# Machine Learning and Cloud Enabled Fall Detection System using Data from Wearable Devices: Deployment and Evaluation

**Italo Araújo[1], Marciel B. Pereira[2], Wendley Silva[3], Igor Linhares[3], Vitor Marx[3], André M. Andrade[2], Rossana M. C. Andrade[1], Miguel F. de Castro[1]**

[1] Departamento de Computação – Universidade Federal do Ceará (UFC)

[2]Campus Crateús - Universidade Federal do Ceará (UFC)

[3]Campus Sobral - Universidade Federal do Ceará (UFC)

***Abstract.*** *In recent years, the popularization of devices to monitor people in combination with Machine Learning (ML) in the context of Internet of Things (IoT) has grown significantly. Then, the number of applications to solve many health issues that require data collection and processing has increased. One of the common concerns by Health institutions is human falls, which can lead to severe health damages or death. Thus, it is crucial to detect quickly when a fall occurs, to reduce the possible sequels. One way to identify potential falls is using data collected from wearable devices as input of an IoT system using ML models, which is the solution proposed in this work using Cloud computing. Thus, we present this solution and its deployment and evaluation that consists of three modules: data acquisition and transfer, intelligent cloud application, and notification service. The best result of the ML models presented is 94.4% of accuracy, considering a low rate of false negatives of 4.3%.*

## 1. Introduction

The Internet of Things (IoT) enables everyday objects to connect to the Internet [Gubbi et al. 2013] providing new services to the users in several domains, such as health applications. These health devices are also known as Internet of Health Things (IoHT) [Rodrigues et al. 2018] objects, with the ability to exchange and process data. These devices are employed to improve patient health [da Costa et al. 2018] by monitoring body sensors of users to health problems and sending alerts to interested people, such as family members or caregivers. In health applications, it is crucial to have a reliable system to avoid the false detection of health risk events and a low delay and response time solution to reduce severe sequels in risk events.

One of the IoHT contributions is the *fall detection*, which are the second leading cause of non-intentional injuries that can also lead to death [WHO 2021]. In this context, one of the goals of IoHT is to prevent risk events related to falls by detecting more quickly when a fall occurs [de Araújo et al. 2018] towards the reduction of possible sequels for monitored users.

According to [Ramachandran and Karuppiah 2020], there are three kinds of architectures used to identify or prevent falls: i) Environmental sensing-based systems, ii)

Vision-based systems, and iii) Wearable sensor-based systems. Any of these architectures is known as a Fall Detection System (FDS).

Analyzing these three possible architectures, environment, and vision-based sensing requires the presence of users; thus, it is not possible to perform monitoring outside sensed location. Vision-based also might present limitations due to privacy issues, i.e., installing cameras in private rooms. Moreover, wearable-based solutions could attain good outcomes compared with environmental-based and video-based FDS. This kind of monitoring can be executed in different environments and circumstances.

Wearable-based FDS are widely explored in the academy and industry, where the solutions can be developed considering different wearable positioning in users' bodies [Wang et al. 2020]. The employment of ML is also widespread but can produce different results depending on chosen models. Thus, it is essential to train and test each ML possible model because results might differ from each other based on data used to generate a new model [Linhares et al. 2020]. Also, the deployment of part of the solution in cloud nodes adds the criterion of response time, which is very important considering the desired performance of FDS.

This paper proposes the development of a Wearable-based FDS that uses ML models, which are deployed in two environments: cloud and edge nodes. Then, the system is composed by mobile applications that collect data from wearables devices and send data to be processed in the Cloud servers and notify the users and the family when a risk is detected. To evaluate the system, we analyzed the metrics of ML models, and inference and response time.

The remaining of this paper is organized as follows. In Section 2, we present the related work found in the literature. Section 3 shows the methodology applied during the execution of our work. Then, we define the proposal of the FDS and the details for obtaining metrics for ML models and architecture in Section 4. In sequence, Section 5 presents the major results and discussion regarding the proposed models and architecture. Finally, we present the main conclusions of this paper in Section 6.

## 2. Related Work

In [Saleh and Jeannès 2019] the authors collect data from a Tri-axial Accelerometer (3DACC) sensor placed in the volunteer's waist. A flow-based algorithm with a sliding time window extracts from data the mean and standard deviation as features, generating a 12 component vector based on $1^{st}$ and $2^{st}$ order moments. The Suport Vector Machine (SVM) model employed in the prediction process reached 99.9% accuracy and 93.05% sensitivity using radial basis kernel.

