Analyzing and Comparing Deep Learning models on a ARM 32 bits microcontroller for pre-impact fall detection

Aurélien Benoit, Christophe Escriba, David Gauchard, Alain Estève, Carole Rossi

▶ **To cite this version:**

Aurélien Benoit, Christophe Escriba, David Gauchard, Alain Estève, Carole Rossi. Analyzing and Comparing Deep Learning models on a ARM 32 bits microcontroller for pre-impact fall detection. IEEE Sensors Journal, inPress, 10.1109/JSEN.2024.3364249 . hal-04462710

HAL Id: hal-04462710
https://hal.science/hal-04462710

Submitted on 16 Feb 2024

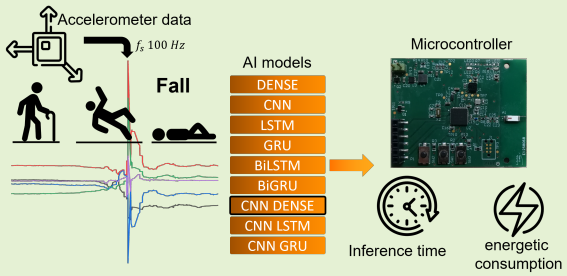
HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Analyzing and Comparing Deep Learning models on a ARM 32 bits microcontroller for pre-impact fall detection

Aurélien Benoit, Christophe Escriba, David Gauchard, Alain Esteve and Carole Rossi

***Abstract*— Automated pre-impact fall detection in real-time using raw data acquired from wearable sensors, such as tri-axial accelerometers, remains an open research problem in the context of elderly care. This paper presents a comparative study of nine neural network models, including Dense, CNN, LSTM, GRU, BiLSTM, BiGRU, CNN Dense, CNN LSTM, and CNN GRU, for predicting an occurring fall before impact with the ground using accelerometer data. The models were optimized using Keras Tuner with the TensorFlow backend, a dominant deep learning software framework. Machine learning classifiers suitable for execution on microcontrollers were built and evaluated based on performance metrics such as accuracy, sensitivity, specificity, storage size, inference time, and energy consumption. The results highlight that the CNN DENSE algorithm provides the best detection accuracy (94.70%) with a lead time of 76.91 ms. Sensitivity and specificity reach 95.33% and 94.18%. The energy consumption and inference time are 6.72 mA and 12.88 ms, respectively. In conclusion, deep learning demonstrates increased classification accuracy and a simplified software architecture, but it comes at the cost of decreased energy efficiency and inference speed. These factors are crucial considerations in developing on-demand fall injury prevention wearable systems.**



***Index Terms*— Deep Learning, Energy consumption, pre-impact fall detection, Tensorflow lite, threshold-based, wearable systems.**

I. Introduction

OVER the past century, the average life expectancy has increased dramatically. As a result, the safety of the elderly has become a major societal concern. In particular, the increased risk of falls is one such distinct health risk factor. Unintentional falls are the leading cause of fatal injuries and the most common cause of nonfatal trauma-related hospital admissions among adults over 65 years-old [1]. Between 2015 and 2050, the proportion of the world's population aged over 60 will almost double, from 12% to 22%. Each year, approximately 684,000 people die as a result of a fall and 37.3 million of falls are recorded that are severe enough to require medical attention. In the United States of America, elderly falls costed $50 billion in 2015, and $43 billion by the year 2020. Every year in France, falls among the elderly result in over 130,000 hospitalizations and 10,000 deaths. The cost of falls among the elderly is currently estimated at 2 billion euros, including 1.5 billion for the french health insurance system.

Manuscript received xxxx xx, xxxx; revised xxxx xx, xxxx; accepted xxxxxx xx, xxxx. Date of publication November xx, xxxx; date of current version xxxxx xx, xxxx.

The authors are with the Laboratory for Analysis and Architecture of Systems (LAAS-CNRS), University of Toulouse, 31077 Toulouse, France (e-mail: abenoit@laas.fr; rossi@laas.fr; cescriba@laas.fr; aesteve@laas.fr; gauchard@laas.fr).

Digital Object Identifier xx.xxxxx/JSEN.xxxx.xxxxxxx

In this context, wearable sensors, such as 3D accelerometers, that can monitor daily activities, quickly identify occurring falls and send out remote notifications, are crucial to elderly care, allowing for earlier assistance. One major hurdle to the deployment of wearable sensors on elderly is the accuracy of pre-impact fall detection system, often generating false alarms,i.e., alerting fall during activities of daily living (ADL). Thus, most of recent researches have focused on reducing false alarms and improving accuracy of the pre-impact fall detection system [2]–[4] using either threshold-based algorithm [5]–[18], conventional machine learning (ML) [15], [19], or deep learning approaches [20]–[36]. Threshold-based algorithms have demonstrated good performances to distinguish pre-impact falls from ADL: sensitivities are in the range of 92% - 99%, specificities are in the range of 95% - 98% with an average lead time (time interval between fall detection and body-ground impact occurrences) in the range of 250 - 400 ms. Two interesting papers present exceptional results with a sensitivity and specificity $>$ 97% [17], [18] but the algorithms were evaluated on a very small sample of people. Despite the good sensitivity and specificity, threshold-based algorithms lack of generalizability and robustness when applied to real scenarios or different groups of persons.
Conventional ML algorithms such as support vector machine (SVM) were explored to detect falls using, as inputs, highly correlated parameters among raw signals of acceleration and

angular velocity. Detection sensitivity $> 95\%$ and specificity $> 90\%$ with a lead time ranging from 63 to 200 ms were demonstrated. For example, *Xie et al.* [15] used a smartwatch using the SVM method and demonstrated 100%, 95.5% and 95.7% forward, backward and lateral fall accuracy, respectively, with an average processing time of 276 ms, which is still too long to envisage a real application for pre-impact fall detection.

More recently, as computing power in hardware, especially GPUs, has advanced rapidly, deep learning-based algorithms such as Convolutional Neural Networks (CNN), Adaboost, and Long Short-Term Memory (LSTM) have been explored for pre-impact fall detection [20]–[36]. The significant advantage of these deep learning approaches is their ability to process raw data directly from wearable sensors and automatically extract high-level motion features through specifically designed network layers.

However, it is important to note that while these works have evaluated their model performances on computers, there is a gap in the literature concerning the assessment of deep learning models, including Recurrent Neural Networks (RNN), LSTM, CNN, and CNN-LSTM, on microcontrollers. It is worth mentioning that there are instances in the literature where deep learning models have been successfully implemented on low-power microcontrollers for various applications, such as spoken keyword spotting [37], human activity recognition [38], [39], gesture recognition [40], acoustic event detection [41], post-stroke assistance and rehabilitation [42], flow analysis in public spaces [43], nutrition monitoring [44], sports activities classification [45], and even for industrial applications [46], [47]. The performance of these models, in terms of power consumption, accuracy, precision, and sensitivity, varies based on the specific models and problems they address.

This comprehensive review of existing research highlights the absence of studies developing pre-impact fall detection systems using deep-learning models integrated into microcontrollers for elderly care. Notably, the work by Yu et al. [4] is the only one that employs an embedded micro-computer (Raspberry Pi 4) to measure lead time and computation latency on a test set.

This paper aims to fill this gap by conducting a thorough comparison of nine deep learning models (Dense, CNN, LSTM, GRU, BiLSTM, BiGRU, CNN Dense, CNN LSTM, CNN GRU) integrated into an ultra-low-power ARM Cortex M4 microcontroller. The evaluation focuses on quantifying the models' performances for pre-impact fall detection using raw data collected from a 3-axis accelerometer. Additionally, inference time and energy consumption are measured for each model integrated into the microcontroller. As a preliminary step to the present investigation, a comparison of optimizers was performed, although the details are not covered in this paper. The major contributions of this research include:

- the first comprehensive comparison of different algorithms for pre-impact fall detection based on a large-scale motion dataset from nine young subjects, encompassing three types of ADL and four types of falls. The comparison extends beyond commonly used accuracy-related measures (e.g., accuracy, sensitivity, and specificity) to include model storage size, inference time, and energy consumption—crucial parameters in designing an on-demand fall injury prevention system for the elderly.
- Pioneering research in determining which algorithms can effectively run on a microcontroller to detect pre-impact fall situations, while also assessing the current limitations of deep learning techniques for pre-impact fall detection due to their high energy consumption efficiency.

This study offers valuable insights for researchers, shedding light on the strengths and limitations of each machine learning method. It also outlines pathways for integrating these methods into autonomous systems, representing a significant step towards the design of future products to address societal needs.

II. Materials and Methods

A. Data collection and preprocessing

We used recently established Kfall public fall databases [48] for the evaluation of the different models. The Kfall dataset was collected from 32 healthy young males (age: 24.9 ± 3.7 years; height: 174.0 ± 6.3 cm; weight: 69.3 ± 9.5 kg) using a 9-axis inertial sensor attached to the lower back of each subject and sampled at 100 Hz. The dataset also includes 21 Activities of Daily Living (ADLs) and 15 simulated falls. Only the three axes of the inertial sensor were utilized in this study. Furthermore, since the Kfalls series is better suited for post-fall detection rather than pre-impact fall detection, we augmented the model training process by incorporating a custom-made dataset from nine volunteers (2 females, 7 males) in good health and without any physical conditions (average height 174.11±6.98 cm; average weight 63.57±7.91 kg). They were equipped with a circuit placed on a belt (Figure 1). Data were recorded using a specific circuitry integrating an nRF52832 System-on-Chip (SoC) from Nordic Semiconductor and a 3-axis accelerometer LIS3DH from STMicroelectronics at 100 Hz. The system description can be found in [5]. We recorded four types of falls, namely forward, backward, lateral right and lateral left, and three ADL, i.e. walking, sitting, picking up an object. Overall, this corresponds to a total of 474 acquisitions, including 188 series of ADL and 286 series of falls (a forward fall signature is plotted in Figure 2). Then, the dataset is divided into test and train sets. The train set includes 80% of the overall dataset (7 volunteers randomly chosen) while the remaining 20% of the dataset (2 volunteers) are devoted to the test set. As a result, the train set is composed of 364 series (140 ADL and 218 falls), and the test set is composed of 110 series (48 ADL and 68 falls). A "tag connect" plug was integrated to reduce the hardware footprint and programmed by Serial Wire Debug (SWD). The entire software project was built with Segger Embbeded Studio. Our system used a 512 kB FLASH and a 64 kB RAM. Note that the addition of the NORDIC and TFLM libraries occupies 29.8 kB and 181.4 kB of RAM and FLASH memories, respectively. Table I summarizes the characteristics of the accelerometers used to build the two different databases (Kfall and ours).

