Hindawi
Wireless Communications and Mobile Computing
Volume 2019, Article ID 9507938, 13 pages
https://doi.org/10.1155/2019/9507938

Research Article

A Real-Time Patient Monitoring Framework for Fall Detection

Dharmitha Ajerla, Sazia Mahfuz, and Farhana Zulkernine

School of Computing, Queen's University, Kingston K7L 2N8, Canada

Correspondence should be addressed to Sazia Mahfuz; sazia.mahfuz@queensu.ca

Received 29 March 2019; Accepted 19 August 2019; Published 22 September 2019

Academic Editor: Rüdiger C. Pryss

Fall detection is a major problem in the healthcare department. Elderly people are more prone to fall than others. There are more than 50% of injury-related hospitalizations in people aged over 65. Commercial fall detection devices are expensive and charge a monthly fee for their services. A more affordable and adaptable system is necessary for retirement homes and clinics to build a smart city powered by IoT and artificial intelligence. An effective fall detection system would detect a fall and send an alarm to the appropriate authorities. We propose a framework that uses edge computing where instead of sending data to the cloud, wearable devices send data to a nearby edge device like a laptop or mobile device for real-time analysis. We use cheap wearable sensor devices from MbientLab, an open source streaming engine called Apache Flink for streaming data analytics, and a long short-term memory (LSTM) network model for fall classification. The model is trained using a published dataset called "MobiAct." Using the trained model, we analyse optimal sampling rates, sensor placement, and multistream data correction. Our edge computing framework can perform real-time streaming data analytics to detect falls with an accuracy of 95.8%.

1. Introduction

The past few years have seen a rapid increase in people using wearable devices. The use of connected wearable sensor devices is predicted to increase from 325 million in 2016 to 929 million in 2021 [1]. The wrist watch is one of the most popular forms of wearable devices. Sensors inside the device measure metrics like the steps taken, stairs climbed, sleep, heartbeat, and oxygen levels. Typically, the data from a wearable sensor device are sent to a cloud service for analysis and displayed on the dashboard of a connected mobile device. On the cloud, the data are synchronized with a data management environment, which is typically provided as a service by the manufacturer of the wearable device. A single accelerometer sensor at 200 Hz generates about 2.3 GB of data per day. The more the sensors or monitoring metrics are added, the more the data are generated by the sensors. There is a steady progression towards the era of the Internet of Things (IoT) where many such devices will be connected generating terabytes of data that must be analysed possibly in real time to provide effective decision support. If all these data have to be uploaded to the cloud for analytics, it would result in a wastage of network bandwidth and a decrease in response time.

Wearable sensors can be used with edge computing devices that perform a bulk of data-processing tasks and then send out only a subset of the collected data to the cloud. This approach reduces the data transmission time and use of network bandwidth and enables edge devices to notify authorities faster about important analytical results. For example, data from wearable sensors can be received and analysed at an edge device to monitor patient health status and notify appropriate authorities in the event of a fall which can be critical for elderly patients.

According to the World Health Organization, a fall is defined as an event in which a person comes to rest onto the ground or any lower level [2]. An estimated 646,000 fatal falls occur each year, making it the second leading cause of death because of unintentional injury, after road traffic injuries [2]. Across the world, death rates are highest among adults over 60 years of age. More than 50% of injury-related hospitalizations are seen in people aged over 65 [2]. Consequently, almost 40% of the injury-related deaths are from falls in the elderly population [2]. In Canada, many

retirement homes and long-term care facilities have a very high patient-to-nurse ratio, and falls do not get reported until after some time. Falls also cause hip fractures, another common problem in the elderly population. According to the Health Quality Ontario, the average time taken to treat a low-urgency condition in Kingston General Hospital (KGH) is 3.1 hours [3]. With the rising cost of healthcare and a growing elderly population having chronic diseases, there is an urgent need to shift elderly patient care from the hospital to other patient care facilities such as smart retirement homes. It would facilitate better patient monitoring and ensure the wellbeing of all the Canadians.

In the 1990s, Personal Emergency Response Systems (PERSs) were quite popular. One of the most common PERS devices is a pendant which is worn on the neck in the form of a chain. A person who needs help would press the pendant, and a notification is sent to the caregiver. This eliminates the problem of long waiting time before a person can rise and seek help. However, it is not an optimal solution as Roush et al. [4] pointed out that a fall often causes fatality in elderly people leaving them disoriented or not conscious enough to act logically such as pressing a button for help. There are also devices available nowadays which can detect falls automatically and send an alarm to the caregivers or ambulance services. But they are very expensive and require a subscription to a monthly service. An alternative solution is required for caregiving centres, which can detect a fall and generate a call for help when needed. A significant amount of research has been done on fall detection using sensors like accelerometers and gyroscopes, which are cheap and included in most of the smart mobile devices available today like smartphones and tablets.

We propose an edge computing framework which is deployed in close proximity, i.e., within a maximum range of 100 ft from the wireless sensor devices, and collects and performs preprocessing of the data to only transfer the important data to the cloud. We develop machine learning models to analyse the data and generate notifications in real time to enable calls for assistance. We validate our framework for real-time fall detection use case scenario by analysing the accelerometer data collected from the wearable sensor devices. We are working with medical collaborators to deploy our framework at clinics and retirement homes to monitor patients in real time as a part of Kingston's Smart City initiative.

The rest of the report is organized as follows: Section 2 has the literature review. Section 3 describes the proposed framework. The details about the fall detection model are explained in Section 4. Section 5 presents an overview of our framework. Experiments and results are illustrated in Section 6, and conclusions are drawn in Section 7.

2. Literature Review

Fall monitoring has been an emerging field with new systems being introduced constantly. Several taxonomies on fall monitoring can be found in the research, but most of them are done for monitoring in general, i.e., for systems using video, audio, and ambient sensors. We present a literature review specifically for wearable sensor devices and the systems they use for monitoring as explained below.

2.1. Fall Detection Models. An efficient fall detection system detects a fall by analysing the raw sensor data. Threshold-based monitoring and machine learning-based predictive analytics approaches have been developed based on data collected from accelerometers and gyroscopes, the two most commonly used sensors for fall detection. In threshold-based techniques, a fall is detected when monitored data values exceed predefined threshold values. In contrast, machine learning techniques analyse data and try to learn hidden patterns to classify the data.

Bourke et al. [5] presented a fall detection mechanism where sensors were placed on the thigh and trunk. By analysing the signals from these sensors, upper and lower fall thresholds were determined. If the resultant value exceeded the upper fall threshold at the trunk, a fall was detected. It was able to detect a fall with 100% specificity. But when tested against the real-time data, a lot of false positives were observed. So, Bourke et al. [6] in his next study monitored the patients after fall impact was detected. Though this decreased the number of fall positives, it still had problems with differentiating some fall-like activities. Kangas et al. [7] found similar results in his study where apart from monitoring the impact and posture, they tested the start of fall and the velocity before impact. While the start of a fall was clearly shown in the forward and sideward falls, it was not useful for detecting backward falls which are the major causes of hip fractures. Bourke et al. [8] calculated vertical velocity along with impact and posture for both scripted and unscripted activities which lead to fewer false positives.

He et al. [9] proposed a fall detection and alerting system using a smartphone. A median filter was used to smooth out raw accelerometer values. Features like signal magnitude area, signal magnitude vector, and tilt angle were analysed against the smoothed values. When the features exceeded a certain threshold, a fall was detected. Vo et al. [10] analysed falls with a smartphone in the hand, chest pocket, or pant pocket. Detection of fall was done in three steps: step 1—when the fall took place, step 2—when the person hit the ground, and step 3—when the person returned to normal activity or continued to lie down. The orientation was analysed in between two steps when monitoring data exceeded predefined threshold values. To monitor a subject after fall, the orientation data were analysed for movements in the third step. This mechanism was tested on five young people, which resulted in 85% accuracy.