The authors in [Fáñez et al. 2020] designed a FDS from accelerometer data with calibration for each user in three steps: i) Activity of Daily Living (ADL) data collection for device calibration and train the first classifier; ii) passing the data through the first classifier, which serves as a filter or threshold; and iii) passing the data for a second trained classifier with the public dataset. Two different SVM models were used in first and second classifiers, as well as a two-class Convolutional Neural Networks (CNN). Although the proposed model generated promising results, the system's primary limitation is the dependency of knowledge of users' behavior.

The work of [Linhares et al. 2020] presented learned lessons on the development

of a FDS integrating smartwatches and a mobile application with ML models. They evaluated several ML algorithms considering three dataset configurations: original, original with SMOTE technique, and original with two features. Random Forest (RF) presented the best results in original data with SMOTE technique configuration, reaching an accuracy of 95.13%, a sensibility of 95.13%, and 94.8% of specificity. Also, the authors created a dataset with five volunteers and 13 scenarios, including falls, resulting in 426 samples.

The work of [Sarabia-Jácome et al. 2020] proposed an edge-computing architecture for fall detection with Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) models. The proposed LSTM model presented an accuracy of 98.75%, trained with the SisFall dataset, that contains two 3DACC and one gyroscope. The model input is composed of a 15 sec-length sample.

The authors in [Zurbuchen et al. 2021] proposed Gradient Boosting Decision Tree (GBDT) as one of ML architectures for their FDS. This work has also demonstrated the sample rate of 20Hz was sufficient to produce high accuracy of 98.73% and 98.52% for GBDT and RF., respectively.

## 3. Methodology

The research described in this paper is the result of an industry-academia partnership and followed a methodology divided into three fronts, as shown in Figure 1. The first consisted of the research conception, followed by the data analysis front, ending with developing the solution itself. We started our project with the research front, and the other two were executed in parallel with the research results.

In the research design, we defined the scope of the research since there are alternatives for creating IoHT solutions, such as the use of environmental sensors, cameras, or wearable devices. Thus, we decided that the initial scope would be to use wearable technologies to monitor a patient for fall detection in several environments. Next, we conducted a literature review to discover the latest solutions created for the defined context.

In sequence, we performed two activities in parallel, and they are related to the search for datasets that could be applied in the generation of machine learning models for our context and the specification of FDS architectures to be used in the development front. As for the datasets, we found several, but only three met the requirements of our solution, these being UmaFall [Casilari et al. 2017], UpFall [Martínez-Villaseñor et al. 2019], and [Özdemir and Barshan 2014].

In the data analysis front, we concatenated those previously-established datasets to create a more robust one containing accelerometer and gyroscope data. As the authors collected them differently, we need to make adjustments to standardize to train and test the models. After this, we extracted five features based on sensors data after several tests, which we will explain better in Section 4. In sequence, we analyzed the features to identify those that best fit our context.

The development front starts with two sub-processes running in parallel, one for generating the machine learning models from the results of the data analysis front. The second is related to developing the solution itself, including the mobile applications. Finally, this front is finished with the integration of the results of the two sub-processes:

models and applications.

The model generation sub-process comprises three activities, where the first receives as input the features defined in the previous step. Then, training is performed on the machine learning algorithms, aiming to generate models that meet the best measures for the project context: i) sensitivity and ii) specificity. These measures evaluate the hit rate of the algorithms regarding fall situations and daily activities, respectively. Then, these models are compared with values found in the literature, trying to verify if they present better results. This model generation and evaluation process are cyclical until a satisfactory model is obtained.

The other sub-process aims at developing the FDS, which was defined to be an integration between two applications – one for the monitored user and the other for the caregivers – which is better described in the section 4. This sub-process consists of the software development process of Software Engineering, which begins with the requirements specification – already started in other steps – the software design, based on the architecture previously defined, the development itself, and the tests to ensure that the software is working correctly. We used an incremental process, so it is cyclical until there is stable, ready-to-use software.