Then, the collected data from both custom-made and Kfall datasets were pre-processed following 2 steps.

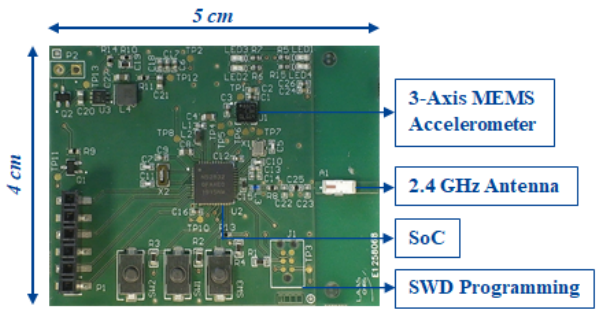


Fig. 1. *Prototype of the pre-impact fall detection circuitry*

TABLE I
CHARACTERISTICS OF THE ACCELEROMETERS USED TO BUILD THE TWO DIFFERENT DATABASES (KFALL AND OURS)

Features sensors	Kfall	Our dataset
Sampling frequency	100 Hz	100 Hz
Sensibility	16 g	8 g
Resolution	16 bits	12 bits

Step 1. Definition of the length of time series

First, all data from the dataset were concatenated into a single row in a parquet file. Each data point within each series was labeled based on the type of movement (fall or ADL). Each file contains 8 columns, consisting of:

- *Time* : time step
- a_x, a_y, a_z : components of the acceleration along x, y, z axes
- *SVM* : sum vector magnitude of the acceleration
- *tiltAngle* : Tilt angle from the y-axis relative to the fall
- *user* : name of volunteer
- *type* : type of movement
- *Test* : test number
- *label* : class for output classification models

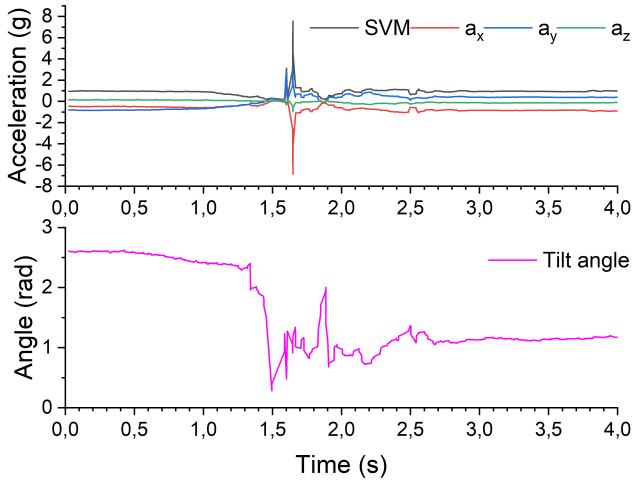


Fig. 2. Forward fall, time evolution of : (top) components of the acceleration along axis x, y, z and sum vector magnitude (SVM), (bottom) tilt angle

The *tiltAngle* (in rad) is calculated using Equation 1 [5] :

$$tiltAngle = cos^{-1}\left(\frac{a_y(t)}{\sqrt{a_x^2(t) + a_y^2(t) + a_z^2(t)}}\right) \quad (1)$$

From each data file, we divided the overall dataset into equal frames corresponding to 25 acquisition steps, with an overlap of one acquisition step considered between two adjacent frames (referred to as a "sliding window"). This multi-frame approach has been found to enhance the performance of the ML models, reducing the amount of data treated as input for each layer of the models and capturing time variations in features. The input dataset was structured with 3 columns *(sample, time-steps, features)*, where each input sample contains 5 features (a_x, a_y, a_z, *SVM*, *tiltAngle*) composed of 25 values each.

Moreover, automated extraction of temporal features from various types of fall data collected using wearable sensors has been demonstrated in previous studies [31]. Finally, since the objective is to classify whether a fall occurred or not, each series is labeled as either "ADL" or "fall" based on the labels that occur most frequently. It is important to note that this labeling may differ from the label of some individual data points, as there are series labeled as "ADL" containing points labeled as "fall" and vice versa. These cases mostly occur at the end of each acquisition series.

Step 2. Normalization

Normalization corresponds to the process of transforming data of different scales to a common scale, typically within the range [0, 1]. This is done to balance the impact of each feature on the model and expedite model convergence [49] [50]. For each feature x_i, the normalization involves subtracting the median value x_{med} and dividing it by the interquartile range ($Q_{75} - Q_{25}$). The normalized value X is obtained using the equation 2 :

$$X = \frac{x_i - x_{med}}{Q_{75} - Q_{25}} \quad (2)$$

We opted for the Robust Scaler normalization method following a comparison with both the Standard Scaler and MinMax Scaler methods (Figure 3). The Robust Scaler is considered the most appropriate for large and wide datasets, as it effectively handles outliers that could significantly impact predictions. Unlike the Standard Scaler, which transforms data to have a mean of 0 and a standard deviation of 1 (suitable for normally distributed data), and the MinMax Scaler, which scales data to a specified range typically between 0 and 1 (preserving the original data's relationships), the Robust Scaler provides robustness against outliers.

The plots in Figure 3 illustrate the data distribution before and after the normalization step. Prior to normalization, raw data are within a small range of values [-2.5, 2.5], but they are not centered around 0, and each feature has a different disparity of values. MinMax scaling spreads the features over the range [0, 1]. Acceleration components on the x, y, and z axes are centered around 0.5, SVM around 0.1, and tilt angle is spread across the range with two density peaks at 0.5 and 0.95. With standardization, data are scaled with a mean value of 0,

and density peaks are present in the [-5, 5] range. By removing the median and spreading the features over the interquartile range (IQR), features are centered around 0 with density peaks in the [-5, 5] range. It's observed that Standard Scaler and Robust Scaler have data at equivalent density scales: [0, 2]. However, Robust Scaler spreads certain features, providing a more balanced representation with uniformly distributed density peaks. In conclusion, Robust Scaler is the most suitable normalization method for the specific problem of pre-impact fall detection, as it balances the representation and facilitates effective model learning.

B. Models construction, optimization and training

In this study, nine ML models were evaluated, namely Dense, CNN, LSTM, GRU, BiLSTM, BiGRU, CNN Dense, CNN LSTM, CNN GRU, that were trained and run using Tensorflow 2.10.

Dense model was chosen as it is the base layer characterized by interlayer connections between adjacent layers and performing linear transformations on input data for the next layer, mainly for the output layer. The CNN model [51], [52] is commonly used for computer vision tasks such as object detection, image segmentation, video recognition [53] or optimization energy consumption for embedded systems [54]. The LSTM model is suitable to classification and prediction of future values based on time series data in anomaly detection, stock prediction or gesture classification [55]. Hybrid models combining Convolutional Neural Networks (CNNs) with Recurrent Neural Networks (RNNs) such as Long Short-Term Memory (LSTM) [56], [57] or Gated Recurrent Unit (GRU) layers have been widely used in various image and natural language processing tasks [58]–[61], including speech emotion recognition and movie sentiment analysis. These models leverage the strengths of both CNNs and RNNs to capture spatial features through convolutional layers and temporal dependencies through recurrent layers. The forward and backward LSTMs are trained separately, but their outputs are combined to produce the final prediction. The output at each time step is obtained by concatenating the forward and backward hidden states, allowing the network to capture both past and future context. Bidirectional GRU (BiGRU) does exactly the same, but employing forward and backward GRU's [62].

1) Choice of the optimizers: the choice of optimizers plays a crucial role in the training process, and evaluating multiple optimizers is a good practice. We used a Bayesian optimization strategy to explore the hyperparameter space and find the optimal configuration for our hybrid model. 7 optimizers were evaluated : Stochastic Gradient Descent (SGD) [63], AdAptive Gradient (AdaGrad) [64], Root Mean Square Propagation (RMSProp) [65], AdAptive Gradient Delta (AdaDelta) [66], AdAptive Moment estimation (Adam) [67], Nesterov AdAptive Moment estimation (Nadam) [68], AdAptive Moment Weight (AdamW) [69]. The optimal choices for pre-impact fall detection application were found to be Adam, Nadam, and AdamW optimizers. Among these, Adam emerged as the most suitable choice due to its integration of momentum, facilitating accelerated convergence, and RMSProp, contributing to the normalization of learning rates. These characteristics make Adam particularly well-suited for addressing the challenges inherent in temporal binary classification problems, where features exhibit variations over time, often characterized by sparse or non-uniform behaviors [20], [31], [32], [34], [70]–[72]. The adaptability of Adam to capture temporal dynamics aligns seamlessly with the nature of our application, enhancing the effectiveness of our hybrid model in detecting falls prior to impact.

2) Kernel initializer, Activation functions and hyperparameters tuning: According to [73], the activation functions Relu and Leaky Relu must be initialized with He. Although ReLU is widely used due to its simplicity and efficiency, it provides zero activation for negative values. This leads to the phenomenon called "dead neurons" or "dying ReLU" as negative neurons do not contribute to learning. To avoid this problem, the Leaky ReLU activation function is used to assign a slope of 0.1 to negative neurons. Hence, when a activation is negative, neurons fall into a "sleep" state but continue to receive gradient updates.

TABLE II
SEARCH SPACE FOR THE OPTIMIZATION OF HYPERPARAMETERS

Model type	Parameters	Range
All	Learning rate	$[10^{-2},10^{-6}]$
	Number of hiddens	[1, 5]
	Optimizers	[Adam,Nadam,AdamW]
Dense	Number of neurons	[16, 128]
CNN	Number of filters	[8, 128]
	Number of kernel_size	[1, 10]
	Number of strides	1
GRU, LSTM	Number of neurons	[16, 256]
	Dropout	[0.1, 0.5]

3) Hyperparameters: Hyperparameters summary is presented in the Table III and final models are presented in the Tables IV and V. Each of them was trained on 200 epochs with Adam optimizer and batch size of 128. Binary cross-entropy was used. In the case of no improvement in the validation loss after 10 epochs, we used a call to stop the training process.