Abbate et al. [11] proposed a mechanism to use both threshold and machine learning algorithms to detect falls. The system was implemented using a smartphone and a wearable sensor placed at the waist with a sampling frequency of 50 Hz. When a fall was detected using a threshold-based technique, it was sent to a classification model for further analysis. The mechanism also had a notification centre, using which the user could turn off the false alarms. The false-positive data were sent to the classification model

again for training. A two-layer feed-forward neural network model was used for classification where features were generated from the input signals and fed into the model. In the case of continuous data acquisition, the model was able to distinguish between false positives and real falls with a specificity of 100%, but no fall occurred during the data collection. So only specificity could be calculated. The authors concluded by stating that small external sensing units will garner a more positive response because of low intrusiveness instead of forcing subjects to carry smartphones in their pockets.

Yuwono et al. [12] proposed a system to optimize the performance of fall detection using a neural network approach with a minimum amount of information from a triaxial accelerometer. The proposed algorithm constantly checks the magnitude of all the accelerometer values. If it exceeds some predefined threshold value, then the signals from 2.5 s before and after the impact are extracted and normalized. Using the discrete wavelet transformation (DWT), *K*-means seeded regrouping particle swarm optimization (RegPOS) and Gaussian distribution of clustered knowledge (GCK) refer each input signal to the cluster centroid and measure the statistical characteristics. Finally, the data are sent to a multilayer perceptron, augmented radial basis function neural network, and a voting ensemble of both. It is concluded that while the results were very promising, training and testing the data are difficult for this technique.

Khan and Hoey [13] surveyed different machine learning algorithms for fall detection. The authors concluded that recurrent neural networks (RNNs) were very well suited for this problem as the data are usually sequential time series data. Theodoridis et al. [14] built long short-term memory (LSTM) recurrent neural network models using a published dataset called the UR fall detection. When compared with support vector machine (SVM) and Bourke et al.'s [5] threshold-based approach, the LSTM model gave better results than the other approaches.

2.2. Fall Prevention Techniques. Besides fall detection, there is a dire need for fall prevention systems. External prevention can be done by installing handrails and teaching techniques to avoid falling, but internal fall prevention is trickier. It requires deep research on the neurological activities to analyse which part of the brain depletes their awareness and lowers the reaction time. Also, the gait and posture can be analysed to correct their balance in case of free fall. Cheng et al. [15] proposed that, by educating the elderly people on fall, teaching different types of exercises, and assessing and monitoring hazard situations, falls can be reduced. Some work has also been done on cushioning the fall so that any injuries can be avoided. Tamura et al. [16] built an innovative system where a wearable airbag inflates automatically when the system predicts a fall 300 ms before the fall occurs. Zhong et al. [17] conducted similar research on real-time falls. The airbag was able to inflate correctly for each fall with a sensitivity of 93.6%. Although systems such as these show great progress towards fall prevention, challenges still exist in handling situations like sideward and forward falls. More work is needed for expediting the inflation process of the airbag and avoiding false alarms.

2.3. Fall Detection Systems. Fall detection systems vary in the components they use. One of the most common sources for the sensor data is a smartphone. The data are collected, analysed, and stored on the smartphone itself. Most of the research studies we have discussed so far are performed through such a device. Because of their limited computational power and storage, advanced machine learning techniques are difficult to implement on smartphones.

2.3.1. Cloud Computing. In fall detection systems, cloud computing is used to retrieve, preprocess, and analyse data remotely in a cloud, which provides a huge amount of computational power and storage. A mobile phone or a wearable device as intermediary devices can be used to send data to the cloud. The data stored in the cloud can be used for long-term analysis. Lai et al. [18] proposed such a cloud computing model for fall detection where the data are analysed using a MapReduce framework on the cloud. The analysed result is sent back to a mobile device to inform the user. The model was able to detect falls with an accuracy of 85% in the case of activities like running.

2.3.2. Edge and Fog Computing. Edge or fog computing applies the concept of cloud computing near the data source. In edge computing, data are retrieved, preprocessed, and analysed at the edge of the network enabling real-time decisions, and only the summary of the data to be uploaded to the cloud. Usually, the terms edge and fog are used interchangeably. Yacchirema et al. [19] used a decision tree-based big data model where if a fall was detected at the fog level, data were sent to the cloud for further analysis. In this case, fog represented a Raspberry Pi board which did the near node analytics. The data were collected at a frequency of 200 Hz. The decision tree model was trained using an existing historic dataset. The system performed with an accuracy of 91.67% and a precision of 0.937.

3. Proposed Framework

The advancement in computational power and big data processing has led to the modern-day data analytics. There has been a great progress in the use of wearable devices, resulting in research on the acquisition and analysis of data from the IoT or connected devices. In this paper, we present an edge computing framework for real-time fall detection. Smartphones are widely used devices for fall detection. However, He et al. [9] suggested a smartphone placed in the pocket causes too much noise. In many research, sensors are placed at the waist. Smartphones are prone to damage in the case of a fall and are also costly to be replaced. So, a cheaper alternative would be very much preferred. We used

cheap sensors called MetaMotionR from MbientLab. The data from the wearable devices were collected at an edge device like a laptop which preprocessed the data using Apache Flink and sent it to the next level of analytics when the magnitude exceeded a predefined threshold. At this level, the data were analysed using TensorFlow where a pretrained LSTM model was used to classify the data as fall or nonfall. We used a public dataset called MobiAct published by Vavoulas et al. [20] to pretrain our LSTM model. Figure 1 shows the general architecture of sensor data-processing frameworks.

4. Fall Detection Using Machine Learning

We developed different machine learning models based on a dataset called "MobiAct" published by Vavoulas et al. [20] to detect fall as explained in our previous work [21]. The dataset contains activities of daily life and four kinds of falls. Since we were only interested in fall detection, we used a subset of the data containing only two kinds of daily activities and four kinds of falls. The daily activities denote two nonfall classes: standing and lying. The four kinds of falls are as follows:

(1) FOL: forward lying (fall forward from standing and use of hands to dampen fall)

(2) FKL: front knees lying (fall forward from standing: first impact on knees)

(3) SDL: sideward lying (fall sideward from standing: bending legs)

(4) BSC: backsitting chair (fall backward while trying to sit on a chair)

We preprocessed both MobiAct and Mbient sensor data using the following operations as listed below:

(1) Scaling: the model trained using the MobiAct dataset was used to classify real-time streaming data. So, to maintain compatibility between the two data sources, MobiAct dataset and the collected real-time fall data using the MbientLab, sensors were rescaled to the range of (+1 g, −1 g) before extracting the features.

(2) Feature extraction: similar to Vavoulas et al. [22] and Kwapisz et al. [23], we generated two sets of features, combined them, and then extracted the optimal feature set from them.

(3) Normalization of feature values: we normalized the extracted features using min-max normalization.

(4) Balancing the dataset: for the MobiAct dataset, the different categories of fall data were merged into a single class of fall data while the two categories of nonfall data were merged into a single class of nonfall data. Then, the fall data were oversampled, and the nonfall data were undersampled to create a combined balanced dataset.

