TinyML-Based Fall Detection System for Elderly Care

Team: Zhengyu Jin (zjin25@uw.edu), Xukang Wang (xkwang96@uw.edu), Zichen Huang (zh0215@uw.edu), Xiaoning Jing (xiaonj2@uw.edu)

Problem Statement

Falls are a major health risk for the elderly, often leading to severe injuries, long-term hospitalization, or even death. The number of elderly people living alone has been continuously growing worldwide. This independence comes with the risk of not receiving prompt attention if an accident occurs. Traditional fall detection systems based on vision or manual monitoring are costly, privacy-invasive, or lack real-time responsiveness. To address this issue, we aim to develop a compact, low-power, and real-time fall detection system using TinyML techniques deployed on a resource-constrained **Arduino Nano 33 BLE Sense** board.

Dataset

To develop and validate our fall detection model, we employ the **SisFall dataset**, a well-established and publicly available dataset specifically designed for fall detection research. SisFall contains acceleration data collected from a variety of fall scenarios (e.g., forward fall, backward fall, lateral fall) and daily activities (e.g., walking, sitting, picking up objects), performed by both young and elderly volunteers. The data is collected using wearable inertial sensors positioned near the waist, providing high-resolution 3-axis accelerometer signals that are well-suited for training lightweight machine learning models.

To be more specific, The **SisFall dataset** used in our project contains a rich collection of sensor data designed specifically for fall detection research. It includes 15 distinct fall types, each recorded across multiple trials and participants. For each fall type, approximately 60 trials are available, leading to a total data dimension of roughly **15 × 60 × 3000 × 9**, where:

- 3000 corresponds to 15 seconds of data sampled at 200 Hz per trial,
- The final 9 channels represent signals from three inertial measurement units (IMUs)—specifically, two accelerometers and one gyroscope, each providing 3-axis data (x, y, z).

Based on our inspection, the first six columns in each data sample likely represent the x/y/z axes of the two accelerometers, while the last three columns correspond to the gyroscope's x/y/z axes. Although the original paper mentions that only one accelerometer was used for prediction, the dataset provides complete 9-axis recordings, offering flexibility for selecting and fusing sensor modalities during model development.

This structural clarity allows us to perform sensor selection, feature engineering, and model training with fine-grained control, and it supports exploration of lightweight yet robust fall detection models suitable for real-time deployment on TinyML platforms like the Arduino Nano 33 BLE Sense.

Modeling Approach

Based on the SisFall dataset, we propose a mathematical model that incorporates time-frequency domain fusion deep learning algorithms utilizing IMU sensor data, along with a model compression framework to reduce computational complexity and enhance deployment efficiency.

- **Preprocessing**

A low-pass filtering module was applied to suppress high-frequency noise and spurious spikes. In addition, a fixed-length window of 3 seconds was used for segmentation. For each sample, the window was selected based on the segment with the maximum signal magnitude, ensuring that the most representative portion of the motion is retained for further analysis.
To further improve recognition accuracy, frequency-domain features (e.g., FFT-based spectral magnitude) can be optionally extracted and integrated into the following deep learning framework depending on the task requirements.

- **Model training's input**

The numbers of 3-second-length window are as followed:
Activities of Daily Living (ADL) data: 947 * 2 * 3
Fall data: 900 * 2 * 3

- **Neural Networks**

The SisFall dataset is well suited for fall detection using models based on Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM).

CNN: In our model design, we selected CNN because the SisFall data is highly sampled (200 Hz) and contains many localized dynamics, such as sudden vibrations and posture changes. CNN is particularly effective at extracting these localized features and identifying key moments in the fall process. Furthermore, the data, collected from multiple sensors with 9 dimensions, enables CNN to learn inter-sensor correlations, which enhances the overall detection performance.

LSTM: The dataset, collected by wearable IMUs, contains continuous acceleration and angular velocity recordings with clear time-series characteristics. Falls, which progress through phases like smooth movement, sudden impact, and stationary, exhibit strong temporal dependencies. LSTM models effectively capture these dependencies, allowing for more accurate differentiation between falls and daily activities.