TABLE III
SUMMARY OF HYPERPARAMETERS

Parameters	Model type	Range
Learning rate	Dense	0.004
	CNN	0.001
	LSTM	0.001
	GRU	0.00001
	BiLSTM	0.0005
	BiGRU	0.0006
	CNN DENSE	0.001
	CNN LSTM	0.001
	CNN GRU	0.001
Optimizer	Adam	
Input Shape	25,5	
Batch size	128	
Epochs	200	
Loss function	Binary cross-entropy	

4) Training Environmental Configuration: in this study, the 16 GB graphics processor unit of NVIDIA RTX A4500 and the NVIDIA CUDA Deep Neural Network library (cuDNN), a

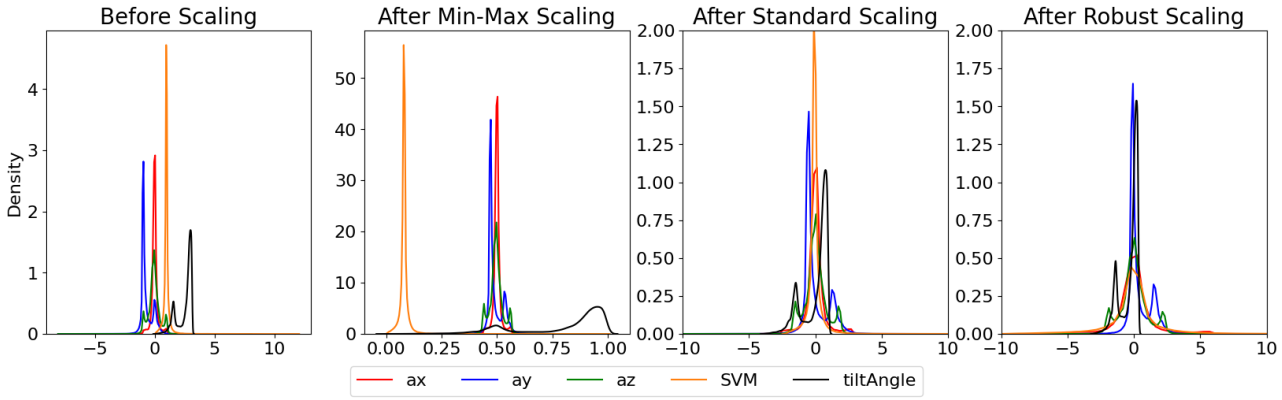


Fig. 3. Comparison of normalisation methods, namely MinMax Scaler, Standard Scaler and Robust Scaler techniques, on dataset (train and test set)

GPU-accelerated library, were used. This study was achieved under the Windows 10 as OS environment, using Intel Core i9-12950HX as CPU and 64Go 4800MHz DDR5 as RAM. The experiment was programmed using Python v3.10.10, Tensorflow 2.10 and CUDA v11.2.152 graphic cards.

5) Porting models into ARM 32 bit Microcontroller: All models with parameters outlined in the provided Table II, underwent reduction procedures as detailed in [74]. Subsequently, these optimized models were implemented on the microcontroller using the TensorFlow Lite for Microcontrollers (TFLite Micro) library. This implementation involved leveraging configuration optimizations to enhance efficiency and streamline the deployment process. Procedure is :

- TARGET=cortex_m_generic : selects the target microcontroller family.
- TARGET_ARCH=cortex-m4+fp : enables support for hardware floating point on cortex-m4.
- OPTIMIZED_KERNEL_DIR=cmsis_nn : enables the use of the CMSIS-NN library optimizations for performance boost on ARM Cortex core microcontrollers.

Reduction or quantization is the process of converting the weights and activations of a neural network from high-precision floating-point numbers to lower-precision representations, resulting in lighter models. This process is crucial for deploying deep learning models on resource-constrained embedded systems, such as our system. Prior to quantization, the models utilize 32-bit floating-point numbers (float32) for representing weights and activations, ensuring high precision. We conducted tests on three quantification techniques—Optimized Integral type with floating fold, Float16 Quantization, and Float8 Quantization. For each model, we selected the quantization method that produced the lightest model. Specifically, Float8 Quantization was employed for Dense, CNN, BiLSTM, CNN DENSE, CNN LSTM, and CNN GRU models. For LSTM, GRU, BiGRU, the Optimized Integral type with floating fold method was chosen.

The memory size before and after quantization is reported in the Table VI. We observed that the float8 models experience size reduction without introducing any additional operators. In contrast, integer models include two extra operators, namely ”Quantize” and ”Dequantize.” The architectures of float16 models exhibit the highest complexity. In the case of CNNs, multiple ”Dequantize” operators are integrated as inputs to the ”Conv2D” operators, and a similar pattern is observed for FullyConnected operators in DENSE models. For recurrent models featuring LSTM and GRU layers, the LSTM and GRU operators are transformed into numerous elementary operators, accompanied by the inclusion of ”while” operators, each containing over twenty operators. Additionally, ”Dequantize” and ”Quantize” operators have been incorporated. The necessity for including Tensorflow operators, in addition to Tensorflow Lite operators, arises during quantization to float. Importantly, the use of signatures before conversion becomes impractical in this scenario. Throughout the remainder of the paper, our focus is exclusively on quantized models, denoting reduced models. The programming languages employed were C and C++. The nRF5 v17.1.0 Software Development Kit (SDK) facilitated Bluetooth low energy and 2.4 GHz support compatibility.

For the nRF52832 System-on-Chip (SoC), the SoftDevice version employed was v7.2.0, and the Segger Embedded Studio version used was v7.10a. Notably, BiLSTM models were not implemented on ARM microcontrollers using TFlite Micro due to the lack of support for the ”REVERSE_V2” operation, essential for inverting data windows before the ”UNIDIRECTIONAL_SEQUENCE_LSTM” operation. To overcome this limitation, we developed a custom ”REVERSE_V2” operator for Tensorflow Lite for microcontrollers. The implementation underwent rigorous unit testing to ensure its functionality, using Bazel v6.2.0. To maintain consistent code formatting in Google style, clang-format v16.0.4 was applied. After completing the tests, the entire library was recompiled. Two key metrics guided our testing process: functional coverage and linear coverage. Functional coverage was employed to verify that all functions were adequately covered, ensuring their correct implementation. Linear coverage, on the other hand, focused on source code coverage, measuring the execution of specific lines of code during the tests. This metric provided insights into the portions of the source code that were executed at least once during testing. Hence, we designed a set of six unit tests, encompassing 2 tests for each compatible input and

output type—float32, int8, and int16. For each data type, one test involved a 1D tensor, while the other utilized a 3D tensor. Through these tests, we achieved comprehensive functional coverage, ensuring that all functions were systematically covered. Additionally, our testing process resulted in a commendable linear coverage of 83.9%, signifying the extent to which specific lines of source code were executed during the testing phase

6) Threshold method in post processing: Upon detecting false alarms in the ADL series concerning pre-impact fall detection, we identified a noteworthy correlation between the variance of the sum of acceleration vectors (SVM) after normalization and actual fall events. In response to this observation, we introduced a filtering condition aimed at mitigating false alarms. This condition involves verifying whether the variance of the classification window surpasses a threshold of 2 (after normalization), equivalent to 1.156713 g. The normalization conversion for raw accelerometer data is represented by Equation 3.:

$$X_{norm} = \frac{x_i - cv}{sv} \tag{3}$$

with X_{norm} normalize value, x_i, raw value, cv, center value of 0.9622855 and scaler value of value 0.09721375. This is a filtering post-processing step. After each window prediction, the SVM variance (Equation 4) is calculated and, if it exceeds the threshold, we consider that a fall is starting.

$$variance = \frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N} \tag{4}$$

with μ, the mean value of the ensemble of the data from one window.

It's important to highlight that the chosen threshold value not only enhances the sensitivity, accuracy, and specificity of pre-fall detection (metrics defined in the next section) but also influences the lead time. The optimal threshold value was identified as 2, ensuring a lead time greater than 100 ms. This time window allows sufficient duration for triggering the inflation of an airbag before making contact with the ground. It's noteworthy that the same threshold is consistently applied across all models for the sake of comparison. This approach aligns with the methodology used by Shi et al. in their work on pre-impact fall detection [34], where a post-processing threshold was also incorporated.

III. Results and Discussion

In the following we consider some important outputs as basis for the analysis: True Positives (TP), which correspond to correctly classified falls before impact, True Negatives (TN), which correspond to correctly classified ADLs, False Positives (FP), which correspond to misclassified falls in ADLs and False Negatives (FN), which correspond to misclassified ADLs in falls. From these outputs, we calculate three metrics:

- **Sensitivity (SEN)** defined as the ratio of the number of correctly classified falls out of the total number of falls, calculated as :

$$SEN = \frac{TP}{TP + FN}$$

This metrics informs if the fall is correctly detected.

- **Specificity (SPE)** defined as the ratio of the number of correctly classified adls out of the total number of adls.

$$SPE = \frac{TN}{TN + FP}$$

Which quantifies the ability of the model to correctly distinguish ADLs from falls.

- **Accuracy (ACC)** defined as the ratio of the number of correct predictions out of the total number of samples :

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

This metric quantifies the model ability to correctly identify real falls and ADL

- **F1-score** defined as the harmonic mean of precision and sensitivity. It compares the correct positive predictions TP with the errors $(FN + FP)$ of the model.

$$F1 - score = \frac{TP}{TP + 0.5(FP + FN)}$$

A high F1-score signifies the effectiveness of the experimental method.

The memory sizes associated with the various models are also provided in kilobytes (kB).

