International Journal of Neural Systems, Vol. 0, No. 0 (2005) 1–??
© World Scientific Publishing Company

# Personalized Watch-based Fall Detection Using a Collaborative Edge-Cloud Framework [*]

Anne Hee Ngu, Vangelis Metsis, Shuan Coyne[†], Priyanka Srinivas[‡], Tarek Salad Uddin Mahmud
*Department of Computer Science, Texas State University, 601 University Drive*
*San Marcos, TX, 78666, USA*
*E-mail: angu@txstate.edu*

Kyong Hee Chee
*Department of Sociology, Texas State University, 601 University Drive,*
*San Marcos, TX,78666, USA*

The majority of current smart health applications are deployed on a smartphone paired with a smartwatch. The phone is used as the computation platform or the gateway for connecting to the cloud while the watch is used mainly as the data sensing device. In the case of fall detection applications for older adults, this kind of setup is not very practical since it requires users to always keep their phones in proximity while doing the daily chores. When a person falls, in a moment of panic, it might be difficult to locate the phone in order to interact with the Fall Detection App for the purpose of indicating whether they are fine or need help. This paper demonstrates the feasibility of running a real-time personalized deep-learning-based fall detection system on a smartwatch device using a collaborative edge-cloud framework. In particular, we present the software architecture we used for the collaborative framework, demonstrate how we automate the fall detection pipeline, design an appropriate UI on the small screen of the watch, and implement strategies for the continuous data collection and automation of the personalization process with the limited computational and storage resources of a smartwatch. We also present the usability of such a system with nine real-world older adult participants.

*Keywords*: fall detection, smart health, model personalization, deep learning, edge computing

[*]This work is supported by the National Science Foundation under the Research Experiences for Undergraduates Program (CNS-1358939, CCF-1659807, CNS-1757893) as well as the recent NSF-SCH award (2123749) at Texas State University.
[†]Shuan has graduated and his current email is mickcoyne@mailbox.org
[‡]Priyanka has graduated and her current email is priyankasrinivas37@gmail.com

## 1. Introduction

Wearable smartwatches paired with smartphones have brought health monitoring applications, such as fall detection, closer to reality. However, a one size fits all algorithm such as Apple Watch's "hard fall" detection or even more advanced deep learning models[1] have proven to be ineffective at covering all patterns of falls and ADL (Activities of Daily Living) data. Our previous work,[2] using simulated data from fourteen young and healthy adults, demonstrated that we can detect most falls as well as ADLs by utilizing a personalization strategy. This strategy involved a deep learning model trained offline on simulated falls, plus labeled ADL data collected from the user (feedback data) while wearing the watch for a specified period. The feedback data from each user was used to create a personalized fall detection model that had over 90% recall with very few false alarms. However, there are two main issues with this personalized fall detection system.

First, our previous fall detection application, called SmartFall, runs the user interface (UI) on the phone, with the watch used mainly for sensing of accelerometer data. This is problematic because older adults have difficulties keeping up with devices that are not directly attached to them. Moreover, when an older adult falls, in a moment of panic, it might be difficult to locate the phone in order to interact with the App for the purpose of indicating whether they are fine or need help if the App is running on the phone. A watch's UI, on the other hand, would allow interaction with the SmartFall App at any time and anywhere.

The second issue in the previous system is that, creating a personalized model was done manually as a proof of concept. The SmartFall App is designed to save the collected data on the phone using a CSV file format. After data have been collected for a period of time, a programmer has to manually organize the data in a file to prepare it for re-training. This is not scalable when the system is being used by more than a handful of users in a nursing home and leaves room for human error.

We propose a scalable solution by automating the entire personalization process using a collaborative edge-cloud framework, from the user initially wearing the watch, to getting feedback or labeled ADL/fall data from the user, re-training a new fall detection model tailored to the user, validating the new model on the cloud, and finally, pushing the new model to the watch automatically. Some of the challenges for automating the personalization process on the watch include continuous collection and robust archiving of labeled data on a limited watch's storage, keeping track of the best personalized model and personalized training dataset for each user, conserving the battery power of the watch, and the validation and selection of the new model daily.

Our solution involves migrating the SmartFall App (UI and the prediction logic) to a single device (smartwatch) and using a robust and efficient Couchbase[3] storage system on both the watch and the cloud for data collection and archiving. The Couchbase on the watch and on the cloud can be synchronized periodically and allows the data on the watch to be purged automatically after synchronization. Couchbase on the cloud provides a central place to store all users' feedback data reliability including tracking the best personalized fall detection model for each user, the personalized training dataset for each user, and most important of all, the fast retrieval of user's feedback data for re-training on the cloud.

We demonstrate the feasibility of automated real-time personalized fall detection and deployment of the system on a commodity-based smartwatch. We describe how we robustly collect labeled feedback data from the user in real-time, test the scalability of the automation pipeline, and the intuitiveness of the App's user interface on the watch. We further validated the usability of the system by recruiting nine real-world participants to wear the watch for three hours each day for seven continuous days. The main contribution of the paper is a prototype data engineering architecture consisting of the following components:

- A simplified UI on the watch interface tailored to the small screen space and easy to use for seniors.
- Automated personalization pipeline including when to re-train, strategies used for re-training, and accurate offline validation of the new model.
- Robust archiving of feedback data using an in-memory queue structure and Couchbase, a NoSQL database.

- The edge-cloud software architecture that enables optimization of the battery power and storage space of the watch. The fall detection App is able to run in the background and only uploads the collected data via a Wi-fi connection at a configurable interval.
- The usability study of the watch-based SmartFall App with nine older adult fall risk participants.

We would like to emphasize that in our system, the fall detection at inference time runs solely on the watch, so there is no latency incurred during prediction since data do not need to be transferred to the server. The watch does not require constant connectivity to the internet and the type of connectivity is dependent on the type of watch. Internet connectivity is needed only for periodical data archiving and for downloading new personalized fall detection models. We prototyped our system on a watch with only WiFi connectivity but that does not imply that our system is for indoor only use or that cellular data cannot be used for communication with the server. Our system is designed to store the sensed data on the local storage first. When internet connection is not detected, so long as there is sufficient local storage, the collected data will be archived locally and sent to the server later.