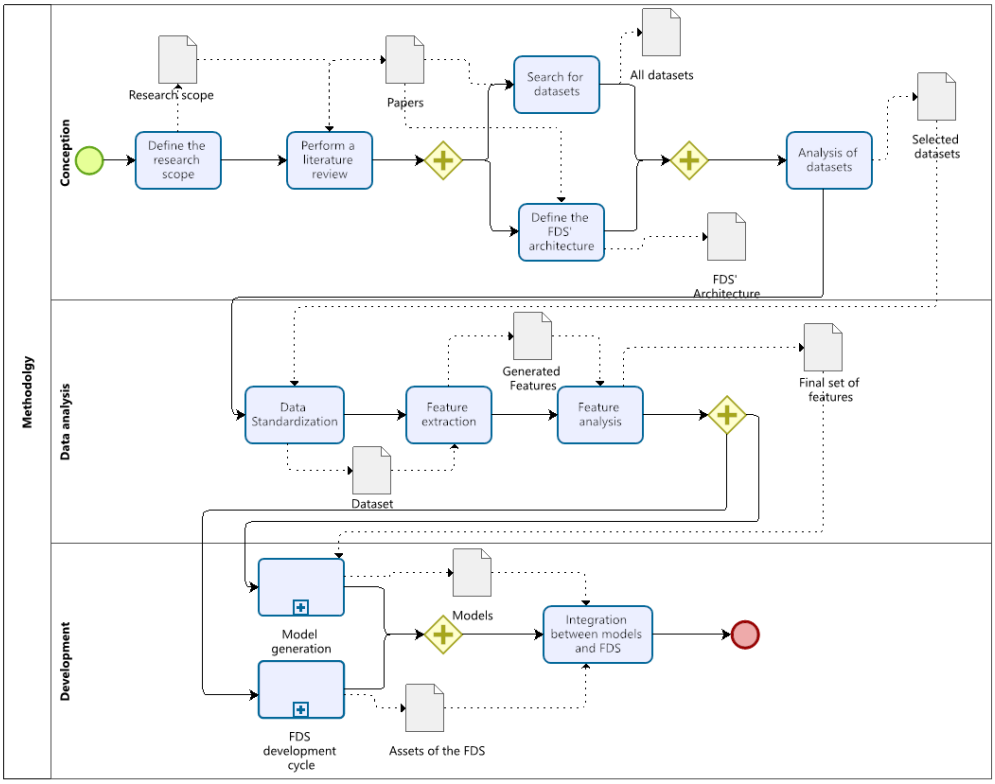


**Figure 1. Methodology of this work**

## 4. Fall Detection System Proposal

We propose the development of a FDS consisting of three layers architecture based on [Hiremath et al. 2014]: i) wearables devices; ii) gateway; and iii) cloud/edge server. In this section, we will address the architecture and the modules of our FDS, especially the ML models generation, which involves the standardization of data, performing training, and tests.

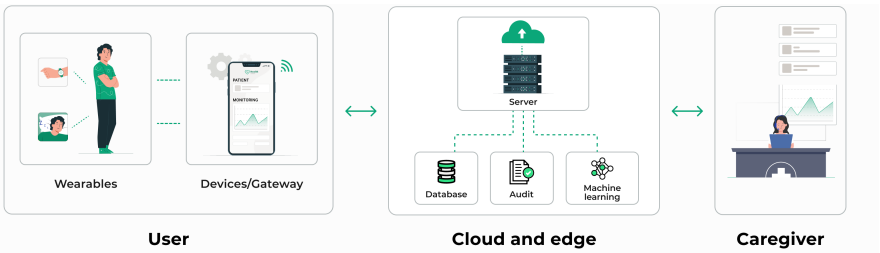


**Figure 2. Layers of our Fall Detection System (FDS) Internet of Things (IoT) solution, adapted from [Hiremath et al. 2014]**

### 4.1. Architecture Summary

The *wearable application* executes on the smartwatch by collecting data from an embedded 3DACC sensor and sending data packages to the *patient mobile application* in the smartphone, which is responsible for sending the collected data to the *Cloud server*. Finally, if the result of inference server present in Cloud is fall detection, the system answers to the monitored patient and the *caregiver mobile application*. We present a detailed description of these modules as follows:

The **Wearable Application** was developed in Android to be executed in devices compatible with WearOS, which is an optimized operating system designed by Google for wearable devices. This module is responsible for gathering data from the accelerometer present in the device and send to the mobile application of the monitored user. The sample rate of collected data must be equal to the frequency used to produce intelligent models to detect falls.