(5) Feature selection: ReliefF algorithm is one of the filter-based feature selection techniques which achieves superior performance in many applications [24]. So, for the MobiAct dataset, the ReliefF algorithm was used to select the top ten most important features from the combined feature set.

We conducted six different experiments using the above dataset, as explained in more detail in Ajerla et al.'s study [21]. We developed multiple machine learning models such as the multilayer perceptron (MLP), LSTM, support vector machine (SVM), weighted K-nearest neighbours, and bagging and boosting trees. When we ignored the one-second delays in detecting falls, the LSTM model performed the best with an accuracy of 99%. We used 58 features and classified the data into a binary fall and nonfall instead of one of the four fall types as listed above. We incorporated this LSTM model in our edge computing framework which is described in the next section. Different configurations were analysed to obtain the optimal parameters of the model.

5. Edge Computing Framework

We present an edge computing framework for fall detection. Real-time data from wearable sensor devices are retrieved by an edge device and sent to Apache Flink for preprocessing and then to TensorFlow for advanced analysis. The system applies our pretrained fall detection models to real-time streaming data collected from wearable sensors using an edge computing platform, as shown in Figure 2. Our system can be used by the healthcare professionals at the clinics, hospitals, and retirement homes and can be extended with other machine learning models to monitor activity, sleep, or tracking dementia patients in real time.

6. Overview of the Framework

We developed a real-time fall detection system which consists of the following components:

(1) Sensors: multiple sensors can be used to retrieve data from the patients.

(2) Edge device: a node at the edge computing platform acts in the pull mode. It retrieves data from the sensors and enables basic data preprocessing and analytics in order to reduce the size of the data to be processed at the next phase and/or uploaded to a remote server or on the cloud.

(3) Streaming data-processing engine: a streaming data-processing engine at the edge computing node retrieves the data and performs real-time analysis such as filtering, noise removal, feature extraction, and summarization as necessary.

(4) Data store: cleaned and preprocessed data coming out of the stream engine are stored temporarily on the edge platform.

(5) Data analytics engine: further analysis of the data is enabled by this analytics engine where pretrained machine learning models can be deployed on the edge platform to classify the data.

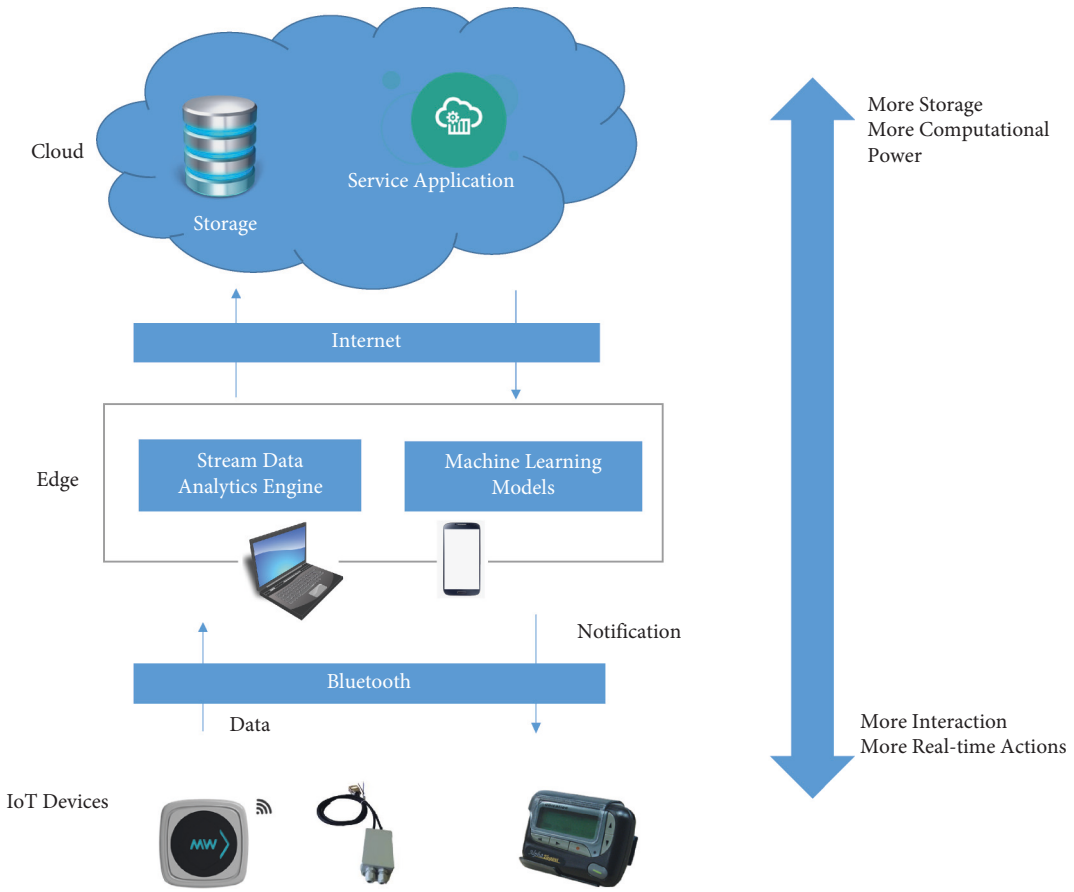

Figure 1: Pipeline for streaming and processing wearable sensor data.

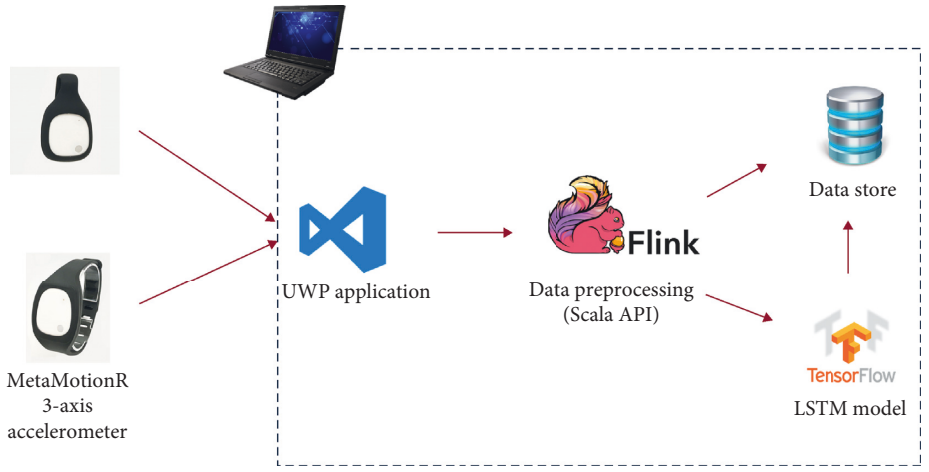

Figure 2: An edge computing framework for detecting falls.

7. Implementation Details

We selected the following for the five components as listed in Section 6 to implement our framework: MbientLab sensors as the wearable device, laptop as an edge device, Apache Flink as the stream-processing engine, local file system as the data store, and TensorFlow as the data analytics engine. The justification behind the selection of the devices and software is given below. Our framework receives data from the sensor devices at the edge computing platform, preprocesses the data using the streaming data-processing engine in real time, and stores the data as comma-separated values (CSVs) files.

7.1. Wearable Device. Nearly 200 million wearable devices were forecasted to be shipped in 2019 alone [25]. A huge variety of wearable sensor devices are available today. Table 1 shows some of the affordable options. All the details were

Table 1: List of wearable sensor devices.