The paper is organized as follows. In Section 2, we present the background and previous work related to our problem. The overview of the system architecture is presented in Section 3. The edge-cloud database synchronization is described in Section 3.5 followed by the personalization automation pipeline in Section 3.6. The evaluation of the pipeline is presented in Section 4. The usability study of the SmartFall App by older adult participants is presented in Section 5. Finally, we presented our conclusion and future work in Section 7.

## 2. Background and Related Work

A recent survey on fall detection systems shows much progress in using machine learning to detect falls given accelerometer data.[4] The datasets used to train models are all synthetically created by utilizing simulated falls and ADLs collected in controlled experiments with primarily young, healthy adult participants.

There has been a wide range of success, however the most success has been achieved using custom hardware mounted on the chest or waist. Unfortunately, chest or waist mounted fall detection systems can be invasive, uncomfortable, or self conscious for users to wear in public. Other systems that range from infrared monitoring[5] to location monitoring[6] all require wearable custom hardware.[4] It is not reasonable to setup a custom array of cameras and sensors throughout a home to detect falls; not only is it invasive, but also it does not help seniors who need to venture outside of the detection area. This is one of the main motivations for creating a fall detection system on a single commodity-based wearable device such as smartwatch which has unrestricted mobility. Moreover, a smartwatch-based fall detection is portable and does not require video/cameras or custom devices. Many of the custom hardware solutions involve mounting the system in a very specific position of the person, usually around the waist or chest. If the device's position is altered, the system stops working. Additionally, even a small device such as a pendant can be difficult to use, and overall frustrating as pointed out by Ref. 7. Thus, we propose a smartwatch-based fall detection system as a familiar device that an elder person would be more inclined to use.

Another challenge of fall detection system is the high false positives generated. A survey paper in 8 described the various strategies used to help combat false positives. However, this remains an unsolved issue. This, in large is due to difficulties in obtaining large amount of quality labeled data for model training. Not only are the datasets synthetic and not representative of the elderly population, but also they are relatively small with limited variation in types of falls and ADLs. When taken to the real world, any activity not represented in the training set can lead to a false positive. This could lead to hundreds, if not thousands of incorrect alarms when scaled to a single nursing home.[8] There have been some proposed strategies to reduce false positives by detecting relevant context to a fall. Specifically, it has been proposed that if you can detect a fall and someone lying still, then they have truly fallen.[9–11] This strategy greatly reduced the false positives. However, this assumes expert knowledge on a dataset that does not exist. Currently, there is no dataset of elderly people falling or performing ADLs while wearing a watch-based accelerometer sensor. There are various instances in which a fall occurs but acceleration data could continue to be recorded. These cases could be things such as Parkinson's, seizures, injury or any instance in which the user is awake but unable to get up or dial for help. We do not know to what proportions of elderly falling resulted in total stillness vs continued movement. While false positives are annoying, false negatives can be deadly. Therefore, any proposed system will need to rely solely on its ability to learn patterns of falls and ignore patterns in ADLs without expert knowledge.

Most recently, personalization has been used to reduce false positives. This has been achieved by two strategies. Both strategies utilized some form of a generic model that was trained on a synthetic fall dataset from wrist-worn devices. The first system utilized a bag-of-words strategy to collect labeled FP (False Positive) data from the user. Each ADL was added to the bag and future detected falls were compared against these previous ADLs. If the data was similar, then it was an ADL. Otherwise, it was a fall. After each detected fall, the user could confirm if this was a fall or another ADL.[12] This personalized bag of words was able to reduce some of the false positives without affecting recall. This system also attempted to use common ADLs for transfer learning, such that new users could benefit from this labeled data. However, it was found that most of the labeled data in the bag was never encountered again. Meaning the bag kept growing as new ADLs kept being received. This does not scale well for mobile devices. There was a severe lack of commonly occurring ADLs and this prevented transfer learning from being a practical solution.

Our personalization of fall detection was first reported in 2. We demonstrated that we could maintain the high recall (sensitivity) of a wrist-worn watch based fall detection system as well as increase the precision (specificity) by collecting personal false positive data, including them in the dataset and then re-train the model. This strategy uses an RNN ensemble model first trained with simulated falls and ADL data to detect if a user has fallen given the user's accelerometer data collected from the watch. Then the user is asked to wear the watch for a few hours. After a couple rounds of various ADLs performed by the user, the collected false positive data is used in combination with the original dataset to regenerate a new model, thus creating a "personalized" fall detection model. This personalized model was nearly twice as precise as the generic model (a model trained without personal ADL data). This process shows that we can create a system that learns the difference between falls and ADLs without expert knowledge. It needs only feedback from the user to increase its precision.

Tsinganos and Skodras[13] also studied personalization of fall. They used a traditional K-Nearest Neighbor (KNN) machine learning algorithm while we used deep learning. Their acceleration data was sampled at 50 Hz compared to ours at 31.2 Hz. Their data was collected using a smartphone while ours was via a smartwatch. To incorporate personalization, the authors added the misclassified ADLs (i.e. false positives) back into the training dataset one sample at a time and concluded that seven samples could reduce false positives by 10%. Similarly, our personalization strategy also collected the false positives and added them back to original training dataset. However, we collected and added the false positives data in batches. The size of the batch is dependent on how long the watch is being worn continuously and how many false positives are generated during that period of time.

Another approach that has been traditionally used to improve the behavior of systems or agents over time is Reinforcement Learning (RL) and its recent adaptations to Deep Reinforcement Learning.[14] Although the goal of our system is to improve its accuracy over time, by collecting personalized information, RL is not the best strategy to follow in our case, as RL is more well-suited to applications where the learning agent receives immediate feedback about the effect of each action/decision. A more suitable approach for our problem is the one known as Incremental Learning.[15]

The personalization process in our prior work is manual and the fall detection App only runs on a smartphone with the watch being a data sensing device. In this paper, we will show how to automate the personalization process using edge to cloud collaborative framework so that fall detection can be adapted to a particular person in real-time and run on a single wearable device.