The **Monitored User Mobile Application**, also developed using Android, is used to handle data obtained from wearable over a Bluetooth connection. Therefore, the application constructs a data window with the same size as the produced models in the inference module. We also developed a login system to identify different users with their consecutive caregivers. The login system is crucial to identify the samples of each monitored user and relate these users to their respective caregivers, according to the agreement of monitoring between caregivers and monitored users.

The **Cloud Server** layer contains four components: i) listening; ii) database server; iii) audit server; and iv) ML inference service. The three first modules are responsible for the user's management, data receiving and storage, and notification service using Message Queuing Telemetry Transport (MQTT) protocol to notify the monitored user and the caregiver. The inference service module predicts a fall using the sensor data as input of the pre-trained ML model.

The **Caregiver Mobile Application** is the final layer composed by a mobile application that receives MQTT notifications from Cloud server. By receiving a fall notification, the caregiver application will display a set of possible actions to be taken by the agent. This application is different than the application of monitored users because caregivers can monitor more than one user simultaneously.

### 4.2. Inference Module using Machine Learning

In order to produce the inference module, we adopted a typical pipeline for the development of a ML model: data acquisition from *datasets*, *preprocessing*, feature

extraction, model processing and training, and evaluation, described in the following Subsections.

### 4.2.1. Datasets

We performed research on datasets that present 3DACC data collected from wrist devices, which we initially considered compatible with the signals that we collected from wearable devices. The selected public datasets were [Özdemir and Barshan 2014], [Casilari et al. 2017] (UMAFall), and [Martínez-Villaseñor et al. 2019] (UPFall), described as follows:

**Dataset 1**: [Özdemir and Barshan 2014] dataset was collected from 17 volunteers, where each volunteer performed 20 Falls and 16 ADL with five trials for each activity. It resulted in 3060 samples, of which 1700 are falls—the data collected from the volunteer's right wrist present a sample rate of 25Hz.

**Dataset 2**: [Casilari et al. 2017] UmaFall dataset presents data obtained from 531 simulated activities performed by 17 users, from which 208 represent falls. The sample rate of left wrist 3DACC signals was 20Hz.

**Dataset 3**: In the [Martínez-Villaseñor et al. 2019] UpFall dataset, the wearable 3DACC sensors are placed on the wrists, with a sample rate of approximately 18Hz. The 17 volunteers produced three trials of five different simulated falls and six ADLs. The dataset presented 561 windowed samples, of which 255 are falls.

### 4.2.2. Data preprocessing

To construct a single dataset from the three works described in Subsection 4.2.1, we performed a data preparation by removing malformed samples from the dataset and converting the signal data to the default unity of degrees per second $(deg/s)$. We also resampled the data to the same sample frequency of 25Hz, which is greater than the minimum recommended for 3DACC-based FDS using ML techniques, according to [Liu et al. 2018]. In the Table 1 we show the characteristics of the final dataset we adopted for training ML models. The data format is timestamp, sensor data, sensor type.

**Table 1. Datasets selected in this research.**

| Parameter | Dataset 1 | Dataset 2 | Dataset 3 | Final Dataset |
|---|---|---|---|---|
| Sample Rate (Hz) | 25 | 20 | 18.4 | 25 |
| Number of ADLs | 1700 | 208 | 255 | 2553 |
| Number of Falls | 1360 | 323 | 306 | 2326 |
| Total of Samples | 3060 | 531 | 561 | 4879 |

The Data Version Control (DVC) is a useful resource that allows organizing datasets by name and version, considering the necessity of working with newly acquired data in the future. In this work, we adopted DVC for the final dataset in order to guarantee robustness considering the reproducibility of experiments.

### 4.2.3. Feature Extraction and Selection

This step consists in selecting a data window containing the feature space of original data provided by 3DACC considering many time-domain statistics, which helps extract information that better characterize each activity [Zurbuchen et al. 2021]. The

metrics we chose for extract features were: *mean*, *median*, *minimum*, *maximum*, *kurtosis*, and *skewness*. Our goal is to find a small set of new features to be used as input for ML models. Thus, we measured the performance of each proposed ML technique considering a subset of new features, so the features that produce the best results are also hyperparameters of our model.