Name	Cost (CAD)	Sensor position	Upload data	Waterproof	Battery
Misfit flash	29.99	Custom	Cloud	Yes	6 months
Mi Band 1S	26	Watch	Bluetooth	Yes	10 days
Up move	19	Clip	Bluetooth	Yes	6 months
Vivofit 3	48.95	Watch	Bluetooth	Yes	1 year
MetaMotionR	91.99	Custom	Bluetooth	Yes	7 days
Arespark	28.13	Watch	Bluetooth	Yes	7 days
Teslasz	16.99	Watch	Bluetooth	Yes	1 day
Fanmis	21.69	Watch	Bluetooth	No	6 days

collected from the retail sites of the device manufacturers. Most of the devices only store the data by the minute or second like Garmin, and the battery life is very low for some of the devices like Teslasz.

We selected MetaMotionR from MbientLab as shown in Figure 3 because of its developmental support, battery life, price (US $91.99), and the availability of raw sensor data. The device allows 10 axes of motion sensing (3-axis accelerometer + 3-axis gyroscope + 3-axis magnetometer + altimeter/barometer). The board is powered by a rechargeable 100 mAh LiPo battery or replaceable CR2032 coin cell battery. It also has a LED, GPIOS, and a push button switch on the board. The board is based on the nRF52 SOC from Nordic built around an ARM Cortex M4F CPU and a Bluetooth low energy 4.2 with 2.4 GHz. A maximum of 8 MB of sensor data can be stored in the off-board memory and downloaded onto a personal device any time in the form of a CSV file or as an array of JSON objects. The weight of the device is 0.2 Oz, and it is water resistant. For our use case scenario of fall detection, we used only the 3-axis accelerometer data. The 16 bit accelerometer in the device generates values ranging from (−2 g, +2 g) to (−16 g, +16 g) with a noise density of 180 $\mu g/\sqrt{Hz}g/Hz$. The Bluetooth connectivity of the device ranges up to 100 ft. It can log data at the local device storage at a rate of 1–400 Hz and stream data to a connected device at a rate of 1–200 Hz [26].

Figure 3: MetaMotionR sensors from MbientLab.

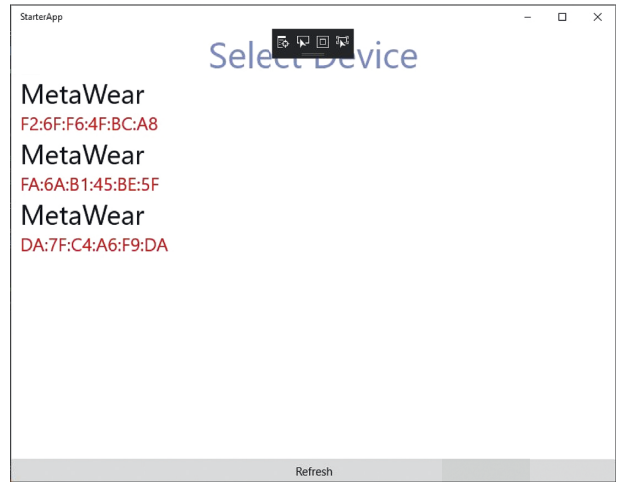

Figure 4: Universal windows platform application for connecting to a wearable device.

7.2. Edge Device. An edge computing mechanism allows data to be collected and processed at a nearby device to reduce data transfer time and bandwidth of the network so that real-time decisions can be taken. Laptops and mobile devices such as cellular phones are commonly used as edge devices.

In our framework, we used a laptop as the edge device as it is a nonobstructive and mobile device and has good computing power to run the neural network-based analytics. MbientLab provides software development kit (SDK) to develop applications on the edge device so that the sensor data can be collected at the desired rate. We developed a Universal Windows Platform application (UWP app) in C-sharp using the template provided by MbientLab to stream raw X-, Y-, and Z-axis accelerometer data to a streaming data-processing engine running on the edge device. Using the SDK, the range of the accelerometer was set to (−2 g, +2 g). The UWP application can be deployed on any machine with Windows operating system (OS). Figure 4 shows the screenshot of a universal windows application displaying all the Bluetooth compatible devices in the vicinity. When the desired device is clicked, the application initializes the wearable device and its configuration. After completing the initialization, the user has an option to start

or stop the accelerometer. When the start button is clicked, a socket connection is opened and the application starts sending raw accelerometer data.

7.3. Streaming Data-Processing Engine. An ideal big data-processing platform for real-time analysis should be open source and able to analyse data in real time. Some of the platforms that satisfy this criterion are Spark, Apex, StreamAnalytix, and Flink. Among the available big data platforms, we chose Apache Flink because of its ability to perform real-time stateful computations on bounded and unbounded data. It can perform computations on different types of data in memory at scale. Flink can process multiple trillions of events and multiple terabytes of data each day [27]. It is also easy to deploy as all communications happen through REST calls. It can be run 24/7 without any inconsistencies which is great for streaming applications. Flink provides different features to maintain consistency like checkpoints, end-to-end exactly once data communication, and high availability setup [27]. It provides three levels of abstraction through its application-processing interfaces (API) process function level, data stream level, and SQL/table level.

We used Scala API at the data stream level to preprocess the data using Apache Flink. From the previous stage, our UWP application initiates the data transfer from the wearable device directly to Apache Flink. We developed an application using Scala API to process, scale, and transform the data and remove noise before handing it over to the next component.

7.4. Data Store. The preprocessed data can either be stored on the edge device or sent to the cloud for storage and further processing. Different data storage systems can be used depending on the data type, flow rate, organization, and query requirements.

We used the local file system to store the data in the form of CSV files. A single wearable device generates up to 2.3 GB of data per day. In this study, we used a laptop as the edge device, and hence we performed the next level of processing on the same device to generate faster notifications. For future, the data can be sent to a cloud server for online training of the machine learning models.

7.5. Data Analytics Engine. The higher-level analytics and classification are done using the analytics engine. This can include training machine learning models, performing an analytic query on the stored preprocessed data, or using the data in decision support systems.

We used TensorFlow as the data analytic and machine learning component to classify the data as fall or nonfall for our use case scenario of fall detection. TensorFlow is an open source library which offers customizable training and layers for the machine learning models. It is one of the libraries that offers advanced machine learning techniques. We used TensorFlow as the data analytic component to build, train, and apply an LSTM neural network model to detect a fall from the human activity data ingested and processed by our framework. For this study, we developed and trained an LSTM model with an offline open source MobiAct dataset and then deployed the model in TensorFlow on the edge computing framework to score the real-time data as fall or nonfall.

The LSTM network was built in Python using Keras with TensorFlow in the backend. It uses softmax activation function, Adam optimizer, mean squared error, and categorical cross entropy as the loss functions for binary and multiclass models, respectively. Each experiment was conducted using 58 feature values at the input layer and 30 LSTM units in the hidden layer. The number of expected outcome dictated the number of nodes at the output layer. Figure 5 depicts the performance of the system against the number of nodes in the LSTM layer. From Figure 5, we can observe that, at 30 nodes, the network gave the best results, and as the number of nodes increased further, performance degraded due to overfitting. Additionally, a dropout of 0.30 was used to avoid overtraining of the models. Each activity in the given dataset was recorded for 10 s. So, the data were fed into the LSTM accordingly; i.e., we used a time step of 10 for all the models. All the output labels were one-hot encoded. One-hot encoding converts the categorical data to a binary sequence for training purposes.

We performed different experiments using multiple machine learning models and selected the best one for scoring in our data-processing pipeline.