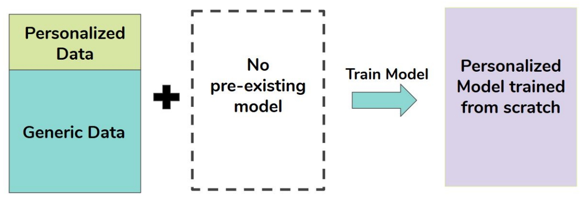


Figure 1: Training from Scratch Strategy.

## 3. SmartFall on a Wearable device with Edge-Cloud Collaboration

Our main goal is to demonstrate the feasibility of implementing a practical and robust automated personalized fall detection system on a single personal device (i.e. smartwatch) within an edge-cloud collaborative framework.

To achieve that, we first ported the deep learning model and the prediction algorithm in our prior work[2] to run on the watch and verified that there was no loss of accuracy or delay in prediction. Personalization requires continuous collection of feedback data that needs to be stored locally on the watch to reduce the communication overhead with the cloud. Since the storage on the watch is very limited (only 2 GB is available on the test watch after installing WearOS), periodically, there is a need to upload the collected data without personal identifying information robustly to the cloud to free up space on the watch for further collection of data. After a new model is validated to be better based on user's feedback, that model will be downloaded to the watch to replace the previous model without asking the user to re-install the App.

### 3.1. *Overview of the personalization process*

We utilized the training from scratch (TFS) personalization strategy described in our previous work[2] for re-training. This personalization strategy reduces false positives (increases precision) by combining a generic dataset containing simulated falls and ADL data from 14 volunteers with personal ADL data collected from the user via simple feedback. TFS aims to improve the model by re-training from scratch with additional false positive data samples collected from a specific user. Figure 1 is an illustration on TFS training strategy.

The personalization strategy enables us to create models that are highly tuned to the user's personal ADL patterns. The personalization process starts when a user is asked to wear the watch for the first time for half an hour of prescribed ADL activities (this list includes common tasks such as walking, hand waving, sitting down, standing up, changing clothes, picking up objects, washing hands, eating food, brushing teeth, etc). This is referred to as the calibration phase or the first round of personalization. During this phase, whenever the system generates a prediction, the user will provide feedback through the watch's UI, see Figure 2.

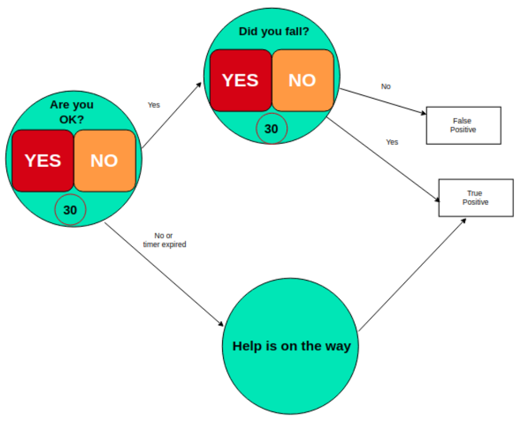


Figure 2: User interface display after a fall is detected.

The labeled feedback data is stored locally in a Couchbase database on the watch and will be uploaded periodically to the cloud's Couchbase database. On the cloud, during the night and when a certain number of false alarms have been generated, the re-training of the model is initiated and a new model is created. This model is validated and will be automatically pushed onto the watch if it is deemed to be a better model. An overview of the SmartFall software architecture system with personalization is shown in Figure 3.

### 3.2. *Watch-based Fall Detection App (SmartFall)*

Our original SmartFall App presented in[16] had all of the user interface's screens on the phone. Accelerometer data are continuously streamed from the watch to the phone. When a fall has been detected by the SmartFall App on the phone, an alert text message can be sent to the caregiver if the user did not re-

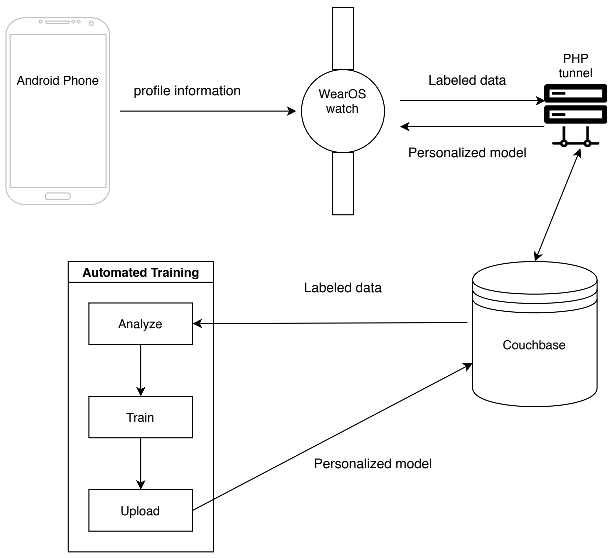


Figure 3: Overview of the system data flow.

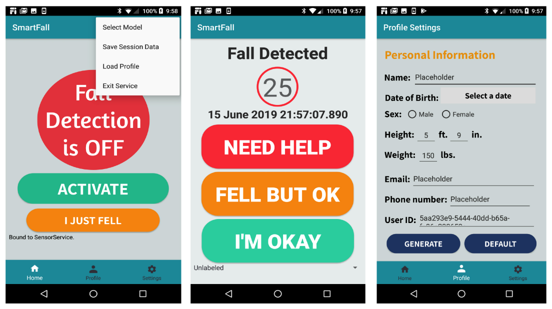


Figure 4: SmartFall User Interface on the Phone.

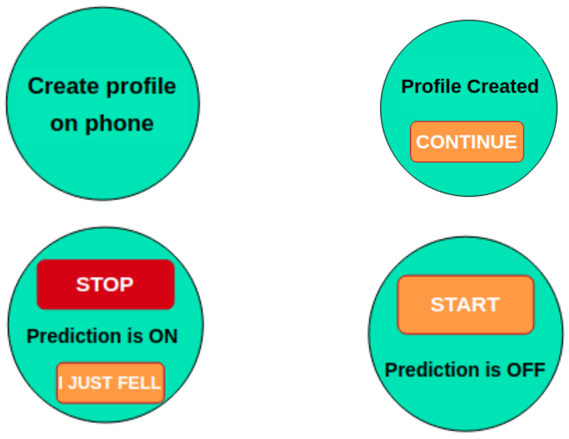


Figure 5: Various watch-based UI screens.

spond within a configurable fixed time interval (e.g. 30 seconds). Figure 4 shows three core user interfaces (UI) for the phone-based SmartFall App. The screen on the left shows the home screen UI for the application and the screen on the center shows the UI when a fall is detected.