We evaluate the new features previously described over all samples in the dataset; thus, we call this step *feature extraction*. The new data will be used as input in the ML models. We use all features in the hyperparameter optimization step. Afterward, we evaluate the models with existing optimized hyperparameters for all combinations of features. Therefore, for each model, we choose features which models attain the best performance.

#### 4.2.4. Model Processing and Training

According to the related work in Section 2, there are a set of common techniques to be adopted in the development of ML models for producing a FDS. Thus, we investigated six ML techniques described as follows:

*Decision Trees (DTs)* are classifiers based on simple decision rules from data features where the top node, known as the root, represents the data, and the bottom node, known as the leaves, represents classes [Kerdegari et al. 2012]. The training process consists of finding thresholds for each tree level according to input data. The common hyperparameters to be optimized are the *maximum depth*, *maximun number of features*, *minimum samples in the leaves* and *minimum samples to split internal nodes*.

*Random Forest (RF)* is an ensemble of DTs classifiers, where each DT classifies the incoming data, and these classifications are accounted as votes to choose the most popular class. The common hyperparameters to be optimized are the same as the DT, in addition to the *number of estimators*. The model inference depends on the majority of estimators' output, and this strategy reduces the variance of estimators compared to the bagging strategy in DT.

*Gradient Boosting Decision Tree (GBDT)* is also an ensemble of DTs that uses lifting algorithm AdaBoost [Ning et al. 2019], the gradient descend and the loss function to optimize the model, depending on the number of classes of input data. Some hyperparameters for GBDT are also the same as the RF, including the *loss function* and its optimization *criterion*.

*Suport Vector Machine (SVM)* is a ML technique that performs a quadratic optimization to find the decision surface over a $k-$dimensional feature space. New data is classified depending on which side it is placed [Kerdegari et al. 2012] For model tunning, the common hyperparameters to be optimized are the *penalty parameter C*, *kernel function* and the *gamma coefficient*.

*Long Short-Term Memory (LSTM)* is an architecture for Recurrent Neural Networks (RNN) that presents units composed of a cell, an input gate, an output gate and a forget gate, which the data flow is controlled. Some hyperparameters to be optimized in this architecture are *activation function*, *dimension of layers*, *optimizer*, *number of epochs* and *batch size*.

*Gated Recurrent Unit (GRU)* is also a RNN-based architecture, similar to LSTM,

but uses the hidden state instead of the cell state to transfer information between units. The learning process is made by tuning the weights of each gate [Dey and Salem 2017]. The common hyperparameters for GRU are similar to LSTM.

We split the data into two groups of train and test with $70\%$ and $30\%$ of samples each, respectively. The training group is also used to perform hyperparameter optimization. After finding the best hyperparameters, we proceed to the train-test step for all ML models. Finally, we store performance metrics of each ML technique for the test set to compare models.

#### 4.2.5. Machine Learning Evaluation Metrics

We evaluated the models of our FDS by adopting three common metrics: *accuracy*, *sensitivity*, and *specificity*, described in Eq. 1, Eq. 2, and Eq. 3, respectively. These metrics depend on the rates for True Positives(TP), True Negatives(TN), False Positives(FP) and False Negatives(FN), as follows:

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$\text{sensitivity} = \frac{TP}{TP + FN} \tag{2}$$

$$\text{specificity} = \frac{TN}{TN + FP} \tag{3}$$

We aim to reach models that produce the highest sensitivity since we want to guarantee the correct prediction of most falls. The second performance parameter for comparing models is the specificity to check the hit rate of daily activities. We also use accuracy to analyze the general results compared to all events.

### 4.3. Cloud Nodes

We deployed the system on the Cloud server and ran the applications in this layer, in which the environment settings will be presented in Section 5. Also, we collect the time performance of Cloud nodes by measuring the latency between data send and the received inference response in the mobile application. The overall response time ($t_r$) is composed by the network delay ($t_n$) between applications and Cloud nodes, and ML inference and data preprocessing time for received data ($t_{ML}$), as shown in Eq. 4.

$$t_r = t_n + t_{ML} \tag{4}$$

## 5. Results and Discussion

In this section, we present the major results provided by ML models for FDS in comparison with the related work we selected for this paper. Also, we discuss the performance evaluation of our solution in the Cloud/Edge environments.