8. Integration and Data Flow

The wearable and edge devices in our proposed framework were connected via Bluetooth LE and sockets. The data flow in the framework works as follows: our UWP application on the edge device uses a Bluetooth connection to first connect to the wearable device and open a socket connection between the wearable device and Apache Flink. Then, the UWP application activates the socket connection and starts streaming data from the wearable device directly to Apache Flink. Next, Flink starts preprocessing the data streams by performing data parsing and magnitude calculation. When the magnitudes of axes exceed a predefined threshold value, a 10 s window of data is sent to TensorFlow via sockets.

The 10 s window was used because our LSTM fall detection model was trained using a temporal dataset of 10 s spans. From our previous work, among the different machine learning models that we had developed, the LSTM model proved to be the most accurate in classifying fall and nonfall. Therefore, we deployed this model in TensorFlow. TensorFlow extracts features from the raw data and feeds the same into the previously trained LSTM model. The model then classifies the data representing an activity as a fall or nonfall.

9. Experimentation and Validation

We conducted multiple experiments as illustrated in this section to validate the performance and overall functionality of our edge computing framework. We used human subjects

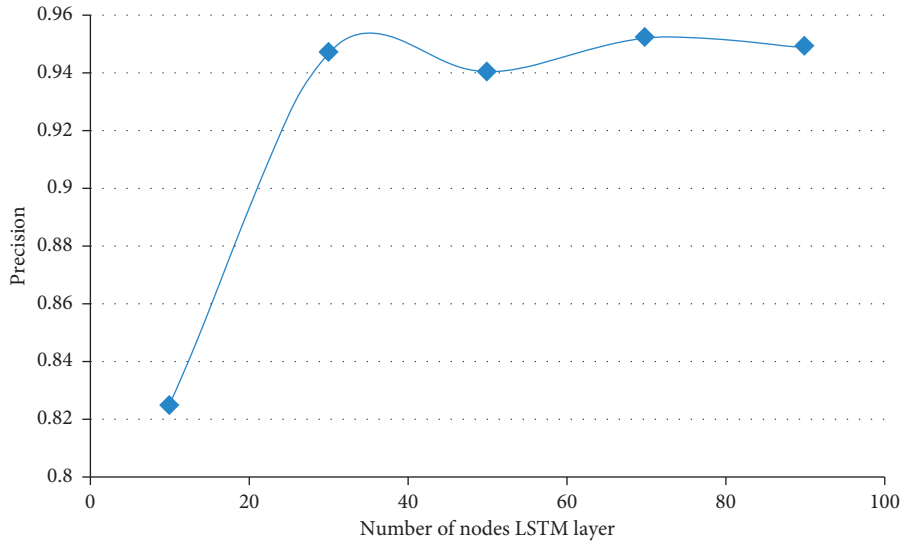

Figure 5: Performance (precision) of the model against the number of nodes in the hidden LSTM layer.

to collect real-time activity and fall data and fed the data through our framework to identify a fall. The data were collected using an accelerometer present in the MbientLab device, MetaMotionR. The subjects were explained about the four types of falls. A yoga mat was provided to avoid any unnecessary injuries. The details about the experimental subjects are given in Table 2. We generated a final dataset containing a total of 64 falls which were annotated by observing the magnitude of the accelerometer data. Our framework classified the streaming data into fall and nonfall activities. To validate the performance of the data-processing pipeline and select the optimal values of data collection frequency and sensor position for both single and multi-sensor monitoring, we conducted the following experiments.

Experiment 1. Different sampling rates with the same sensor position: four types of fall data as listed in Section 3 were collected at frequencies 12.5 Hz, 25 Hz, 50 Hz, 100 Hz, and 200 Hz. The sensor device was placed at the waist in all the experiments.

Experiment 2. Varying sensor position with static sampling frequency: four types of fall data were collected at the best frequency as learned from Experiment 1 by placing the device at different parts of the body, namely, the middle of right calf, left side of the chest, the right wrist, the side waist, and the middle of the thigh. Figure 6 shows the different sensor positions on the body.

Experiment 3. Two wearable devices with the static sampling frequency and sensor position: the optimal sensor positions and frequencies were selected based on the results of experiments 1 and 2. Four types of fall data were collected.

Figure 7 shows the performance of the LSTM model across different sampling rates. The sampling rate of 50 Hz performed the best among all the other frequencies. From our analysis, the frequencies below 50 Hz were not able to capture all the falls, while the frequencies above it performed with similar accuracy. By using a reduced frequency of 50 Hz instead of 200 Hz, we reduced the amount of data that needed to be stored per day from 2.3 GB to 0.31 GB. Nevertheless, the data reduction also had a disadvantage of losing information, so a combination system as suggested by Ren et al. [28] may be better. They introduced an energy efficient mechanism where the data were usually collected at 50 Hz and changed to 200 Hz when a possible fall was detected.

Table 2: Details of the experimental subjects.

Experiment	Subjects	Age	Height	Weight	Falls
1	1	22	168	65	20
2	1	22	168	65	20
3	6	22–40	150–171	55–73	24

Figure 8 shows the performance of the model when the sensor is placed at different parts of the body. It can be inferred from the figure that a sensor placed at the wrist position gives the highest accuracy with the waist and calf following closely. The waist was found to be the steadiest part of the body with the least noise in the data. The thigh had a higher rate of noise in the data. We took the first three best performances and formed a sensor combination of the waist + leg (calf) and waist + wrist. Data were collected from 3 subjects wearing the first combination of sensors and 3 others wearing the second combination of sensors. Figures 9 and 10 show the magnitude of falls when the waist + leg (calf) and waist + wrist combinations are used, respectively. It can be observed that, in almost all the cases, the leg observed the fall before the waist. Such information could be useful for fall prevention techniques in the future. While the wrist mostly detected a fall at the same time as the waist, it gave a better distinction between falls and other activities when combined with the waist.

The performance of the sensor combination models is shown in Table 3. Since the model was trained using the MobiAct dataset, we observe that the model works great

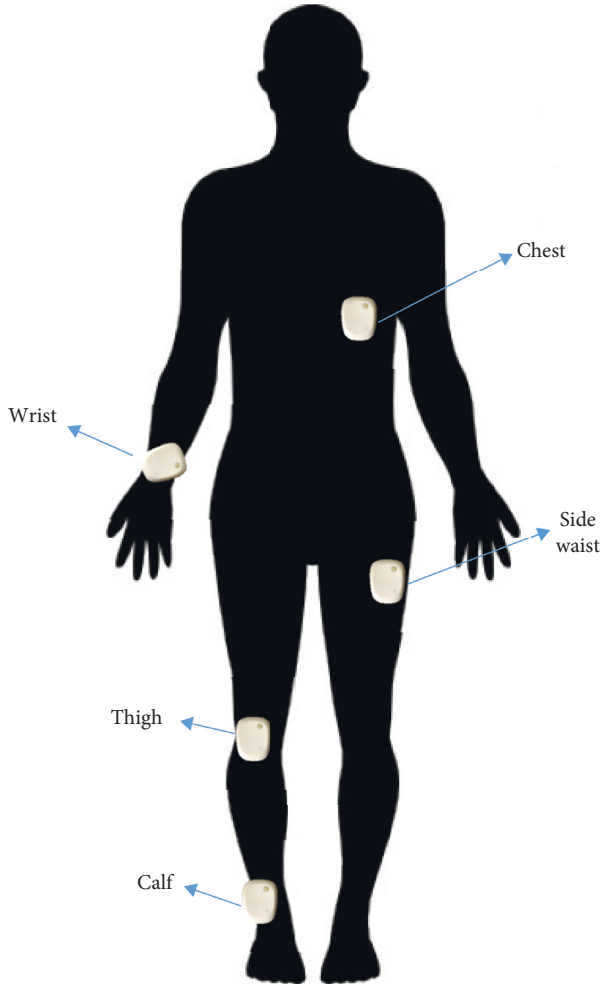

Figure 6: The sensor positions at different parts of the body.