Since the watch has a very small screen space. To migrate the user interface to the watch, the UI screen to collect feedback from the user after a fall prediction is re-designed to start with just two "yes" and "no" buttons (see figure 2) rather than listing all the choices as shown in the center screen in Figure 4. If the user pressed "yes", the next screen will ask whether the user needs help or not. If the user did not press either "yes" or "no", after a specified period of time, an alert message will be sent automatically to the designated caregiver. We followed the best practices advocated in 17 for the design of the UI for older adults. The three main principles we adopted were a strict color scheme with high contrast, legible and big fonts, simple description of the system to engage them to use it.

We leave the profile creation on the phone. Profile creation (Figure 4) needs to be done only once and it is very tedious to create the profile using the watch's UI. Profile creation includes generating a unique id for a user as well as collecting contact details of the carer in case of a fall.

After the watch and the phone were paired, by opening the SmartFall App on the phone and then opening the corresponding App on the watch, the watch will display "Create profile on phone". After the user created the profile on the phone and pressed the "upload" button on the phone, the watch will display "Profile Created" as shown in Figure 5. The SmartFall App will be activated and run on the watch (refer to Figure 5) when the user pressed the "continue" button on this UI. From now onwards, the user only needs to interact with SmartFall App via the watch. There is no need to interact with the App unless the system detects that a fall has occurred. The watch will vibrate to alert the user that a prediction has occurred and the UI in Figure 2 will appear.

When the user presses "STOP" button on the SmartFall App (Figure 5), the system is deactivated, any remaining data collected during this period are uploaded to the cloud before shutting down the App. For privacy concerns, the only data being uploaded are the labeled accelerometer data with system-generated user-id attached. No other profile information is ever uploaded to the cloud.

### 3.3. *Real-time Fall Prediction*

One of the requirements for fall detection is that the sampling rate must be sufficiently high to capture a fall and samples must be received in order. The system should not miss any real fall (i.e. high recall/sensitivity) and it should not generate too many false alarms (i.e. high precision/specificity). The system must be able to predict falls in real-time with very little time delay. We determined through experimentation[18] that 35 data points received/sampled about every 32 ms is sufficient to capture the general pattern for a fall.

The fall prediction is made using a pre-trained deep learning model created in TensorFlow 2.0 as described in 19. Our earlier work demonstrated that using an ensemble of four LSTM RNN models to make predictions for falls yields a slightly better precision compared to a single model. This model architecture was adopted in previous iterations when inference ran on the smartphone and the watch acted merely as a sensing device. In the current version, where inference runs directly on the smartwatch, the added computational complexity cannot be justified by the minor increase in precision. Therefore, we use a single RNN model which still achieves the same recall as the four RNN models but with slightly lower precision. However, this lower precision, as demonstrated in this paper, can be mitigated by personalization which enables the SmartFall App to collect feedback data from the user to improve the model dynamically. The fall prediction is made on a sliding window of data that is 35 samples (time-steps) in length. Each prediction output a probability of fall between 0 and 1. The sliding window shifts by one time-step at each prediction. That means consecutive windows have a $K-1$ time step overlap, where $K$ is the size of the window. Figure 6 displays our model architecture.

The model contains an input layer, two hidden layers, and an output layer. The input layer contains 3 nodes for the raw data; the accelerometer (x, y, z) vectors. It then feeds through our hidden layers: a recurrent layer of size 30 LSTM nodes, and a fully connected dense layer of size 30 nodes. The output is a 2-node softmax layer which outputs a predicted probability that a fall has occurred. The input data is batch normalized, a batch size of 64 is used. The BinaryCrossEntropy with ADAM optimizer is used for the loss function. This model is lightweight relative to many deep learning architectures, and makes inference computation much more efficient for mobile devices.

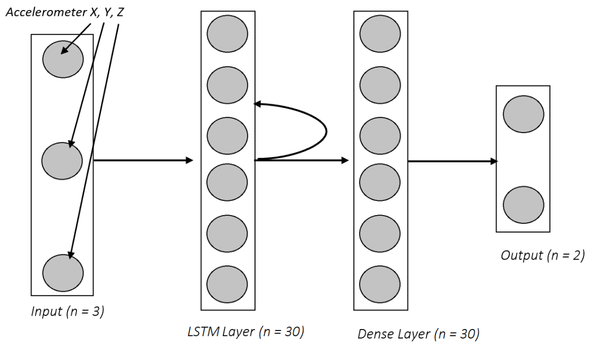
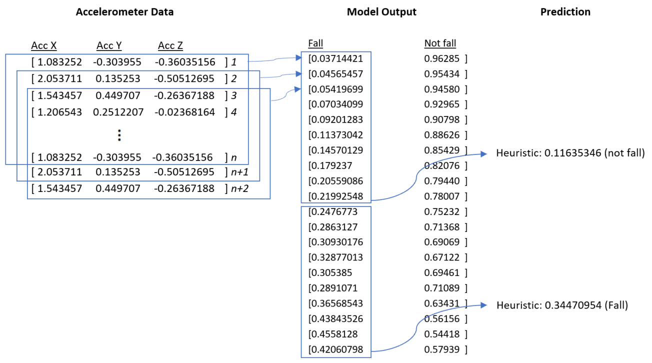


Figure 6: Deep learning model architecture.



```
Acc X       Acc Y       Acc Z
[ 1.083252  -0.303955   -0.36035156  ] 1
[ 2.053711   0.135253   -0.50512695  ] 2
[ 1.543457   0.449707   -0.26367188  ] 3
[ 1.206543   0.2512207  -0.02368164  ] 4
              ⋮
[ 1.083252  -0.303955   -0.36035156  ] n
[ 2.053711   0.135253   -0.50512695  ] n+1
[ 1.543457   0.449707   -0.26367188  ] n+2
```

```
Fall           Not fall
[0.03714421    0.96285 ]
[0.04565457    0.95434 ]
[0.05419699    0.94580 ]
[0.07034099    0.92965 ]
[0.09201283    0.90798 ]
[0.11373042    0.88626 ]
[0.14570129    0.85429 ]   Heuristic: 0.11635346 (not fall)
[0.179237      0.82076 ]
[0.20559086    0.79440 ]
[0.21992548    0.78007 ]
[0.2476773     0.75232 ]
[0.2863127     0.71368 ]
[0.30930176    0.69069 ]
[0.32877013    0.67122 ]
[0.305385      0.69461 ]
[0.2891071     0.71089 ]
[0.36568543    0.63431 ]   Heuristic: 0.34470954 (Fall)
[0.43843526    0.56156 ]
[0.4558128     0.54418 ]
[0.42060798    0.57939 ]
```

Figure 7: Prediction scheme for the deep learning model.

