A. Hanson, Adam T. Barth, John Lach, and Gang Zhou. 2009. Accurate, Fast Fall Detection Using Gyroscopes and Accelerometer-Derived Posture Information. In *Proceedings of the 2009 Sixth International Workshop on Wearable and Implantable Body Sensor Networks (BSN '09)*. IEEE Computer Society, USA, 138–143. <https://doi.org/10.1109/BSN.2009.46>
5. A. Z. Rakhman, L. E. Nugroho, Widyawan and Kurnianingsih, "Fall detection system using accelerometer and gyroscope based on smartphone," 2014 The 1st International Conference on Information Technology, Computer, and Electrical Engineering, Semarang, Indonesia, 2014, pp. 99-104, <https://doi.org/10.1109/icitacee.2014.7065722>

6. Arduino. (n.d.). *Arduino Nano 33 BLE documentation*. Arduino. Retrieved February 21, 2025, from <https://docs.arduino.cc/hardware/nano-33-ble/>
7. N. Alushi. *Accessing Accelerometer Data on Nano 33 BLE*. Retrieved March 21, 2025, from <https://docs.arduino.cc/tutorials/nano-33-ble/imu-accelerometer/>
8. Krishnamoorthi, R. (2018). Quantizing deep convolutional networks for efficient inference: A whitepaper. arXiv preprint arXiv:1806.08342. <https://doi.org/10.48550/arXiv.1806.08342>
9. David, R., Duke, J., Jain, A., Janapa Reddi, V., Jeffries, N., Li, J., ... & Rhodes, R. (2021). Tensorflow lite micro: Embedded machine learning for tinymml systems. *Proceedings of Machine Learning and Systems*, 3, 800-811. <https://doi.org/10.48550/arXiv.2010.08678>