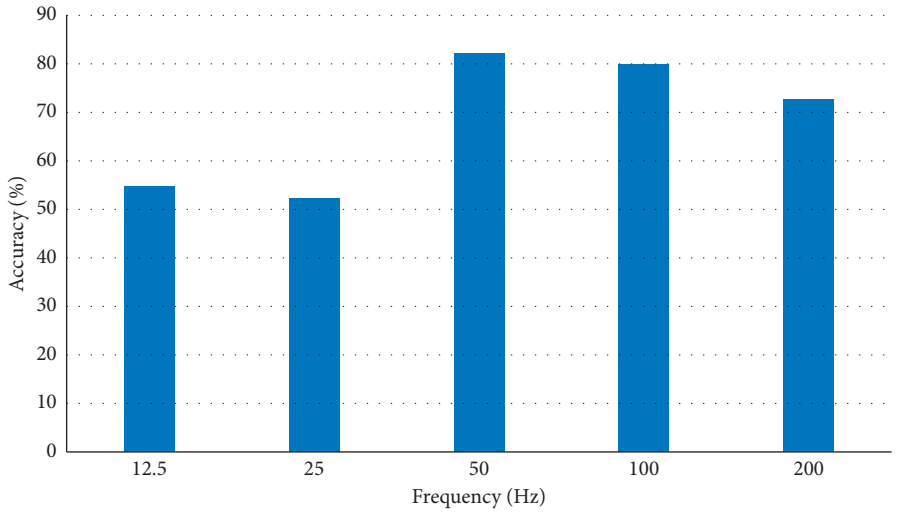

Figure 7: The accuracy of the model at different sampling rates.

when tested on the MobiAct data. For real-time fall detection with a sensor placed at the waist at 50 Hz, the model was able to detect fall with a 0.75 precision, 0.64 recall, and 85.2% accuracy. In the case of multistream data, both the waist + leg and waist + wrist were able to detect a fall with approximately similar precision and recall ignoring an one-second

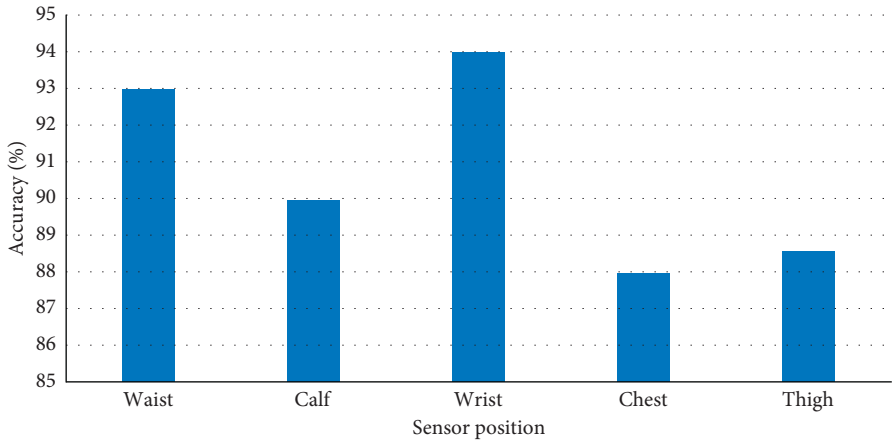

Figure 8: The accuracy of the model against the sensor position.

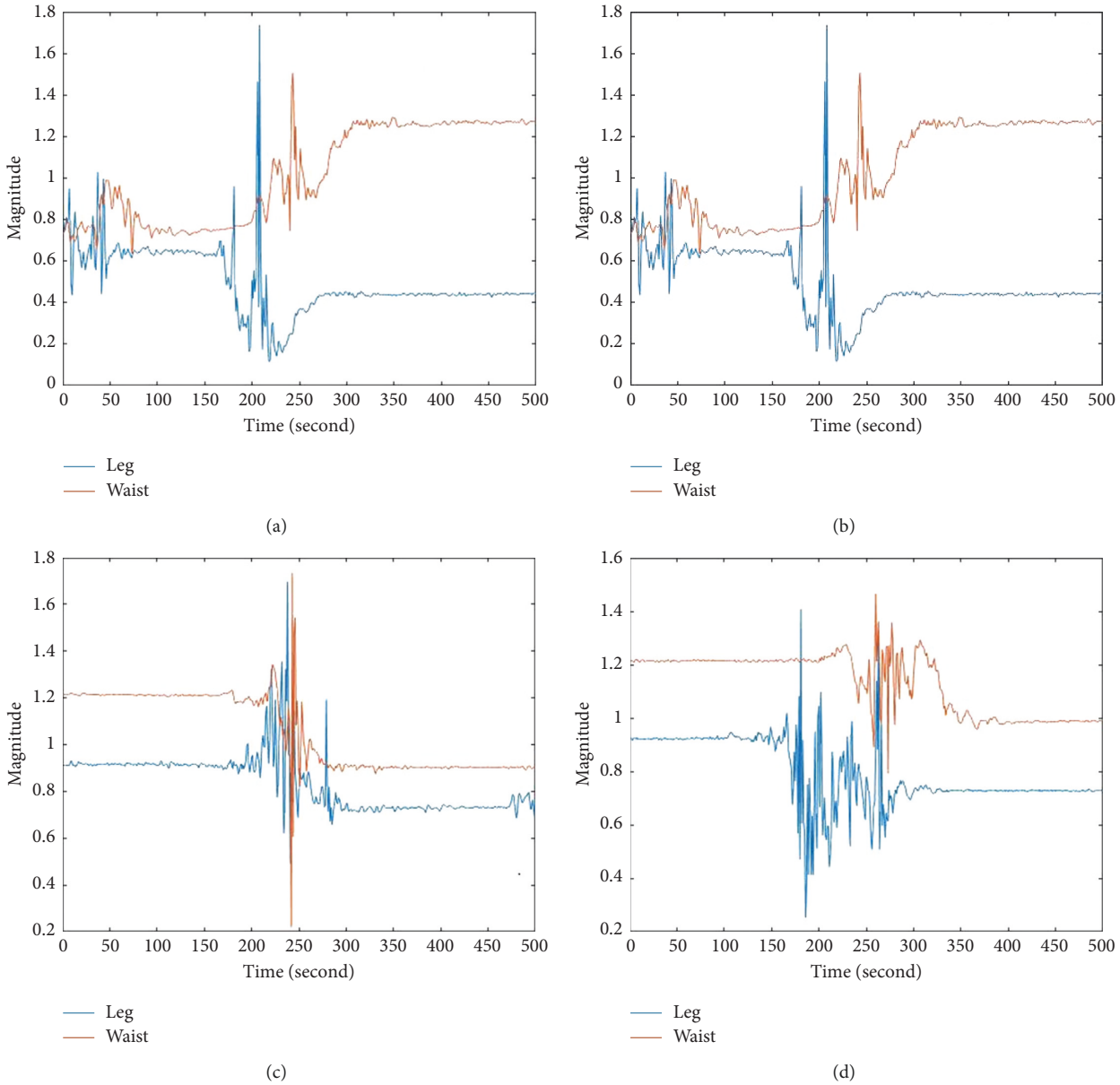

Figure 9: The magnitude of sensor data observed for different fall types when multiple sensors at the waist + leg (calf) were used: (a) FOK; (b) FOL; (c) BSC; (d) SDL.