Each prediction will output a probability of a fall and we average the last 20 probabilities of prediction together to infer fall or not fall. Basically, if the averaged probability reaches a threshold of 0.3, we determine that a fall has occurred. Figure 7 outlines this schematic.

The 0.3 threshold was determined via grid search to give the best results in our dataset and it is adjusted for different users via personalization. We use a nested queue data structure to store the sensed data in memory during prediction and mark them for archiving as labeled feedback data after getting confirmation from the user. The nested queue is visualized in Figure 8.

The original phone version of the SmartFall App is designed to archive the sensed data samples using a CSV file for simplicity. However, to keep track of

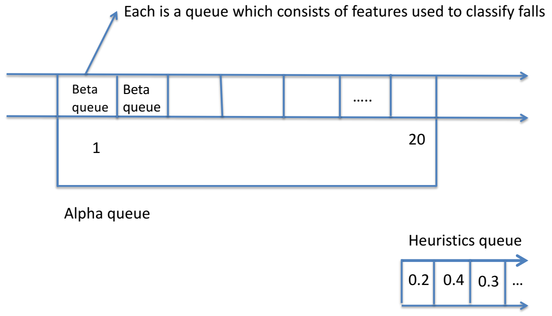


Figure 8: A visualization of the Alpha and Beta queue.

where to slide the window for each prediction, there is a need to store the line numbers of the CSV file in memory. A prediction was made by reading the CSV file and re-winding to the line number recorded. This solution was inadequate for the long term usage of the App, since the file would get bigger and bigger and the prediction time would get slower and slower when the App was being used continuously for a period of time. This problem is ultimately mitigated by storing the latest sensed data in the queue data structure in memory. Upon a successful prediction, the data in the queue is archived to the Couchbase database system on the watch. This storage system will be emptied periodically by uploading past archived data points to the cloud and removing them from the local storage to free up limited storage space on the watch.

The structure of the queue is as follows: the main queue is designed with a length of 20 (the number of predictions we wish to average over to get an accurate prediction). We call this the *Alpha queue*. The main role of alpha queue is to mark all the samples that have been used to make inference for a fall for ultimate storage in Couchbase with least I/O and for re-training. Data in each cell of the Alpha queue have $K - 1$ time steps (i.e. 34 data points) overlap with the next cell. This is done to ensure that for every new data point collected, we make a new fall prediction. Each item in the Alpha queue is a queue of samples (accelerometer data). We call these *Beta queues*. Each Beta queue is of length 35 (the prediction window size). A sample is simply one instance of linear acceleration data sampled at 32 ms and the timestamp from when it was sampled.

The Beta queue starts as an empty queue and is populated every 32 ms with the latest data from the watch's sensor. Once the Beta queue has reached its max length of 35 samples, the content is copied into the Alpha queue. That same 35 samples in the Beta queue is then used as input to the RNN Model for prediction, and the prediction result for those 35 samples is stored in a *heuristics queue*. Since Beta queue is limited to storing 35 samples, whenever a new sample has arrived in 32 ms time period, we pushed the new sample into the head of the Beta queue, and popped the oldest sample off the tail of the Beta queue. These 35 samples is again saved in the Alpha queue and then sent for another prediction. The predicted result is stored in the heuristics queue. This process repeats continuously. Once the Alpha queue is full (reached 20), we are ready to start making the final inference by checking whether the averaged probability over the 20 predictions is greater than the 0.3 threshold.

If the system infers that a fall has occurred (threshold is $> 0.3$), we empty the Alpha queue by saving the data to the Couchbase database on the watch with unknown label at this point because we do not yet know whether it is TP or FP. However, these data are saved immediately as they could contain true positive fall samples which are rare and valuable and thus must be archived robustly. This eager archiving of data also ensures that our system is robust in collecting user's feedback data which is very important for personalization. When the user provides the correct feedback on the UI, these data will be labeled and can be used for re-training as discussed in the next section. If no fall is predicted and the queue is full, we pop the oldest item (one Beta queue) out of the Alpha queue and save that as true negative (TN) data.

### 3.4. *Data Archiving*

Couchbase[3] is chosen as the storage system in both the watch and the cloud for the ease of data archiving. Couchbase was chosen for its ability to scale up easily, fast I/O, compact JSON format, and built-in synchronization of data between clients and servers with CAP (Consistency-Availability-Partition) guarantee. Details of Couchbase's server architecture can be found in .[20] In our implementation, we did not use the built-in Sync gateway module of Counchbase. Our application does not require synchronization in both directions. Moreover, a single node Couchbase

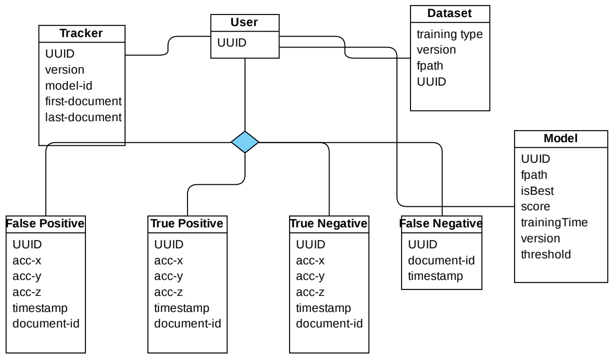


Figure 9: Structure of the archived data on the cloud/server.

server is sufficient for prototyping our system. We are mainly interested in sending data to the server for archival and sending a query to check for the availability of new models. There is never a need to synchronize raw data from to the server to the watch. Details of our data synchronization is found in section 3.5.