TABLE IV
DETAILS OF OPTIMIZED MODELS

Block	layer	Parameters	Values
DENSE			
flatten	-	-	-
dense	1	neurons	104
	2	-	88
	3	-	24
	4	-	1
CNN			
conv	conv1d	filters	64
		kernel size	3
		stride	1
	maxpooling1d		2
	conv1d	filters	32
		kernel size	3
		stride	1
	maxpooling1d		2
	conv1d	filters	64
		kernel size	4
		stride	1
	maxpooling1d		2
	conv1d	filters	32
		kernel size	3
		stride	1
	maxpooling1d		2
dense	dense 1	neurons	1
LSTM			
lstm	1	neurons	48
	2	-	112
	3	-	112
	dropout		0.1
dense	1	-	1
GRU			
gru	1	neurons	48
dense	1	-	1
BiLSTM			
bidirectional	lstm	neurons	224
dense	1	-	112
	dropout		0.1
dense	2	-	1
BiGRU			
bidirectional	gru	neurons	256
dense	1	-	128
	dropout		0.1
dense	2	-	1

TABLE V
DETAILS OF OPTIMIZED MODELS (CONTINUED)

Block	layer	Parameters	Values
CNN Dense			
conv	conv1d	filters	64
		kernel size	4
		stride	1
	maxpooling1d		2
	conv1d	filters	64
		kernel size	2
		stride	1
	maxpooling1d		2
	conv1d	filters	64
		kernel size	4
		stride	1
	maxpooling1d		2
	conv1d	filters	32
		kernel size	4
		stride	1
	maxpooling1d		2
dense	1	neurons	32
flatten	-	-	-
	dense 2	-	1
CNN LSTM			
conv	conv1d	filters	128
		kernel size	7
		stride	1
	maxpooling1d		2
	conv1d 2	filters	128
		kernel size	7
		stride	1
	maxpooling1d 2		2
lstm	1	neurons	48
	dropout		0.1
flatten	-	-	-
dense	1	-	1
CNN GRU			
conv	conv1d	filters	96
		kernel size	7
		stride	1
	maxpooling1d		2
	conv1d 2	filters	96
		kernel size	7
		stride	1
	maxpooling1d 2		2
gru	1	neurons	64
flatten	-	-	-
dense	1	-	1

A. Benchmarking models for pre-impact fall detections

Table VII presents the performance of different models in detecting a fall before reaching the ground. To benchmark the various models, we exclusively utilize the KFall dataset. It is important to highlight that, in the event of a fall, a positive lead time, indicating the existence of a time interval between detection and ground impact, results in a True Positive (TP) value. Conversely, if no such time interval is present, it is categorized as a False Negative (FN). Similarly, in the context of ADL, a False Positive (FP) is assigned if at least one window is classified as a fall, while a True Negative (TN) is recorded if no such classification occurs.

The CNN, Dense, BiLSTM, and CNN Dense models exhibit the highest accuracy, achieving 92.61%, 85.96%, 86.90%, and 94.70%, respectively. Regarding sensitivity, the CNN, LSTM, and CNN LSTM models demonstrate the highest scores at 92.33%, 98.97%, and 96.80%. In terms of specificity, the CNN DENSE , GRU, and BiLSTM models perform the best with scores of 94.18%, 97.25%, and 94.41%, respectively. The table also includes information on the average lead time in milliseconds (ms). Notably, the LSTM, BiGRU, and CNN LSTM models yield the most substantial lead times. In terms of memory usage, the most resource-intensive models are LSTM, CNN LSTM, and BiLSTM, utilizing 741.768 kB, 1011.928 kB, and 531.664 kB, respectively. On the other hand, the most memory-efficient models are GRU, CNN, and Dense, with consumption rates of 40.536 kB, 99.160 kB, and 101.220 kB, respectively.

We also assessed the inference time and number of CPU cycles for the models running on both the computer and the microcontroller (refer to Table VIII). The runtimes can be categorized into three groups: Dense, CNN models exhibit the most optimal performances with 0.138 ms, 0.713 ms, respectively, on the computer. Upon adding a dense layer, the CNN's runtime extends to 0.723 ms. However, BiLSTM performs 100 times slower on the microcontroller than on the

TABLE VI
DETAILS OF MEMORY SIZE REDUCTION FOR TENSORFLOW LITE MODELS (IN KILOBYTES, KB)

Model	Dense	LSTM	GRU	CNN	BiLSTM	BiGRU	CNN DENSE	CNN LSTM	CNN GRU
Unoptimized	101.220	741.768	40.536	99.160	531.664	567.160	101.864	1011.928	409.976
Optimized Integral type with floating fold	29.304	**196.632**	**21.376**	40.856	154.816	**159.640**	42.616	268.256	122.528
Optimized 16 bits	53.368	392.276	27.256	59.576	280.496	292.768	62.776	517.512	214.060
Optimized 8 bits	**28.600**	198.720	22.312	**39.352**	**142.888**	162.664	**38.968**	**264.656**	**118.82**
Reduction(%)	71.74	73.20	47.26	60.31	70.88	71.31	61.74	73.84	71.01

Figures in bold correspond to the models running on the microcontroller.

TABLE VII
PERFORMANCE METRICS FOR EACH MODEL USING THE KFALL DATASET.

Model	Dense	LSTM	GRU	CNN	BiLSTM	BiGRU	CNN DENSE	CNN LSTM	CNN GRU
Accuracy (%)	85.96	74.49	76.22	92.61	86.90	80.16	94.70	79.81	80.77
Sensibility (%)	93.40	98.97	57.52	98.39	80.20	92.88	5.33	96.80	82.92
Specificity (%)	77.57	46.42	97.25	87.18	94.41	65.62	94.18	60.52	78.35
F1-score (%)	87.58	80.57	71.92	92.84	86.62	71.92	94.56	83.60	82.03
Lead time mean (ms)	214.26	611.26	122.42	233.76	121.16	440.52	176.91	593.23	236.34
Memory size microcontroller (kB) FLASH - 512 kB, RAM - 64kB									
FLASH	307.3	495.4	345.0	318.0	454.2	495.2	317.67	489.7	457.9
RAM	31.5								

computer, while the convolution models are 16 and 17 times slower on the microcontroller. The slowest model is BiGRU (515.19 ms), while the fastest model is the DENSE (0.188 ms). It's noteworthy that the execution of the CNN GRU model on the microcontroller encountered failures.

TABLE VIII
ENERGY CONSUMPTION AND INFERENCE TIME OF MODELS DEPLOYED ON THE MICROCONTROLLER (NRF52832). THE AVERAGE POWER CONSUMPTION OF THE MICROCONTROLLER IS MEASURED USING AN OTII (OPEN TEST INSTRUMENTATION INTERFACE) POWER ANALYZER DEVELOPED BY QOITECH.

Model	run on PC (4.30 GHz)		
	Infer. time	CPU cycle	Average consumption
DENSE	0.138 ms	0.594M	x
CNN	0.713 ms	3.066M	x
GRU	2.029 ms	8.724M	x
LSTM	2.245 ms	9.653M	x
BiLSTM	1.876 ms	8.066M	x
BiGRU	14.798 ms	63.632M	x
CNN DENSE	0.723 ms	3.108M	x
CNN GRU	1.168 ms	5.023M	x
CNN LSTM	1.673 ms	7.194M	x
	run on Nordic (3.3 V - 64 MHz)		
DENSE	2.188ms	0.140M	6.50mA
CNN	11.362 ms	0.727M	6.53 mA
GRU	94.143 ms	6.025M	7.50 mA
LSTM	303.145 ms	19.401M	8.50mA
BiLSTM	194.551 ms	12.452M	7.88 mA
BiGRU	515,194 ms	32.972M	7.85 mA
CNN DENSE	12.878 ms	0.824M	6.72 mA
CNN LSTM	28.894 ms	1.850M	7.03 mA
CNN GRU	x	x	x

In summary, among all tested models, the CNN emerges as the top performer for pre-impact fall detection, boasting an average lead time of 176.91 ms and a detection accuracy of 94.70%. This outcome is unsurprising given the inherent strengths of the CNN model in processing intricate and extended temporal sequences. By automatically extracting hierarchical patterns at various scales from temporal sequences, the model adeptly captures both short- and long-term features, facilitated by the utilization of convolution filters of different sizes. Secondly, the architecture employs weight sharing among convolution filters, allowing the learning of valuable features irrespective of their exact position in the temporal sequence. Moreover, the convolution operations incorporate temporal translation invariance, enabling the CNN to identify patterns without dependence on their specific temporal location. This feature is particularly crucial for pre-impact fall detection.

Importantly, falls exhibit variations in terms of their duration and acceleration signal from one instance to another. Notably, CNN models are distinguished by matrix and convolution operations that are parallelized and optimized for ARM architectures. The utilization of convolution effectively reduces the size of tensors as data propagates through the network. The commendable performance of CNN is coupled with a low memory cost (39.352 kB) and minimal electrical consumption (averaging 6.53 mA). By augmenting the CNN model with an increased number of convolution filters and the addition of a Dense layer, we obtain a CNN DENSE model with 94.70% accuracy, 95.33% sensitivity, and 94.18% specificity. The heightened complexity of the model results in an inference time of 12.878 ms and leads to a slight increase in power consumption (6.72 mA compared to 6.53 mA).

The Dense model stands out as the fastest, achieving an inference time reduced to 2.188 ms, and demonstrating a relatively good detection accuracy of 85.96% and sensitivity of 93.40%. However, the specificity is not as favorable, registering at only 77.57%. This could be attributed to the complexity of the temporal features utilized. The underlying cause might be a class imbalance issue, where there is a larger number of fall series compared to that of ADLs. In scenarios with such an imbalance, higher specificity tends to come at the expense of lower sensitivity. To address this concern, training took into account the class imbalance by defining an initial bias for the output layer using the equation: $bias =$

$\log(\textit{fall's number}/\textit{adl's number})$. The reason behind the lower specificity in Dense models is their consideration of each data point independently, making it challenging to correctly identify the temporal and spatial dependencies present in the time series. The favorable inference time is attributed to the utilization of an interconnected linear neuron with the equation $y = Wx + b$ where y is the output, W is the weight matrix, x is the input and b is the bias vector. Additionally, the use of weight matrices is instrumental in conserving memory space and minimizing data transfer between the memory and the microprocessor.