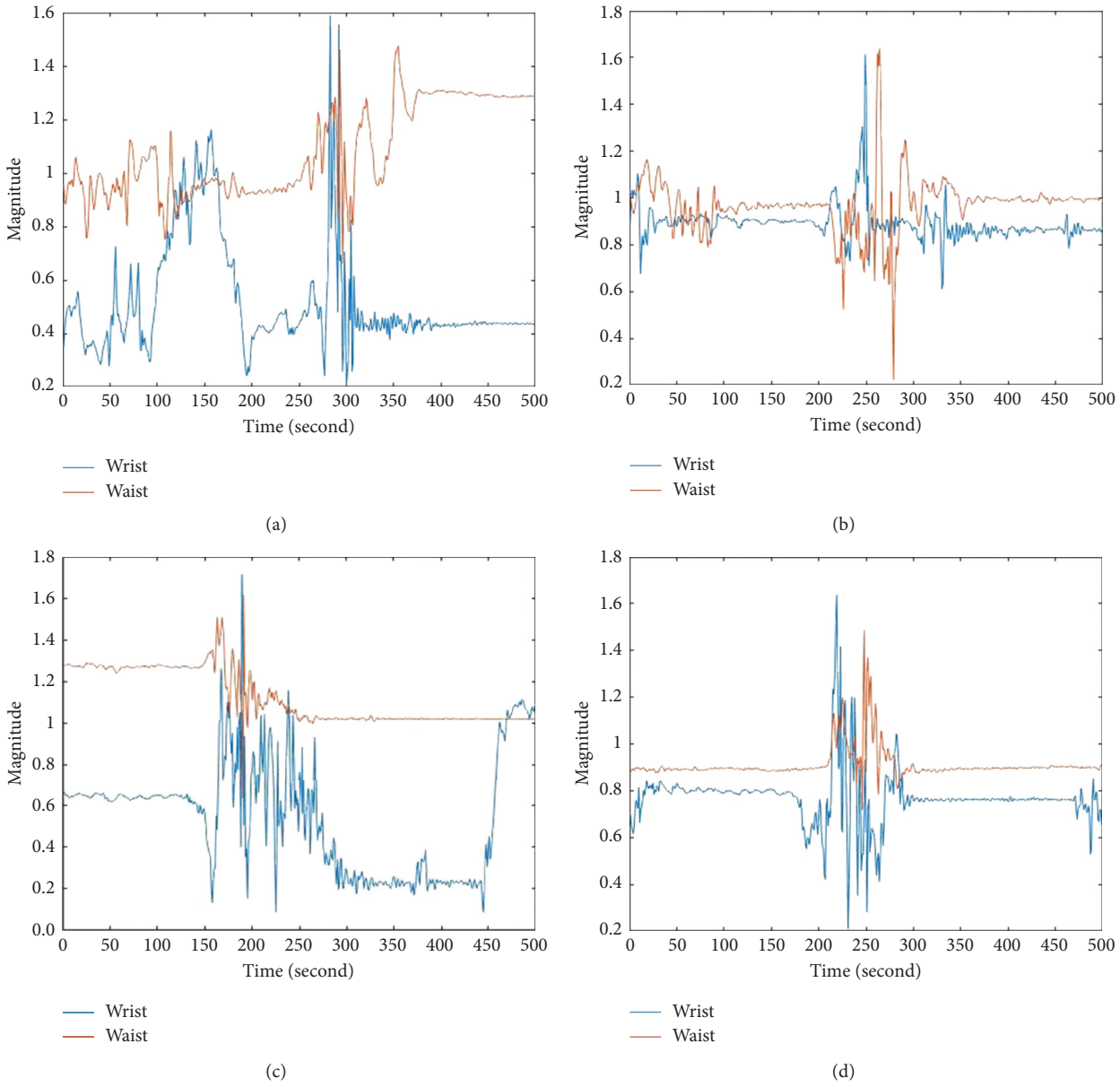

Figure 10: The magnitude of sensor data observed for different fall types when multiple sensors at the waist + wrist were used: (a) FOK; (b) FOL; (c) BSC; (d) SDL.

Table 3: Performance of the LSTM model across sensor combinations.

Experiment	Precision	Recall	Overall accuracy (%)
MobiAct	0.99	0.99	99.62
Waist (50 Hz)	0.75	0.64	85.2
Waist + leg (calf)	0.95	0.94	94.1
Waist + wrist	0.96	0.96	95.8

delay as explained in Section 4. The highest performance was obtained with the waist + wrist combination giving 0.96 precision, 0.96 recall, and 95.8% accuracy. From the results, it can be concluded that multistream data can be used to detect falls with high accuracy. Table 4 shows the performance comparison of some of the relevant work.

Table 4: Performance comparison of relevant work.

Work	Technique	Performance (%)
Hsieh et al [29]	k-NN, SVM	k-NN ACC: 96.2
Colon et al [30]	Threshold	ACC: 81.3
Ajerla et al [31]	LSTM	ACC: 95.8

As can be observed in Table 4, the threshold-based approach discussed by Colon et al. [30] performed less accurately than the other two approaches in fall detection. They also identified the patterns associated with fall based on the locations of the smartphone in the user's body such as pants' side pocket, shirt chest's pocket, or texting. The performance of the k-NN classifier discussed by Hsieh et al. [29] is slightly better than the performance achieved by our LSTM model in

fall detection. But our experiments also provided insights on the optimum sensor combination as well as the optimum frequency for data collection for fall detection.

In the initial evaluation of the resultant model, garbage data were recorded when the connection was lost between the sensor and edge device. Before deploying in large scale, the framework should be tested with a second edge device to handle situations as described above such that when the first edge device loses connection, the 2nd device can be used to provide reliable service.

10. Conclusions

Fall detection can enable better patient care for the elderly people who are more prone to fall to allow emergency assistance. We presented an edge computing framework to detect fall automatically using real-time monitoring with cheap wearable sensors. Our LSTM model detected fall with 99% accuracy as reported in our prior work [21]. In this study, we deployed our LSTM fall detection model in an edge computing framework to monitor real-time patient activity and identify fall. We used wireless wearable sensor devices called MetaMotionR from MbientLab to monitor human activity, which sent real-time streaming data to an edge device. We chose a laptop as an edge device and implemented a data analytic pipeline using custom APIs from MbientLab, Apache Flink, and TensorFlow, to process the streaming sensor data. Experiments showed that our framework was able to correctly identify fall from real-time sensor data 95.8% of the time. We empirically determined the best positions for the sensors as the waist and the frequency of data collection as 50 Hz. We demonstrated that the use of multistream data using multiple sensors leads to better performance. By training models on existing published datasets and then refining the models proved to be efficient and can be used in transfer learning for other use case scenarios.

In future, we like to extend the framework to support multiple types of sensors, parallel data-processing pipelines, and a cloud platform to create a solution for monitoring patients at the clinics, hospitals, and retirement homes. We plan to develop machine learning models to identify other activities and analyse biometrics like the heartbeat of the subject before and after fall, sleep pattern, motion pattern, and track patients' activity using the MbientLab MetaTracker which can be especially useful for monitoring patients with Alzheimer's disease.

Data Availability

The code for execution of the pipeline are available on GitHub: https://github.com/Dharmitha/StreamDataMbient. The online public dataset MobiAct used in this research work was collected from [20]: https://bmi.teicrete.gr/en/the-mobifall-and-mobiact-datasets-2/.