Regarding the hardware requirements for the ML pipeline, we trained and tested our models in a machine with 16 GB of RAM, 512GB of SSD storage, and a 10th generation Intel Core i7 processor. All the experiments were executed in a computer with Ubuntu 20.04 LTS and the models were developed using ML the libraries *scikit-learn*, *PyTorch* and *TensorFlow*, compatible with Python 3.6, under the virtual environment *pipenv*. We adopted DVC to version the ML artifacts – datasets and models.

The Cloud environment must support the modules described in Subsection 4.1. This environment was deployed in HuaweiCloud using a dual-CPU ECS virtual machine, with 8GB of RAM and Ubuntu 18.04 LTS as the operating system, and the ModelArts inference service that executes pre-trained models in Python.

### 5.1. Machine Learning Models' Performance

We performed data training and test according to the procedure described in Subsection 4.2.4 using the ML models: DT, RF, GBDT, SVM, LSTM and GRU. We collected best hyperparameters for each model according to a grid search method. Therefore, we present in Table 2 the best hyperparameters found after optimization process. The LSTM and GBDT models were optimized in 100 epochs using sigmoid activation function and dropout of 5%.

**Table 2. Best hyperparameters for each developed ML model.**

| Model | Hyperparameter | Value | Hyperparameter | Value |
|---|---|---|---|---|
| **DT** | *Max. depth* | 40 | *Max. features* | 0.6 |
| | *Min. samples leaf* | 30 | *Min. samples split* | 0.001 |
| **RF** | *Max. depth* | 11 | *Max. features* | 0.6 |
| | *Min. samples leaf* | 30 | *Min. samples split* | 0.001 |
| | *N. of estimators* | 80 | | |
| **GBDT** | *Max. depth* | 5 | *Loss function* | exp. |
| | *Criterion* | MSE | *Min. samples split* | 0.01 |
| | *N. of estimators* | 100 | | |
| **SVM** | *penalty C* | 5 | *Kernel Function* | poly. |
| | *gamma* | 0.1 | | |
| **LSTM** | *Batch Size* | 100 | *Dim. Layer 1* | 20 |
| | *optimizer* | Adam | *Dim. Layer 2* | 4 |
| **GRU** | *Batch Size* | 100 | *Dim. Layer 1* | 10 |
| | *optimizer* | Adam | *Dim. Layer 2* | 4 |

In sequence, we show the evaluation of new features as explained in Subsection 4.2.3 considering the best set of hyperparameters of models. We evaluated this step over only the train set, which results are shown in Table 3. Considering the new feature subset obtained in the feature extraction step, we performed model training and test considering also the best hyperparameters of Table 2, as described in Subsection 4.2. In Table 4 we show the reached performance for each ML model in their respective test set regarding evaluation metrics shown in Subsection 4.2.5.

**Table 3. Minimum set of new features for each model.**

| Model | Features | Accuracy | Sensibility | Specificity |
|---|---|---|---|---|
| DT | Mean, Std., Skw. | 0.933 | 0.897 | 0.916 |
| RF | Mean, Std., Skw. | 0.993 | 0.962 | 0.978 |
| SVM | Mean, Std., Skw. | 0.976 | 0.975 | 0.975 |
| GBDT | Mean, Std., Kurt. | 0.912 | 0.915 | 0.908 |
| LSTM | Mean, Max., Min., Skw., Kurt. | 0.890 | 0.879 | 0.901 |
| GRU | Median, Max., Min., Skw., Kurt. | 0.879 | 0.886 | 0.873 |

The reached metrics in Table 4 are worse than those presented in Table 3 because of the evaluation with the test set, regarding we used hyperparameters obtained in model training. Regarding the accuracy and sensibility, the GBDT model presented the best performance. We adopted sensibility as an important criterion because we want to reduce as much as possible the rate of False Negativess (FNs). The second-best model, the RF,

**Table 4. Performance metrics for best ML models.**

| Model | DT | RF | GBDT | SVM | LSTM | GRU |
|---|---|---|---|---|---|---|
| Accuracy (%) | 86.3 | 91.6 | **94.4** | 89.5 | 89.7 | 88.1 |
| Sensibility (%) | 87.8 | 93.6 | **95.7** | 90.5 | 89.5 | 88.0 |
| Specificity (%) | 84.8 | 90.0 | **93.0** | 88.5 | 89.8 | 88.3 |

is another well-known algorithm for fall detection according to the literature. Although GBDT and RF were introduced in other FDS, there is no generalization for a group of datasets simultaneously. In Table 5 we summarize the results we obtained in our work compared to the literature.