All data are stored using Couchbase's document structure. Figure 9 gives an overview of the structure of the archived data in the cloud/server. There are 4 document types used to store the sensed accelerometer data. These correlate to true positive ("TP"), false positive ("FP"), true negative ("TN"), false negative ("FN") data. These four types of data are tracked for each user for personalization. When a fall is detected, the entire Alpha queue's data in the memory is saved to the local database. Since data is processed in windows with $K-1$ time steps overlap, to save only unique data samples, we have to remove the overlapping data such that the samples are still in temporal order, and each sample is unique. To remove overlapping data, we use a simple strategy of popping/removing each Beta queue from the Alpha queue and only save the first sample of each Beta queue except for the last Beta queue (the 20th) where all data points are saved. This gives a final size of 54 samples being saved when there is a positive prediction. We need to save all the data used to make the 20 predictions. Since the prediction is done on a sliding window with $K-1$ data points overlap, the final prediction consists of 20 beta queues which has 54 unique samples. After obtaining the actual label from the user from the UI's prompt, the saved data is updated with the correct label of either TP (True Positive) or FP (False Positive).

When the Alpha queue is full and a fall has not been predicted, a Beta queue needs to be popped off and saved as TN (True Negative) data. Because of the overlap, we only save the the oldest item in each of the popped Beta queue. To minimize I/O, we accumulate true negative data samples to 375 before saving to the local database as TN. Since the majority of archived data is TN, the 375 is a number that is large enough (represents around 512 KB) to reduce the overhead of creating Couchbase documents and small enough for fast transfer to the cloud.

The final type of accelerometer data that needs to be archived are the FN (False Negative) data. These data are generated when a fall has occurred but was not detected by the application. We designed a button on one of the UI screens of the watch (see Figure 5) labeled "I JUST FELL" to allow for the recording of a timestamp marking the moment when a conscious user indicated that a fall occurred but was missed by the fall detection system. This is used by the system to re-label sections of the data as fall. False negative information is thus saved to the database in a simple meta record consisting of only the timestamp and user-id to facilitate the relabeling process later. The labeling of FN will incur some level of inaccuracy because the recorded timestamp might not align with the moment the person falls. To mitigate this misalignment, we label 54 data points before the recorded timestamp as FN. The number 54 is the number of samples we save for true or false positives.

Other important database structures that are needed for automatic personalization are the *Tracker*, the *Model* and the *Dataset*. The tracker document is first created by the SmartFall App on the watch. It associates a specific deep learning fall detection model the App was using when it was activated. As better models were downloaded to the watch, newer tracker documents were created to track which set of feedback data was recorded with which model. Tracker document contains the *ID* of the first and last document recorded when using the model. It also contains the name of the current *model* and its version number, and the *UUID*, the id of the user, which tells us which user this tracker document belongs to.

The Model document is used to store different personalized fall detection models that have been generated for a user. The *fpaths* field is an array that stores the filenames and locations of all the models.

This is designed as an array to accommodate multiple learning models that might be needed in the future. The *version* field keeps track of the latest model while the *isBest* field is a Boolean that determines which model is the best. The *scores* field is a map of all the statistics generated during offline validation (including the precision and recall curve) when the best model was tested on the test dataset. The training time is the number of seconds it took to generate this model. The *threshold* stores the best threshold value to use for the best model.

The Dataset document is used to store the *training dataset* used for each user. With personalization, each model is trained using data specific to a user. This document contains a *training type* parameter that identifies which re-training strategy was used. Currently, we only experimented with Training From Scratch (TFS). This attribute enables us to scale to other re-training strategies if they are available. The *version* is used to keep track of the version of the dataset used. The *fpaths* is the file name of the CSV file and the location of the file that contains the training data of a specific user.

The watch database used the same document type for storing the collected accelerometer data as on the cloud database for the ease of synchronization. It does not have the Model and the Dataset documents. The watch database has an additional document called User Profile which stores information such as the caregiver's contact details.

Note that if the watch accelerometer data is stored using a traditional relational database management system, a design with the least redundancy will require data to be stored using two relations. A meta table that indicates who owns a specific data sample and a separate table for each of the accelerometer data samples recorded every 32 ms. In order to retrieve accelerometer data for a specific user for re-training purposes, the query will involve the join over two large tables and will incur high latency. This will have a big impact on the scalability of the personalization pipeline.

## 3.5. *Database Synchronization*

Synchronizing data collected on the watch to the cloud database is a core part of achieving automation in personalization process of fall detection. We upload 20 saved documents in batches periodically (the interval can be configured) to avoid continuous usage of the watch's Wifi connection which can drain the battery.

Once data are confirmed to be uploaded successfully, we delete them from the watch's database to free up storage. A Tracker Document is designed to synchronize the fall detection model used on the watch and the cloud's database. The Tracker details are also uploaded to the cloud database to keep track of the latest model being used on the watch, but never deleted from the watch.

All archived data is associated with a user-id which is a 32-letter string that is generated using a random number generator during a profile creation. Here is an example of a UUID: *9d94c957-5f4b-49ba-b555-b31db45ca9f7*. Each time the user starts the SmartFall App, the App queries the cloud database for the best model using the UUID. If the watch does not already have the best model, it is then downloaded to the watch. Fall detection will then proceed to start with the best model.

## 3.6. *Automation of model validation and selection*

The automation is divided into two parts; both parts are run and co-dependent on each other. The first part is run to evaluate the user's feedback data stored in the database, it determines if the current model is generating too many false positives or false negatives and the system needs to train a new model. If the decision is to re-train, the request is passed on for the training of a new model to the second part.

The second part of the system handles the training of the model in the cloud using GPU, offline validation of the trained model, and saving of the new model to the cloud database for eventual transfer to the watch.

### 3.6.1. *Part One*

*Criteria for Re-Training* We analyze the archived data of a user in the database within a specific time interval to see if we need to train a better model. Since each user's fall detection model and the archived data are tracked using the Tracker documents, we simply need to grab the latest tracker document for each user. This tracker document contains the first and last document ID that contains data relevant to the model the user is currently using. With the relevant data retrieved, we need a way to evaluate how the current model is performing. If the model is performing well, there is no need to re-train.