CNN & LSTM: Local feature extraction and general behavioral comprehension can be enhanced by the combination of CNN and LSTM. Whereas LSTM records temporal relationships, CNN extracts local spatial characteristics like acceleration and angular velocity variations. The two can work together to increase the model's durability and detection accuracy, particularly for precise fall event recognition in complicated situations.

Additionally, while the model is being implemented, we will attempt to incrementally lighten the model by adopting smaller channel convolutions and lowering the number of layers and channels. This will allow us to examine how alternative model structures affect the model's performance.

- **Compression**

Quantization: We planned to employ post-training quantization (PTQ) by changing the model weights and activations from FP32 to INT8 in order to decrease the size of the model and increase inference efficiency. Without sacrificing accuracy, this will drastically reduce memory consumption and speed up processing.

(Optional) Pruning: We will also consider structured pruning, optionally removing 30%–50% of less important weights or channels based on magnitude or sensitivity analysis. This sparsification improves inference speed and lowers computational cost, particularly when backed by efficient libraries or hardware.

Deployment Plan

Once the best-performing model has been chosen—whether it is a tiny 1-D CNN, a LSTM, or another lightweight architecture—we will export it as an INT8 TensorFlow Lite for Microcontrollers binary and flash it, together with a minimal ring-buffer preprocessing routine, to the Nano 33 BLE Sense's 1 MB Flash (peak RAM < 30 KB).

The firmware will sample the LSM9DS1 IMU at 50 Hz, apply a 4-order 5 Hz Butterworth low-pass filter and per-axis normalization, then run inferences in sliding 3-second windows every 0.5 s; a fall verdict will light the onboard LED and send a BLE "FALL" packet to a companion app.

Evaluation will proceed in two stages: offline, we will report sensitivity, specificity and F1 on a held-out SisFall test split plus a leave-one-subject-out cross-validation to gauge generalization; on-device, we will replay sensor traces and perform staged fall/ADL reenactments, logging latency (< 150 ms target), false-alarm rate, and current draw (< 6 mA average) to ensure 3-hour battery life on a 500 mAh cell. Thresholds and sample rate can be tuned in firmware to trade detection-rate against power should the chosen model overshoot or undershoot these targets.

Challenges & Solutions

Challenge 1: The Arduino Nano 33 BLE Sense has limited memory, restricting the size and complexity of deployable models.
Solution: To address the hardware constraints of the Arduino Nano 33 BLE Sense, we plan to explore several model optimization strategies. We intend to evaluate lightweight neural network architectures such as Tiny-CNN or shallow MLP, which are known for their low parameter counts and computational demands. We also propose to apply quantization techniques to convert model weights and activations to INT8 precision, thereby reducing memory usage and enabling fixed-point computation. In addition, we will consider using model pruning to remove redundant parameters and further compress the model without significantly sacrificing accuracy. The final model will be exported using TensorFlow Lite for Microcontrollers to ensure it can operate efficiently within the board's flash and RAM limitations.

Challenge 2: The system must process 1-second sliding windows and make predictions every 0.5 seconds, with a target latency of under 150 ms.
Solution: To meet the real-time processing requirements, we plan to implement a ring-buffer structure for efficient ingestion and storage of streaming IMU data, which will support sliding window operations without repeated memory allocation. We also aim to perform data normalization and low-pass filtering immediately on the device after sampling, minimizing redundant preprocessing and ensuring consistent signal quality. Additionally, we propose to reduce the input dimensionality by selectively utilizing the most relevant sensor channels, which will help decrease computational load and improve inference speed, thus allowing the system to meet its latency target.

Challenge 3: It's impractical and unsafe to test real fall events frequently.
Solution: Given the challenges in collecting authentic fall data, we rely on replaying high-fidelity recordings from the SisFall dataset to evaluate system performance in controlled tests. Additionally, we conduct staged reenactments of fall-like movements, such as rapid sitting or stumbling, to simulate real-world use cases. These simulations are complemented by extensive logging and analysis of onboard sensor data, allowing us to iteratively refine the model and firmware without endangering human subjects.