Certain models clearly exhibit underperformance, notably BiLSTM, LSTM, GRU, BiGRU, CNN LSTM, and CNN GRU in terms of pre-impact fall detection accuracy, specificity, and inference time. BiLSTM achieves commendable performance for pre-impact fall detection with an average lead time of 121.16 ms and a detection accuracy of 86.90%. However, due to their capability to store long-term information, BiLSTM models can be sensitive to subtle variations in the data. In the context of fall detection, this sensitivity may result in normal fluctuations or noise being considered, impacting overall performance. Additionally, the inferior performance of BiLSTM comes at the expense of high memory cost (531.664 kB) and elevated electrical consumption (average of 7.88 mA). *Ye and Yu* [75] also reported the good performances of BiLSLM model (Table X) to detect a fall (accuracy 90.47%) using data from an IMU and pressure sensors. But the authors did not mention about the sampling frequency.Among them, GRU and LSTM emerge as the least effective performers, achieving accuracies of 76.22% and 74.49%, respectively. During the conversion of the GRU model to the tflite format, the GRU layer transforms into 10 basic operators, including 2 flex operators. This transformation is a consequence of employing Tensorflow Lite's Flex delegate to optimize the execution of floating-point operations. However, it's worth noting that the Flex delegate is currently incompatible with TFLite Micro for direct integration into a microcontroller. Following quantization, the basic operators are replaced by a "While" operator, as the GRU layer is not directly portable to microcontrollers. This process involves defining the expected input data format for the model (Figure 4). Tensorflow's quantization tool then converts and adapts the required operators for the model. Given that the GRU model entails recurrent calculations performed on each element of an input sequence, the "While" operator efficiently and flexibly handles dynamic variations in sequence lengths.

B. Time complexity analysis

To gain insights into the respective inference times of the models, we conducted an algorithmic analysis on their time complexity. Time complexity quantifies the number of basic operations, such as multiplications and summations, performed by the model, expressed as a function of the input size nn (refer to Table IX for the big notation expressions of time complexity). Analyzing time complexity is a more rigorous approach than measuring inference time for model benchmarking, as the latter is susceptible to various parameters, including the hardware platform, compiler optimization, and the APIs utilized for model implementation.

TABLE IX
EQUATIONS PROVIDING THE TIME COMPLEXITY OF EACH MODEL IN BIG NOTATION

Model	time complexity (big $\mathcal{O}$ notation)
DENSE	$\Theta\left(na_1 + a_4k + \sum_{i=1}^{4-1} a_i a_{i+1}\right)$
CNN	$\mathcal{O}(\sum_{i=1}^{2} f_{i-1}s_i^2 f_i m_i^2) + DC$
LSTM	$\mathcal{O}(\sum_{i=1}^{3} 4h_i(d_i + 3 + h_i) + DC$
GRU	$\mathcal{O}(3h(d + 2 + 3h)) + DC$
BiLSTM	$\mathcal{O}(\sum_{i=1}^{2} 4h_i(d_i + 3 + h_i) + DC$
BiGRU	$\mathcal{O}(\sum_{i=1}^{3} 3h_i(d_i + 2 + 3h_i) + DC$
CNN DENSE	$\mathcal{O}(ns^2 fm^2) + \Theta\left(na_1 + a_1a_2 + a_2k\right)$
CNN LSTM	$\mathcal{O}((\sum_{i=1}^{2} f_{i-1}s_i^2 f_i m_i^2) + 4h(d + 3 + h)) + DC$
CNN GRU	$\mathcal{O}((\sum_{i=1}^{2} f_{i-1}s_i^2 f_i m_i^2) + 3h(d + 2 + 3h)) + DC$

DENSE : n - inputs, a - hidden neurons, k - output neurons,
CNN : f - filters, m - size output feature map, s - size filter
LSTM / GRU : d - entries lenght of sequence, h - hidden cell
Θ - strict bound , $\mathcal{O}$ - upper bound
DC :$\Theta\left(na_1 + a_1k\right)$

According to *He et Sun* [76], the time complexity of convolution layers is estimated by $\mathcal{O}(\sum_{i=1}^{d} f_{i-1}s_i^2 f_i m_i^2)$, where d is the number of convolutional layers, f_i is the number of filters in the i^{th} layer, f_{i-1} is the number of input channels of the ith layer, s_i is the spatial size of the filter, and m_i is the spatial size of the output feature map. From equations given in Table IX, we observe that different parameters and features influence the inference time: in the LSTM and GRU models, the time complexity depends mainly on the number of hidden cells (h) and the length of the input sequences (d). The time complexity of the Dense model depends on the number of hidden neurons (a) and the number of output neurons (k). Hybrid models (CNN DENSE, CNN LSTM and CNN GRU), which combines different architectures, combine different complexity terms for the different parts of the convolution and recurrence architectures. The Dense model has the shortest inference time of all considered models, simply because it is the simplest, restricted to basic operations such as matrix multiplication and direct addition. The integration of TensorFlow Lite for Microcontrollers on the microcontroller improves inference on ARM microcontrollers.

The table in Appendix I (Table XII) details the number of operations executed by the machine. Models incorporating FullyConnected and/or convolution layers are well-integrated, benefiting from their low number of operators and the optimization of matrix multiplication for modern architectures. Additionally, convolution layers use shared filters to identify patterns in input data, reducing the overall number of parameters.

Recursive models such as LSTM, while employing a small number of operators, are computationally intensive. LSTM necessitates significant computational repetition for each time step in the sequence and is not optimized for 32-bit architec-

TABLE X
COMPARISON OF PERFORMANCE METRICS FOR EACH MODEL FROM OPEN LITTERATURE

Model	Dataset	ACC	SEN	SPE	Lead time
CNN DENSE	**Kfall****	**94.70**	**95.33**	**94.18**	**176.91 ms**
FDSNeXt [20]	KFall**	91.87	-	-	-
CNN [20]	KFall**	85.688	-	-	-
LSTM [20]	KFall**	90.121	-	-	-
CNN LSTM [20]	KFall**	84.036	-	-	-
PreFall k.d. [77]	Kfall**	98.05	94.79	98.53	551.3 ms
CNN LSTM [25]	Sisfall* Kfall**	97.52	99.24	-	500 ms

*Sampling frequency note : * 200 Hz , ** 100 Hz*
k.d. : knowledge distillation

tures. GRU, BiGRU, and CNN GRU share a common WHILE operator procedure in their TensorFlow Lite architecture. A closer examination of this procedure reveals that the operator can be decomposed into a more complex architecture based on 35 elementary operators (ADD, SPLIT, CONCATENATION, MUL, or RESHAPE, Figure 5 in Appendix II). Despite compacting the model as explained in section II-B.5, the GRU layer remains suboptimal for microcontrollers. Future work will involve making substantial contributions to TensorFlow operator support to ensure optimal operation on ARM systems.

C. Comparison with the literature

Table X presents different pre-detection and falls detection models as a function of ACC, SEN, SPE and lead time. Compared with studies utilizing data sampled at 100 Hz, our performance stands out with superior accuracy. In a study by *Mekruksavanich et al.* [21] on the Kfall dataset, CNN, LSTM, and CNN LSTM models were tested for fall detection, yielding accuracies of 85.69%, 90.12%, and 84.04%, respectively.

Another study by *Hnoohom et al.* [20] achieved improved detection accuracy (91.87%) by implementing a multi-kernel CNN model, FDSNeXt, on the Kfall dataset. However, at our sampling frequency of 100 Hz, our simple CNN DENSE model outperforms highly complex models such as FDSNeXt, showcasing better accuracy.

Nevertheless, at lower sampling frequencies, multi-kernel CNN models demonstrate superior performance compared to ours. For instance, *Wang et al.* [23] achieved 98.6% accuracy by incorporating pressure sensors (connected insoles) along with IMU data in a CNN model. Similarly, *Choi et al.* [22] utilized acceleration and gyroscopic data (40 features) to attain 99% accuracy with a modified CNN model, along with a lead time of 662 ms.

When examining the Sisfall dataset with a sampling frequency of 200 Hz, CNN LSTM models surpass our work in both accuracy and lead time, as evidenced by studies conducted by [25]. In these investigations, the CNN model is applied to create feature maps from acceleration and gyroscopic data. Furthermore, when combined with a Dense Model (MLP - Multilayer Perceptron), an impressive accuracy of 98.4% is achieved [24].

Despite these models exhibiting high performance compared to our work, there is limited information available on their power consumption. Consequently, there is a need to expand our literature review to encompass fall detection models. Table XI provides an overview of various pre-detection and fall detection models, considering factors such as inference time, power consumption, and the type of hardware used.

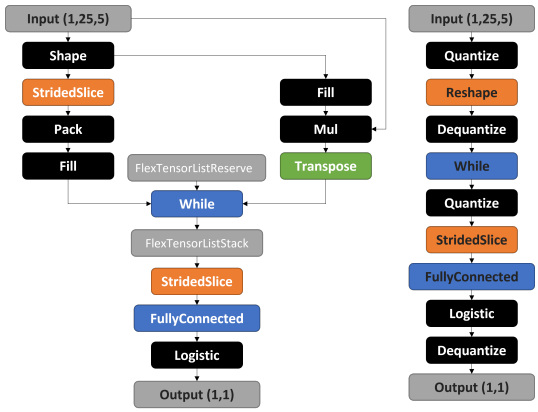


Fig. 4. *GRU model structure (left) before and (right) after quantization*

It's worth noting that the performance of CNN-LSTM in our study is relatively lower compared to its reported use in the literature. Typically, CNN-LSTM architectures consist of three convolutional blocks and two LSTM blocks. For instance, in the work by [4], this architecture achieves an impressive 99.16% accuracy, 99.32% sensitivity, and 99.01% specificity using a 50-step input window with 9 input characteristics. Similarly, [25] reports an accuracy of 97.52% using the same type of model with a sliding window size of 200 steps for 6 input features. But, in our study, we utilized a smaller window of 25 steps with 5 input features. The variation in accuracy may be attributed to differences in the input window size and the number of features compared to the architectures reported in the literature. Additionally, as highlighted in Table X, employing 100 Hz input data (such as Kfall) results in a reduction in the accuracy of the CNN LSTM, likely due to information loss. Conversely, as the sampling frequency increases, so does the accuracy of the model.