It is expected that activities that produce high acceleration values on the wrist will generate more false positives than sedentary activities. To evaluate the number of false positives that occur for each user while controlling for different activity levels and life styles (e.g. active subjects vs. less active ones), we leverage a new custom metric for measuring false positives for evaluation in.[19] This new metric takes into consideration the number of acceleration "spikes" that occurred during a time period and uses those as a normalization factor for the false positive count. We consider a "spike" to occur when the magnitude of acceleration for a user exceeds the double of their average acceleration. The average acceleration is computed by processing all the acceleration data recorded during the evaluation period. When a spike is detected, we ignore the next 16 sample points (approximately half a second) before examining another spike. This ensures that a single jump in acceleration magnitude, which could generate consecutive high sample points, is not read as multiple spikes. The total number of false positives (FP) a particular model detects will be compared against the total number of spikes a user emitted to give our *Normalized Precision (NP)* value:

$$NP = \frac{\#\_of\_spikes - FP}{\#\_of\_spikes} \tag{1}$$

In brief, spikes measure performance in relation to activity levels. We choose .98 to be the threshold for which if the spike score achieves, we do not need to re-train. This means that 2 percent or less of all high acceleration activities result in a false alarm. If the spike score does not reach the threshold, the retrieved FP data is prepared for re-training.

*Data Trimming and dataset creation* In analyzing when re-training should happen, we use all the captured data in a specific interval to calculate the spikes. Note that majority of data collected from a user is TN (True Negative) data which can contain large amounts of low acceleration data if the user is not active. Adding too much low acceleration data to the original dataset will result in a highly unbalanced fall training data set.

This trimming process removes some low accelerometer data such that the percentage and diversity of each type of data should remain the same as the original generic dataset. We found that removing data points that are not within 750 data points from a spike (roughly 24 seconds) and keeping a buffer of around 250 data points on each spike (roughly 6 seconds) will achieve the right diversity of data. The data remains in chronological order after being trimmed; however, now all long periods of low acceleration data are removed.

After the data are trimmed, a new training dataset is created and appended to the original dataset (generic set), then written to a CSV file with a new version number. The file path of this CSV file is uploaded to the database as the latest training set. Finally, the file path of the new training set is passed to the second part of the system that is responsible for model creation and validation.

### 3.6.2. *Part Two*

Model Generation Creating models in TensorFlow 2.0 using the TFS (Training From Scratch) method has been described in earlier works.[2] To recap, the TFS method discards the previous model and trains a new model from a random initialized state using the new dataset. This new dataset is a combination of the original dataset with new data appended to it. The re-training takes place in the cloud/server. To manage multiple users using this personalized SmartFall system, we implemented a FIFO queue to schedule re-training automatically. An array consisting of the UUID, version number of model, training dataset path, and the testing dataset path is used to store the details of each job. The training thread periodically checks the queue every few minutes. If there is a job in the queue, it schedules the job to run using a GPU server in the cloud/server. Each job in the queue is run one by one sequentially until the queue is empty.

*Model Validation* Once training is complete, the new model must be validated offline. A high-performing personalized fall detection model is a model that has high sensitivity and specificity. A missed fall is represented as a False Negative (FN) and a "false alarm" is represented as a False Positive (FP). Since we are dealing with time-series data, validation of the model needs to account for the sequential nature of the data. We evaluate the model on the test set using a simulation program that replicates how predictions are made in real time as mentioned earlier in Section 3.3.

In our system, the final inference of fall or not fall is based on a threshold which is set as 0.3 in the generic model. With personalization, this threshold will vary between users and play a critical role in selecting the best model. Therefore, for a newly generated model, we first validate that model against test data with various thresholds until we can find a threshold that will give the precision better than the generic model or the existing model with a predetermined recall of 95%. If we cannot find a threshold that gives a better precision at 95% recall, this means the newly generated model is not better than the generic model or the prior one. Otherwise, the new model with the specified threshold is set as the best model for this particular user.

Figure 10 shows a box highlighting the best precision for a personalized model compared to the best precision of the generic model when the recall is fixed above .95. Here, the new personalized model is better and will be selected as the user's new model.

## 4. Evaluation

### 4.1. *Performance of Personal Model*

We first want to confirm that the personalized model from the automated pipeline is indeed better than the generic model. We used the SmartFall dataset[a] collected from 14 volunteers to train and test the generic model. This dataset is divided into 2/3 for training and 1/3 for testing. For personalization, we recruited two volunteers to wear the watch running our SmartFall for a period of time (45 mins to an hour) and performed a scripted set of activities of daily life. Whenever a fall is predicted, if it is a false positive, the volunteer will label that. All the labeled data will be used to train the personalized model at the end of this personalization period. We tested the two personalized models using the test dataset. Figure 10 shows that at 0.95 recall, both the two personalized models have above 0.9 precision versus the generic model whose precision is below 0.9 with the same recall.

[a]This dataset is available in: *http://www.cs.txstate.edu/ hn12/data/SmartFallDataSet*

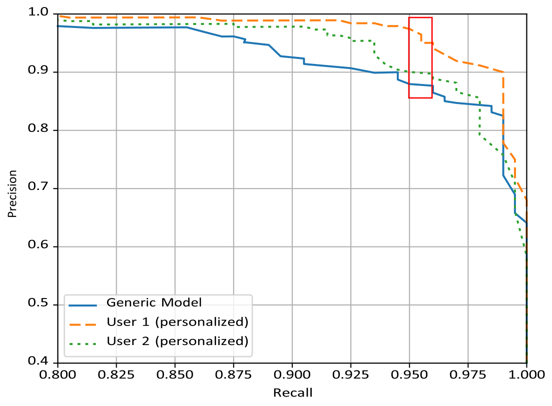


Figure 10: Performance of Generic vs Personalized model created using the automated pipeline. The box highlights where the system looks for the best threshold given that recall is fixed above .95. The precision achieved at this point is compared to the user's previous model to ensure the model has improved.