Disclosure

This research is a part of Dharmitha Ajerla's Master of Science project [31].

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The authors would like to acknowledge the effort of the volunteers who participated in the data collection process.

References

[1] Global Connected Wearable Devices 2016–2021—Statistic, September 2019, https://www.statista.com/statistics/487291/global-connected-wearable-devices/.

[2] Falls, September 2019, http://www.who.int/en/news-room/fact-sheets/detail/falls.

[3] Time Spent in Emergency Departments-Health Quality Ontario, September 2019, https://www.hqontario.ca/System-Performance/Time-spent-in-emergency-departments.

[4] R. E. Roush, T. A. Teasdale, J. N. Murphy, and M. S. Kirk, "Impact of a personal emergency response system on hospital utilization by community- residing elders," *Southern Medical Journal*, vol. 88, no. 9, pp. 917–922, 1995.

[5] A. K. Bourke, J. V. O'Brien, and G. M. Lyons, "Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm," *Gait & Posture*, vol. 26, no. 2, pp. 194–199, 2007.

[6] A. K. Bourke, P. W. van de Ven, A. E. Chaya, G. M. OLaighin, and J. Nelson, "Testing of a long-term fall detection system incorporated into a custom vest for the elderly," in *Proceedings of the 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 2844–2847, Vancouver, Canada, August 2008.

[7] M. Kangas, A. Konttila, P. Lindgren, I. Winblad, and T. Jämsä, "Comparison of low-complexity fall detection algorithms for body attached accelerometers," *Gait & Posture*, vol. 28, no. 2, pp. 285–291, 2008.

[8] A. K. Bourke, P. van de Ven, M. Gamble et al., "Evaluation of waist-mounted tri-axial accelerometer based fall-detection algorithms during scripted and continuous unscripted activities," *Journal of Biomechanics*, vol. 43, no. 15, pp. 3051–3057, 2010.

[9] Y. He, Y. Li, and S.-D. Bao, "Fall detection by built-in tri-accelerometer of smartphone," in *Proceedings of the IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*, pp. 184–187, IEEE, Hong Kong, China, January 2012.

[10] Q. V. Vo, G. Lee, and D. Choi, "Fall detection based on movement and smart phone technology," in *Proceedings of the IEEE RIVF International Conference on Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF)*, pp. 1–4, IEEE, Ho Chi Minh City, Vietnam, February-March 2012.

[11] S. Abbate, M. Avvenuti, F. Bonatesta, G. Cola, P. Corsini, and A. Vecchio, "A smartphone-based fall detection system," *Pervasive and Mobile Computing*, vol. 8, no. 6, pp. 883–899, 2012.

[12] M. Yuwono, B. D. Moulton, S. W. Su, B. G. Celler, and H. T. Nguyen, "Unsupervised machine-learning method for improving the performance of ambulatory fall-detection systems," *Biomedical Engineering Online*, vol. 11, no. 1, p. 9, 2012.

[13] S. S. Khan and J. Hoey, "Review of fall detection techniques: a data availability perspective," *Medical Engineering & Physics*, vol. 39, pp. 12–22, 2017.

[14] T. Theodoridis, V. Solachidis, N. Vretos, and P. Daras, "Human fall detection from acceleration measurements using a recurrent neural network," in *Precision Medicine Powered by pHealth and Connected Health*, pp. 145–149, Springer, Berlin, Germany, 2018.

[15] P. Cheng, L. Tan, P. Ning et al., "Comparative effectiveness of published interventions for elderly fall prevention: a systematic review and network meta-analysis," *International Journal of Environmental Research and Public Health*, vol. 15, no. 3, p. 498, 2018.

[16] T. Tamura, T. Yoshimura, M. Sekine, M. Uchida, and O. Tanaka, "A wearable airbag to prevent fall injuries," *IEEE Transactions on Information Technology in Biomedicine*, vol. 13, no. 6, pp. 910–914, 2009.

[17] Z. Zhong, F. Chen, Q. Zhai et al., "A real-time pre-impact fall detection and protection system," in *Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 1039–1044, IEEE, Auckland, New Zealand, July 2018.

[18] C.-F. Lai, M. Chen, J.-S. Pan, C.-H. Youn, and H.-C. Chao, "A collaborative computing framework of cloud network and WBSN applied to fall detection and 3-D motion reconstruction," *IEEE Journal of Biomedical and Health Informatics*, vol. 18, no. 2, pp. 457–466, 2014.

[19] D. Yacchirema, J. S. de Puga, C. Palau, and M. Esteve, "Fall detection system for elderly people using IoT and big data," *Procedia Computer Science*, vol. 130, pp. 603–610, 2018.

[20] G. Vavoulas, C. Chatzaki, T. Malliotakis, M. Pediaditis, and M. Tsiknakis, "The mobiact dataset: recognition of activities of daily living using smartphones," in *Proceedings of the International Conference on Information and Communication Technologies for Ageing Well and e-Health*, pp. 143–151, Rome, Italy, April 2016.

[21] D. Ajerla, S. Mahfuz, and F. Zulkernine, "Fall detection using physical activity monitoring data," in *Proceedings of the 7th International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, Big Mine 2018*, ACM SIGKDD, London, UK, April 2018.

[22] G. Vavoulas, M. Pediaditis, C. Chatzaki, E. G. Spanakis, and M. Tsiknakis, "The MobiFall dataset," *International Journal of Monitoring and Surveillance Technologies Research*, vol. 2, no. 1, pp. 44–56, 2014.

[23] J. R. Kwapisz, G. M. Weiss, and S. A. Moore, "Activity recognition using cell phone accelerometers," *ACM SigKDD Explorations Newsletter*, vol. 12, no. 2, pp. 74–82, 2011.

[24] Z. Wang, Y. Zhang, Z. Chen et al., "Application of reliefF algorithm to selecting feature sets for classification of high resolution remote sensing image," in *Proceedings of the 2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pp. 755–758, Beijing, China, July 2016.

[25] Wearable Device Unit Shipments Worldwide 2017–2019, 2021—Statistic, September 2019, https://www.statista.com/statistics/385658/electronic-wearable-fitness-devices-worldwide-shipments/.

[26] Wristband Sensor Research Kit, September 2019, https://mbientlab.com/store/wristband-sensor-research-kit/.

[27] What is Apache Flink?, September 2019, https://flink.apache.org/flink-architecture.html.

[28] L. Ren, W. Shi, Z. Yu, and Z. Liu, "Real-time energy-efficient fall detection based on SSR energy efficiency strategy," *International Journal of Sensor Networks*, vol. 20, no. 4, pp. 243–251, 2016.

[29] C.-Y. Hsieh, C.-N. Huang, K.-C. Liu, W.-C. Chu, and C.-T. Chan, "A machine learning approach to fall detection algorithm using wearable sensor," in *Proceedings of the International Conference on Advanced Materials for Science and Engineering (ICAMSE)*, pp. 707–710, IEEE, Tainan, Taiwan, November 2016.

[30] L. N. V. Colon, Y. de la Hoz, and M. Labrador, "Human fall detection with smartphones," in *Proceedings of the 2014 IEEE Latin-America Conference on Communications (LATINCOM)*, pp. 1–7, IEEE, Cartagena de Indias, Colombia, November 2014.

[31] D. Ajerla, "A real-time patient monitoring framework for fall detection," Queen's University, Ontario, Canada, M.Sc. project, 2018, http://cs.queensu.ca/~farhana/assets/Sample_MSc_Project.pdf.