The utilization of a Cortex M4 significantly diminishes power consumption in comparison to a microcomputer and an FPGA. This is expected as the latter two must handle more intricate background execution tasks, leading to higher power consumption. *Novac et al.* [74] specifically focuses on the quantization and deployment of deep neural networks on low-power 32-bit microcontrollers. In their work, they used the UCI-HAR dataset for everyday life activity classification, which comprises data acquired at 50 Hz from a smartphone IMU [81]. The ResNet model [82] they employed demonstrated lower energy consumption (4.8 mA) compared to our model, albeit with a longer inference time (119.541 ms) owing to the complexity of the architecture.

In a different context, *Hammerla et al.* [83] utilized BiLSTM for human activity recognition, achieving an accuracy of

TABLE XI
COMPARATIVE LITERATURE ON ENERGY CONSUMPTION METRICS

Model	Inference time (ms)	Average consumption	Setup	Goal	Size (Kb)	Advantages	Disadvantages
Our work CNN DENSE	**12.878**	**22.17 mW (6.72 mA)**	**Nordic + Accelerometer**	**Pre-impact fall detection**	**38.969**	**Portable in MCU, High performance detection on dataset , 3-axis sensor, ultra-low consumption**	**Low Lead time**
PreFallKD [77]	36.8	-	STM32 + IMU	Pre-impact fall detection	59.557	Portable in MCU, High performance detection on dataset	all sensors of IMU (6-axis)
ConvLSTM Lite [4]	2.1	-	Raspberry PI + IMU	Pre-impact fall detection	-	Low inference time, Compact model	No embbeded in MCU, Use reccurent layer
YOLO-LW+SVM [78]	344	300 mW	RISC-V KPU (neuromorphic) + Camera	Fall detection	1800	High performance detection, low consumption	Use a camera for detection
CNN [79]	413	2.5 W	FPGA + Camera	Fall detection	-	CNN model	High inference time, Use a camera for detection
OpenPose-light+SVM [80]	100	15 W	Jetson TX2 + GPU + Camera	Fall Detection	-	High performance detection	Use a camera for detection
ResNet [74]	1034	16 mW (4.8 mA)	STM32 + IMU	ADLs classification	119.541	Portable in MCU, one accelerometer, ultra-low consumption	High inference time, Use MicroAI framework

91.9%, a latency of 470 ms, and a memory footprint of 1.5 MB on an ARM Cortex-M3.

D. Summary and Paths to Improvement

We have demonstrated the implementation of machine learning models on a microcontroller within an integrated circuit. However, the accuracy of pre-impact fall detection is not currently sufficient for commercial product implementation, as false alarms remain an issue (best accuracy 95%). This is a significant barrier to social acceptability, as trust is a key factor in the acceptance and adoption of wearable technologies by the elderly, as established in previous studies [84] [85].

Nevertheless, we anticipate that these limitations will be addressed in the near future with the availability of larger and better datasets for model training. In this study, we worked with 278 time series captured at a frequency of 100 Hz, resulting in a small sliding window size. Additionally, the dataset used here is based on the movements of a young adult and not an elderly person, as it was ethically challenging for us to create datasets of falls and activities of daily living (ADLs) involving elderly individuals for research purposes.

It's worth noting that the Sisfall dataset [86] does exist and comprises a substantial number of simulated activities of daily living (ADLs) in elderly people, with an acquisition frequency of 200 Hz and 908 series. However, the Sisfall dataset contains relatively few data points on falls by the elderly (75 series from one volunteer). Consequently, we anticipate that the results of this study could be modified and improved with the availability of larger and more relevant datasets.

A third limitation identified in this study, which also represents a potential avenue for improvement, is the introduction of a condition based on the variance of the classification window to enhance the accuracy of fall detection models on microcontrollers (MCUs). This approach aims to reduce false alarms while maintaining high performance in detecting forward drops.

Lastly, the size of the microcontroller poses a challenge for the integration of classification models, especially since the developed system was initially intended to work with a threshold algorithm [5]. Addressing these limitations and considering these avenues for improvement will likely contribute to the advancement of fall detection models on microcontrollers.

IV. CONCLUSION

In this paper, we assessed the integration of Deep Learning models into an ultra-low power embedded system designed for pre-impact fall detection. Using Kfall dataset, we found that the CNN DENSE model demonstrated satisfactory performances with accuracy reaching 94.70%, sensitivity 95.33% and specificity 94.18%. The average current consumption is 6.72 mA, equivalent to a power consumption of 22.17 mW, with a minimum inference time of 2.188 ms for the Dense model and a maximum of 515.194 ms for the BiGRU model. These results were compared with a comprehensive review of the state of the art, revealing a lack of data and concern about energy consumption, a critical aspect for nomadic deployment. This study has shed light on the limitations of deep learning for pre-impact fall detection, particularly in terms of energy efficiency, interpretability, and continuous learning capacity. As a promising avenue for future work, spiking neural networks (SNNs) are introduced. SNNs leverage the asynchronous functioning of the human brain, employing discrete pulses to transmit information. This bio-inspired approach offers several potential advantages, including reduced intensive computations, greater adaptability to continuous learning, and improved interpretability of models through the analysis of

neuronal activations and discharge patterns. By delving into this direction, it should be possible to significantly enhance the performance of portable artificial intelligence systems, aligning them more closely with the operating principles of the human brain. A neuromorphic chip will be integrated into the system to design a new pre-impact fall detection model based on spiking neural networks.

APPENDIX I

TABLE XII
COMPARATIVE NUMBER OF OPERATORS MODELS

Model	Nb. of op.
Dense	11
CNN	17
LSTM	8
GRU	32
BiLSTM	12
BiGRU	63
CNN Dense	14
CNN LSTM	14
CNN GRU	50

APPENDIX II

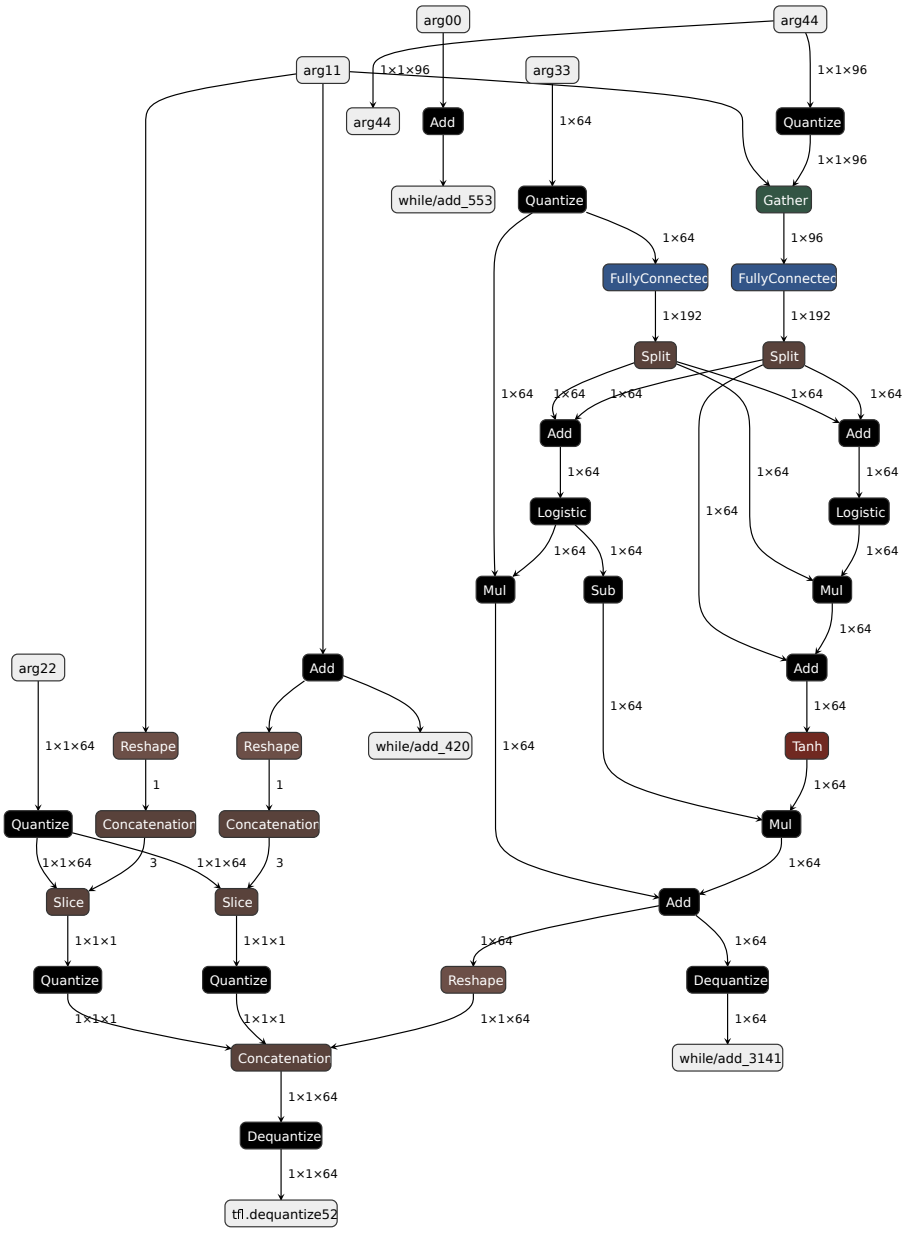


Fig. 5. *Architecture in operator WHILE from CNN GRU model*

REFERENCES

[1] "Falls," Available at https://www.who.int/news-room/fact-sheets/detail/falls (2021/04/26).

[2] T. C. Kolobe, C. Tu, and P. A. Owolawi, "A Review on Fall Detection in Smart Home for Elderly and Disabled People," *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 26, no. 5, pp. 747–757, May 2022.