**Table 5. Comparison of developed FDS with related word.**

| Work | Dataset Size | Model | $acc^{best}$ (%) | $sens^{best}$ (%) | $spec^{best}$ (%) | Proposal |
|---|---|---|---|---|---|---|
| [Saleh and Jeannès 2019] | 4500 | SVM | 99.85 | 99.72 | 99.85 | Only model |
| [Fáñez et al. 2020] | 531 | CNN | 90.23 | 90.9 | 87.26 | Only model |
| [Linhares et al. 2020] | 627 | RF | 95.13 | 94.80 | 95.27 | Model, wearable |
| [Sarabia-Jácome et al. 2020] | 4497 | LSTM | 98.75 | 97.6 | 97.44 | Model, wearable, Edge |
| [Zurbuchen et al. 2021] | 4505 | GBDT | 98.73 | 98.26 | 99.21 | Only model |
| **Our Work** | **4879** | **GBDT** | 94.4 | 95.7 | 93.0 | **Model, wearable, Cloud** |

## 5.2. Cloud Performance

We deployed four different models that presented the best metrics in both proposed environments. The ModelArts service provided an endpoint to handle new incoming data for model inference in the Cloud. In the Edge, we developed a *nginx* HTTP application to process incoming data in the same way as the Cloud.

The response time represents the overall system delay, which depends on the network connection and model inference time, as shown in Eq. 4. Our proposed system must pursue the lowest interval between the event and the response because it could involve a risk of death depending on the practical case. Moreover, the inference time also depends on the complexity of ML models and computing resources demanded by intelligent services. We show in Table 6 the average time performance considering models deployed in Cloud, except LSTM and GRU models, which were incompatible with the intelligent service in Cloud used in this work.

**Table 6. Time performance of tested models in Cloud Environment.**

| Models | Train Time (sec.) | Inference Time (sec.) | Network Time (sec.) | Response Time (sec.) |
|---|---|---|---|---|
| DT | **1.7** | 0.113 | 1.702 | 1.815 |
| RF | 16.9 | 0.118 | 1.741 | 1.859 |
| GBDT | 95.3 | **0.099** | 1.723 | 1.822 |
| SVM | 43.9 | 0.101 | **1.660** | **1.761** |

All models deployed in Cloud presented similar average inference and response time. The GBDT presented the smallest inference time, thus, considering this model presented the best performance through models' metrics, we adopted GBDT as main model for our FDS. The DT has the advantage of a smaller train time, which allows a fast model improvement when working with new samples for training. Also, the number of users that send requests to FDS service impacts the performance of models due to the high demand.

## 6. Conclusion

In this paper, we presented the proposal of a Cloud-based FDS solution, considering an architecture that uses wearable devices and smartphones to monitor users. We successfully employed a combination of three publicly available datasets as training data or our proposed models.

Considering the ML models performance, GBDT presented best sensibility and accuracy. Although our proposed model did not present performance as better as the related work, we evaluate the deployment of ML models in a Cloud environment, in which input data is collected from wearables.

We choose the wearable approach in this work because of the premise that users will be more likely to adopt a wearable solution through their privacy and portability. The system has the potential to become a viable product because it is not necessary for the user to be in a restricted environment to the system work, as well as it depends on only one wearable. As future work, the proposed a set of improvements for our FDS considering new models, datasets, and the inclusion of environmental sensors and cameras.

## Acknowledgements

We would like to thank CNPq for the Productivity Scholarship of Rossana Andrade DT-1 (No 306362/2021-0) and Huawei for supporting this research under the Brazilian Informatics Law (No 10.176 of 1/11/2001) incentives.

## References

[Casilari et al. 2017] Casilari, E., Santoyo-Ramón, J. A., and Cano-García, J. M. (2017). UMAFall: A multisensor dataset for the research on automatic fall detection. *Procedia Computer Science*, 110:32–39.

[da Costa et al. 2018] da Costa, C. A., Pasluosta, C. F., Eskofier, B., da Silva, D. B., and da Rosa Righi, R. (2018). Internet of health things: Toward intelligent vital signs monitoring in hospital wards. *Artificial Intelligence in Medicine*, 89:61–69.