[3] R. Parmar and S. Trapasiya, "A Comprehensive Survey of Various Approaches on Human Fall Detection for Elderly People," *Wireless Pers Commun*, vol. 126, no. 2, pp. 1679–1703, Jun. 2022.

[4] X. Yu, B. Koo, J. Jang, Y. Kim, and S. Xiong, "A comprehensive comparison of accuracy and practicality of different types of algorithms for pre-impact fall detection using both young and old adults," *Measurement*, vol. 201, p. 111785, Sep. 2022.

[5] F. A. S. Ferreira de Sousa, C. Escriba, E. G. Avina Bravo, V. Brossa, J.-Y. Fourniols, and C. Rossi, "Wearable Pre-Impact Fall Detection System Based on 3D Accelerometer and Subject's Height," *IEEE Sensors Journal*, vol. 22, no. 2, pp. 1738–1745, Jan. 2022.

[6] N. Otanasap, "Pre-impact fall detection based on wearable device using dynamic threshold model," *2016 17th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pp. 362–365, 2016.

[7] Z. Zhong, F. Chen, Q. Zhai, Z. Fu, J. P. Ferreira, Y. Liu, J. Yi, and T. Liu, "A real-time pre-impact fall detection and protection system," *2018 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 1039–1044, 2018.

[8] D. De Venuto and G. Mezzina, "Multisensing architecture for the balance losses during gait via physiologic signals recognition," *IEEE Sensors Journal*, vol. 20, no. 23, pp. 13 959–13 968, 2020.

[9] S. Ahn, D. Choi, J. M. Kim, S. Kim, Y. S. Jeong, M. Jo, and Y. Kim, "Optimization of a pre-impact fall detection algorithm and development of hip protection airbag system," *Sensors and Materials*, 2018.

[10] G. Wu and S. Xue, "Portable preimpact fall detector with inertial sensors," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 16, no. 2, pp. 178–183, 2008.

[11] M. Nyan, F. E. Tay, and E. Murugasu, "A wearable system for pre-impact fall detection," *Journal of Biomechanics*, vol. 41, no. 16, pp. 3475–3481, 2008.

[12] O. Aziz, C. M. Russell, E. J. Park, and S. N. Robinovitch, "The effect of window size and lead time on pre-impact fall detection accuracy using support vector machine analysis of waist mounted inertial sensor data," *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 30–33, 2014.

[13] G. Serpen and R. H. Khan, "Real-time detection of human falls in progress: Machine learning approach," *Procedia Computer Science*, vol. 140, pp. 238–247, 2018.

[14] L. Zhang, Q. Wang, H. Chen, J. Bao, J. Xu, and D. Li, "Ard: Accurate and reliable fall detection with using a single wearable inertial sensor," *Proceedings of the 1st ACM Workshop on Mobile and Wireless Sensing for Smart Healthcare*, p. 13–18, 2022.

[15] J. Xie, K. Guo, Z. Zhou, Y. Yan, and P. Yang, "ART: Adaptive and Real-time Fall Detection Using COTS Smart Watch," *2020 6th International Conference on Big Data Computing and Communications (BIGCOM)*, pp. 33–40, Jul. 2020.

[16] O. K. Botonis, Y. Harari, K. R. Embry, C. K. Mummidisetty, D. Riopelle, M. Giffhorn, M. V. Albert, V. Heike, and A. Jayaraman, "Wearable airbag technology and machine learned models to mitigate falls after stroke," *Journal of NeuroEngineering and Rehabilitation*, vol. 19, no. 1, p. 60, Jun. 2022.

[17] H. Jung, B. Koo, J. Kim, T. Kim, Y. Nam, and Y. Kim, "Enhanced algorithm for the detection of preimpact fall for wearable airbags," *Sensors*, vol. 20, no. 5, 2020.

[18] S. Shan and T. Yuan, "A wearable pre-impact fall detector using feature selection and support vector machine," *IEEE 10th international conference on signal processing proceedings*, pp. 1686–1689, 2010.

[19] I. P. E. S. Putra, J. Brusey, E. Gaura, and R. Vesilo, "An event-triggered machine learning approach for accelerometer-based fall detection," *Sensors*, vol. 18, no. 1, 2018.

[20] N. Hnoohom, S. Mekruksavanich, and A. Jitpattanakul, "Pre-Impact and Impact Fall Detection Based on a Multimodal Sensor Using a Deep Residual Network," *Intelligent Automation and Soft Computing*, vol. 36, no. 3, pp. 3371–3385, 2023.

[21] S. Mekruksavanich, P. Jantawong, N. Hnoohom, and A. Jitpattanakul, "The Effect of Sensor Placement for Accurate Fall Detection based on Deep Learning Model," *2022 Research, Invention, and Innovation Congress: Innovative Electricals and Electronics (RI2C)*, pp. 124–129, Aug. 2022.

[22] A. Choi, T. H. Kim, O. Yuhai, S. Jeong, K. Kim, H. Kim, and J. H. Mun, "Deep Learning-Based Near-Fall Detection Algorithm for Fall Risk Monitoring System Using a Single Inertial Measurement Unit," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 30, pp. 2385–2394, Aug. 2022.

[23] L. Wang, M. Peng, and Q. Zhou, "Pre-Impact Fall Detection Based on Multi-Source CNN Ensemble," *IEEE Sensors Journal*, vol. 20, no. 10, pp. 5442–5451, May 2020.

[24] M. Feng and J. Liu, "A pre-impact fall detection data segmentation method based on multi-channel convolutional neural network and class activation mapping," *Physiol. Meas.*, vol. 43, no. 8, p. 085008, Aug. 2022.

[25] R. Jain and V. B. Semwal, "A novel Feature extraction method for Pre-Impact Fall detection system using Deep learning and wearable sensors," *IEEE Sensors Journal*, pp. 1–1, Oct. 2022.

[26] M. M. Islam, O. Tayan, M. R. Islam, M. S. Islam, S. Nooruddin, M. Nomani Kabir, and M. R. Islam, "Deep learning based systems developed for fall detection: A review," *IEEE Access*, vol. 8, pp. 166 117–166 137, 2020.

[27] E. Torti, A. Fontanella, M. Musci, N. Blago, D. Pau, F. Leporati, and M. Piastra, "Embedded real-time fall detection with deep learning on wearable devices," *2018 21st Euromicro Conference on Digital System Design (DSD)*, pp. 405–412, 2018.

[28] X. Yu, H. Qiu, and S. Xiong, "A novel hybrid deep neural network to predict pre-impact fall for older people based on wearable inertial sensors," *Frontiers in Bioengineering and Biotechnology*, vol. 8, 2020.

[29] A. M. Sabatini, G. Ligorio, A. Mannini, V. Genovese, and L. Pinna, "Prior-to- and post-impact fall detection using inertial and barometric altimeter measurements," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 24, no. 7, pp. 774–783, 2016.

[30] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, "Deep learning for sensor-based activity recognition: A survey," *Pattern Recognition Letters*, vol. 119, pp. 3–11, 2019.

[31] R. Jain, V. B. Semwal, and P. Kaushik, "Deep ensemble learning approach for lower extremity activities recognition using wearable sensors," *Expert Systems*, vol. 39, no. 6, p. e12743, 2022.

[32] T. H. Kim, A. Choi, H. M. Heo, H. Kim, and J. H. Mun, "Acceleration Magnitude at Impact Following Loss of Balance Can Be Estimated Using Deep Learning Model," *Sensors*, vol. 20, no. 21, p. 6126, Oct. 2020.

[33] Y. Kim, H. Jung, B. Koo, J. Kim, T. Kim, and Y. Nam, "Detection of Pre-Impact Falls from Heights Using an Inertial Measurement Unit Sensor," *Sensors*, vol. 20, no. 18, p. 5388, Sep. 2020.

[34] J. Shi, D. Chen, and M. Wang, "Pre-Impact Fall Detection with CNN-Based Class Activation Mapping Method," *Sensors*, vol. 20, no. 17, p. 4750, Aug. 2020.

[35] T. H. Kim, A. Choi, H. M. Heo, H. Kim, and J. H. Mun, "Acceleration magnitude at impact following loss of balance can be estimated using deep learning model," *Sensors*, vol. 20, no. 21, 2020.

[36] B. Koo, X. Yu, S. Lee, S. Yang, D. Kim, S. Xiong, and Y. Kim, "Tinyfallnet: A lightweight pre-impact fall detection model," *Sensors*, vol. 23, no. 20, 2023.

[37] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," *arXiv*, 2018.

[38] P.-E. Novac, G. Boukli Hacene, A. Pegatoquet, B. Miramond, and V. Gripon, "Quantization and deployment of deep neural networks on microcontrollers," *Sensors*, vol. 21, no. 9, 2021.

[39] Y. L. Coelho, F. d. A. S. d. Santos, A. Frizera-Neto, and T. F. Bastos-Filho, "A lightweight framework for human activity recognition on wearable devices," *IEEE Sensors Journal*, vol. 21, no. 21, pp. 24 471–24 481, 2021.

[40] D. Hughes, A. Krauthammer, and N. Correll, "Recognizing social touch gestures using recurrent and convolutional neural networks," *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2315–2321, 2017.

[41] G. Cerutti, R. Prasad, A. Brutti, and E. Farella, "Compact recurrent neural networks for acoustic event detection on low-energy low-complexity platforms," *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 4, pp. 654–664, 2020.

[42] S. Jacob, V. G. Menon, F. Al-Turjman, V. P. G., and L. Mostarda, "Artificial muscle intelligence system with deep learning for post-stroke assistance and rehabilitation," *IEEE Access*, vol. 7, pp. 133 463–133 473, 2019.

[43] C. Xie, F. Daghero, Y. Chen, M. Castellano, L. Gandolfi, A. Calimera, E. Macii, M. Poncino, and D. Jahier Pagliari, "Efficient deep learning models for privacy-preserving people counting on low-resolution infrared arrays," *IEEE Internet of Things Journal*, vol. 10, no. 15, pp. 13 895–13 907, 2023.

[44] P. Sundaravadivel, K. Kesavan, L. Kesavan, S. P. Mohanty, and E. Kougianos, "Smart-log: A deep-learning based automated nutrition monitoring system in the iot," *IEEE Transactions on Consumer Electronics*, vol. 64, no. 3, pp. 390–398, 2018.

[45] Y.-L. Hsu, H.-C. Chang, and Y.-J. Chiu, "Wearable sport activity classification based on deep convolutional neural network," *IEEE Access*, vol. 7, pp. 170 199–170 212, 2019.

[46] A. Albanese, M. Nardello, G. Fiacco, and D. Brunelli, "Tiny machine learning for high accuracy product quality inspection," *IEEE Sensors Journal*, vol. 23, no. 2, pp. 1575–1583, 2023.

[47] P. D. Rosero-Montalvo, P. Tözün, and W. Hernandez, "Time-series forecasting to fill missing data in iot sensor data," *IEEE Sensors Letters*, vol. 7, no. 9, pp. 1–4, 2023.

[48] X. Yu, J. Jang, and S. Xiong, "A large-scale open motion dataset (kfall) and benchmark algorithms for detecting pre-impact fall of the elderly using wearable inertial sensors," *Frontiers in Aging Neuroscience*, vol. 13, 2021.

[49] S. Bhanja and A. Das, "Impact of Data Normalization on Deep Neural Network for Time Series Forecasting," *arXiv*, Jan. 2019, arXiv:1812.05519 [cs, stat].

[50] M. Belkin, D. Hsu, S. Ma, and S. Mandal, "Reconciling modern machine learning practice and the bias-variance trade-off," *Proc. Natl. Acad. Sci. U.S.A.*, vol. 116, no. 32, pp. 15 849–15 854, Aug. 2019.

[51] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, pp. 106–154.2, Jan. 1962.

[52] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybernetics*, vol. 36, no. 4, pp. 193–202, Apr. 1980.

[53] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *Journal of Big Data*, vol. 8, no. 1, pp. 1–74, Dec. 2021.

[54] A. Sanchez-Flores, L. Alvarez, and B. Alorda-Ladaria, "A review of CNN accelerators for embedded systems based on RISC-V," *2022 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, pp. 1–6, Aug. 2022.

[55] G. Van Houdt, C. Mosquera, and G. Nápoles, "A review on the long short-term memory model," *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5929–5955, Dec. 2020.

[56] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, conference Name: Neural Computation.

[57] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *NIPS 2014 Workshop on Deep Learning*, Dec. 2014.

[58] L. Bi, G. Hu, M. M. Raza, Y. Kandel, L. Leandro, and D. Mueller, "A gated recurrent units (gru)-based model for early detection of soybean sudden death syndrome through time-series satellite imagery," *Remote Sensing*, vol. 12, no. 21, 2020.

[59] B. Athiwaratkun and J. W. Stokes, "Malware classification with lstm and gru language models and a character-level cnn," *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2482–2486, 2017.

[60] R. K. Rana, "Gated recurrent unit (gru) for emotion classification from noisy speech," *ArXiv*, vol. abs/1612.07778, 2016.

[61] A. Nfissi, W. Bouachir, N. Bouguila, and B. L. Mishara, "CNN-n-GRU: end-to-end speech emotion recognition from raw waveform signal using CNNs and gated recurrent unit networks," *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 699–702, Dec. 2022.

[62] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[63] H. Robbins and S. Monro, "A Stochastic Approximation Method," *Ann. Math. Statist.*, vol. 22, no. 3, pp. 400–407, Sep. 1951.

[64] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *J. Mach. Learn. Res.*, vol. 12, no. null, pp. 2121–2159, Jul. 2011.

[65] Colin Reckons, "Lecture 6.5 — Rmsprop: normalize the gradient [Neural Networks for Machine Learning]," Available at https://www.youtube.com/watch?v=defQQqkXEfE (2023/02/23), Feb. 2016.

[66] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," *arXiv*, Dec. 2012.

[67] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv*, Jan. 2017.

[68] T. Dozat, "Incorporating Nesterov Momentum into Adam," *Proceedings of the 4th International Conference on Learning Representations*, Feb. 2016.

[69] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," *arXiv*, Jan. 2019.

[70] S. K. Yadav, A. Luthra, K. Tiwari, H. M. Pandey, and S. A. Akbar, "ARFDNet: An efficient activity recognition & fall detection system using latent feature pooling," *Knowledge-Based Systems*, vol. 239, p. 107948, Mar. 2022.

[71] X. Wu, Y. Zheng, C.-H. Chu, L. Cheng, and J. Kim, "Applying deep learning technology for automatic fall detection using mobile sensors," *Biomedical Signal Processing and Control*, vol. 72, p. 103355, Feb. 2022.

[72] M. Musci, D. De Martini, N. Blago, T. Facchinetti, and M. Piastra, "Online Fall Detection Using Recurrent Neural Networks on Smart Wearable Devices," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1276–1289, Jul. 2021.

[73] A. Geron, *Hands-on machine learning with scikit-learn, keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*, 2nd ed. Sebastopol, CA: O'Reilly Media, 2019.

[74] P.-E. Novac, G. Boukli Hacene, A. Pegatoquet, B. Miramond, and V. Gripon, "Quantization and Deployment of Deep Neural Networks on Microcontrollers," *Sensors*, vol. 21, no. 9, p. 2984, Jan. 2021.

[75] B. Ye and L. Yu, "Fall detection system based on BiLSTM neural network," *Transactions on Engineering and Computing Sciences*, vol. 7, no. 5, pp. 01–12, Nov. 2019.

[76] K. He and J. Sun, "Convolutional neural networks at constrained time cost," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5353–5360, Jun. 2015.

[77] T.-H. Chi, K.-C. Liu, C.-Y. Hsieh, Y. Tsao, and C.-T. Chan, "Prefallkd: Pre-impact fall detection via cnn-vit knowledge distillation," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023, pp. 1–5.

[78] B.-S. Lin, T. Yu, C.-W. Peng, C.-H. Lin, H.-K. Hsu, I.-J. Lee, and Z. Zhang, "Fall Detection System With Artificial Intelligence-Based Edge Computing," *IEEE Access*, vol. PP, pp. 1–1, Jan. 2022.

[79] X. Yang, S. Tang, T. guo, K. Huang, and J. Xu, "Design of indoor fall detection system for the elderly based on zynq," *2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, vol. 9, pp. 1174–1178, 2020.

[80] W.-J. Chang, C.-H. Hsu, and L.-B. Chen, "A pose estimation-based fall detection methodology using artificial intelligence edge computing," *IEEE Access*, vol. 9, pp. 129 965–129 976, 2021.

[81] J. Reyes-Ortiz, D. Anguita, A. Ghio, L. Oneto, and X. Parra, "Human Activity Recognition Using Smartphones," UCI Machine Learning Repository, 2012.

[82] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[83] N. Y. Hammerla, S. Halloran, and T. Ploetz, "Deep, convolutional, and recurrent models for human activity recognition using wearables," *CoRR*, vol. abs/1604.08880, 2016.

[84] J. Li, Q. Ma, A. H. Chan, and S. S. Man, "Health monitoring through wearable technologies for older adults: Smart wearables acceptance model," *Applied Ergonomics*, vol. 75, pp. 162–169, Feb. 2019.

[85] S. Malwade, S. S. Abdul, M. Uddin, A. A. Nursetyo, L. Fernandez-Luque, X. K. Zhu, L. Cilliers, C.-P. Wong, P. Bamidis, and Y.-C. J. Li, "Mobile and wearable technologies in healthcare for the ageing population," *Computer Methods and Programs in Biomedicine*, vol. 161, pp. 233–237, Jul. 2018.

[86] A. Sucerquia, J. D. López, and J. F. Vargas-Bonilla, "SisFall: A Fall and Movement Dataset," *Sensors*, vol. 17, no. 1, p. 198, Jan. 2017.

Aurélien BENOIT graduated from the University Institute of Technology Lyon (IUT) in 2017 with a two-year technical degree in Industrial Engineering and Maintenance (GIM), from the University Claude Bernard Lyon 1 (UCBL) in 2020 with a bachelor's degree in Electronics, Electrical Energy and Automation, and from the University Toulouse 3 Paul Sabatier (UT3) in 2022 with a master's degree in Embedded Systems and Microsystems. He is currently preparing a PhD in embedded systems at LAAS-CNRS, University of Toulouse. His research focuses on the development of new gas-generating technologies and materials, and electronic miniaturization to design a next-generation miniaturized portable airbag.

Christophe ESCRIBA received the Ph.D. degree in electrical engineering from the University of Toulouse in 2006. He is carrying out his research with LAAS-CNRS, Toulouse. He is an Associate Professor with the Electrical Engineering Department, National Institute of Applied Sciences Toulouse. His areas of research interests include the electronic architectures of complex systems and healthcare applications.

David GAUCHARD is a software research engineer since 2002 at Laboratory of Analysis and Architecture of System (CNRS-LAAS). He previously got a PhD in telecommunication network simulation. He is now working as user support for various domains, such as software designer and developer for research projects, microcontrolers, and sysadmin targetting HPC infrastructures.

Alain ESTEVE is Research Director at the French National Center for Scientific Research CNRS, currently working at LAAS laboratory. He has contributed to 130 articles in scientific journals. His research focuses on the multiscale modeling of micro and nanotechnologies. He has contributed to numerous work on the modification of surfaces and interfaces of technological interest involving oxides, semiconductors, metal oxides and (bio)-organic molecules. He is currently working on nanoenergetic materials of the thermite type to gain deeper understanding on their structure to properties relationship, and to develop predictive simulation capability to treat their combustion.

Carole ROSSI received the degree in engineering physics and the Ph.D. degree in electrical engineering and physics from the National Institute of Applied Sciences Toulouse, France, in 1994, and 1997, respectively. She is currently a Researcher with the French National Center for Scientific Research (CNRS). She is also working with the Laboratory for Analysis and Architecture of the Systems (LAAS). She is interested in the use of nanomaterials in energy applications for security applications. This includes the use of novel fuel/oxidizer assemblies for advanced pyrotechnical applications as igniters, thrusters or microactuators. She has coauthored more than 190 articles and holds six European patents. She was awarded from the European Council research in advanced grant category in 2019.