[de Araújo et al. 2018] de Araújo, I. L., Dourado, L., Fernandes, L., d. C. Andrade, R. M., and Aguilar, P. A. C. (2018). An algorithm for fall detection using data from smartwatch. In *2018 13th Annual Conference on System of Systems Engineering (SoSE)*, pages 124–131.

[Dey and Salem 2017] Dey, R. and Salem, F. M. (2017). Gate-variants of gated recurrent unit (GRU) neural networks. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1597–1600.

[Fáñez et al. 2020] Fáñez, M., Villar, J. R., de la Cal, E., González, V. M., Sedano, J., and Khojasteh, S. B. (2020). Mixing user-centered and generalized models for fall detection. *Neurocomputing*.

[Gubbi et al. 2013] Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645 – 1660.

[Hiremath et al. 2014] Hiremath, S., Yang, G., and Mankodiya, K. (2014). Wearable internet of things: Concept, architectural components and promises for

person-centered healthcare. In *2014 4th International Conference on Wireless Mobile Communication and Healthcare (MOBIHEALTH)*, pages 304–307.

[Kerdegari et al. 2012] Kerdegari, H., Samsudin, K., Ramli, A. R., and Mokaram, S. (2012). Evaluation of fall detection classification approaches. In *2012 4th International Conference on Intelligent and Advanced Systems (ICIAS2012)*, volume 1, pages 131–136.

[Linhares et al. 2020] Linhares, I., Andrade, R., Junior, E. C., Almir, P., Oliveira, B., and Aguilar, P. (2020). Lessons learned from the development of mobile applications for fall detection. In *The Ninth International Conference on Global Health Challenges*, pages 18–25.

[Liu et al. 2018] Liu, K.-C., Hsieh, C.-Y., Hsu, S. J.-P., and Chan, C.-T. (2018). Impact of sampling rate on wearable-based fall detection systems based on machine learning models. *IEEE Sensors Journal*, 18(23):9882–9890.

[Martínez-Villaseñor et al. 2019] Martínez-Villaseñor, L., Ponce, H., Brieva, J., Moya-Albor, E., Núñez-Martínez, J., and Peñafort-Asturiano, C. (2019). Up-fall detection dataset: A multimodal approach. *Sensors*, 19(9).

[Ning et al. 2019] Ning, Y., Zhang, S., Nie, X., Li, G., and Zhao, G. (2019). Fall detection algorithm based on gradient boosting decision tree. In *2019 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, pages 1–4.

[Özdemir and Barshan 2014] Özdemir, A. T. and Barshan, B. (2014). Detecting falls with wearable sensors using machine learning techniques. *Sensors*, 14(6):10691–10708.

[Ramachandran and Karuppiah 2020] Ramachandran, A. and Karuppiah, A. (2020). A survey on recent advances in wearable fall detection systems. *BioMed research international*, 2020.

[Rodrigues et al. 2018] Rodrigues, J. J., Segundo, D. B. D. R., Junqueira, H. A., Sabino, M. H., Prince, R. M., Al-Muhtadi, J., and De Albuquerque, V. H. C. (2018). Enabling technologies for the internet of health things. *IEEE Access*, 6:13129–13141.

[Saleh and Jeannès 2019] Saleh, M. and Jeannès, R. L. B. (2019). Elderly fall detection using wearable sensors: A low cost highly accurate algorithm. *IEEE Sensors Journal*, 19(8):3156–3164.

[Sarabia-Jácome et al. 2020] Sarabia-Jácome, D., Usach, R., Palau, C. E., and Esteve, M. (2020). Highly-efficient fog-based deep learning aal fall detection system. *Internet of Things*, 11:100185.

[Wang et al. 2020] Wang, X., Ellul, J., and Azzopardi, G. (2020). Elderly fall detection systems: A literature survey. *Frontiers in Robotics and AI*, 7:71.

[WHO 2021] WHO (2021). Falls - world health organization. http://www.who.int/news-room/fact-sheets/detail/falls. (Acessed on 06/08/2021).

[Zurbuchen et al. 2021] Zurbuchen, N., Wilde, A., and Bruegger, P. (2021). A machine learning multi-class approach for fall detection systems based on wearable sensors with a study on sampling rates selection. *Sensors*, 21(3).