### 4.2. *Personalization pipeline*

Next, we want to confirm that the automated personalization pipeline as a whole can repeat the results of the previous manual method for personalizing fall detection models described in.[2] The manually generated model did not use a database to archive the collected data or store the personal test dataset for each user and data must be prepared manually for each user for re-training which is labor-intensive. Moreover, the manual system cannot handle multiple users' personalization at the same time. We validated both personalized models using the simulated fall prediction program on the test data set. Figure 11 shows the comparable Precision-Recall (PR) curves of both models.

As stated in Section 3.6.1, we used an additional metric called "spike score" to further validate the model generated via the automated pipeline. The spike score takes into consideration the number of acceleration "spikes" that occurred during a time period and uses those as a normalization factor for the false positive rate. Different users can have varying levels of activities and counting the absolute number of false positives generated is not accurate in deriving the accuracy of the model. We computed the spike score for the personalized model generated automatically. The spike score is 0.97 which is close to the personalized model generated manually that

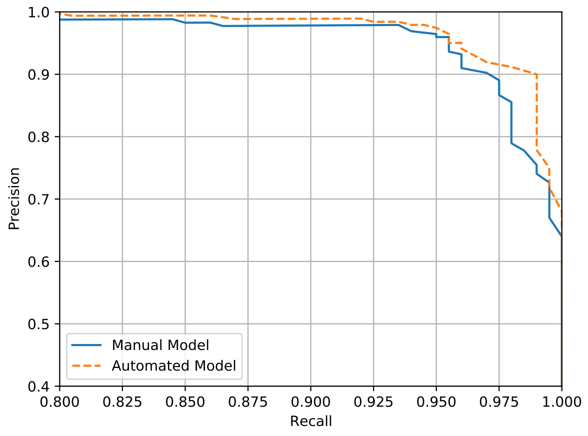


Figure 11: Automated model vs Manual model.

achieved a spike score of 0.98.

To ensure recall is still as good as the manually generated model, we asked the same two volunteers to perform 20 simulated falls on a mattress (five of each: back, front, left and right) and recorded the correctly detected (TP) and missed falls (FN), as well as possible false alarms (FP). The confirmed recall is around 90% for both models.

## 4.3. *How long does it take to personalize?*

We want to evaluate how many rounds of personalization are needed to derive a satisfactory model for a user. The following protocol was only tested on young and healthy users.

(1) The user is first told to wear the watch running SmartFall App with the generic model on their left arm wrist.
(2) The user will perform a set of prescribed activities for 30 minutes. This is the calibration phase.
(3) The user will keep track of false positives and provide feedback when prompted during the calibration phase.
(4) The user will press the "STOP" button to deactivate the SmartFall App at the end of the calibration phase.
(5) The recorded feedback data will be uploaded to the cloud's database automatically at the end of calibration phase.
(6) The user will perform the simulated fall test to record the recall of the model at the beginning and the end of the study.

In the cloud, the system will analyze the feedback data and compute the spike score every night. If the spike score is high (above .98), no new model will be generated. This means no personalization is needed. Otherwise, a new model is generated and validated as described in section 3.6.2.

After this initial round, we simply ask the same user to wear the watch for a few hours each day for five days and label the false positive, true positive, or false negative predictions if they pop up using the newly created personalized model. At the end of each night, the system will analyze the feedback data and create a new personalized model if there is sufficient labeled data and the spike score is below .98. If the newly created model validated to be better than the existing model, the watch will automatically download the new model and the associated threshold value the next time when the SmartFall App is activated. This process repeats for five days.

Table 1 shows the result of personalization for three different users over a period of five days. At the end of the five-day testing period, we found that User3 only requires one round, User2 requires two rounds, and User2 required four rounds to achieve a spike score of $\geq$0.97. All users performed simulated fall tests at the start and at the end of the personalization process to verify that the recall is retained. Table 2 shows the recall of the model before and after the personalization. For User1, the recall is 0.95 before personalization and it decreased to 0.85 on the fifth round of personalization. For User2, the recall is 0.85 before personalization and it is 0.7 after the personalization. For User3, the recall is 0.85 and decreased to 0.75. This shows that there is a definite tradeoff between recall and precision. The falls that are missed are mostly the right falls when the users are wearing the watch on their left wrists.

This experiment suggested that there is no fixed number of personalization rounds for every user. It is highly dependent on how the current model performs in relation to the kind of ADL activities performed. Our personalization is a continuous process and the goal is to always have the best model for each user. The experiment also shows that the number of FP has decreased significantly on Day 5 due to personalization.

Table 1: Performance of the model with continuous personalization.

| | | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|---|
| User1 | Hours worn | 0.87 | 2.99 | 2.83 | 2.72 | 3.65 |
| | # of FP | 27 | 20 | 5 | 9 | 12 |
| | # of spikes | 314 | 922 | 1099 | 953 | 1863 |
| | Spike Score | 0.91 | 0.97 | 0.99 | 0.99 | 0.99 |
| User2 | Hours worn | 1.15 | 0.72 | 2.95 | 2.06 | 2.08 |
| | # of FP | 109 | 16 | 14 | 36 | 8 |
| | # of Spikes | 779 | 343 | 2753 | 1295 | 581 |
| | Spike score | 0.86 | 0.95 | 0.97 | 0.97 | 0.98 |
| User3 | Hours worn | 0.65 | 2.3 | 1.9 | 2.2 | 2.1 |
| | # of FP | 50 | 5 | 9 | 14 | 18 |
| | # of spikes | 674 | 693 | 1607 | 2718 | 2667 |
| | Spike score | 0.92 | 0.99 | 0.99 | 0.99 | 0.99 |

Table 2: Comparison of recall before and after personalization.

| | User1 | User2 | User3 | Avg. |
|---|---|---|---|---|
| Generic model | 0.95 | 0.85 | 0.85 | 0.88 |
| Personalization | 0.85 | 0.7 | 0.75 | 0.76 |

Table 3: Performance of watch's battery when SmartFall is running with WiFi off.

| Hour | Test1 | Test2 | Test3 | Test4 | Test5 | Avg. |
|---|---|---|---|---|---|---|
| 1 | 80 | 80 | 79 | 79 | 80 | 79.5 |
| 2 | 58 | 59 | 59 | 58 | 59 | 58.6 |
| 3 | 39 | 35 | 39 | 39 | 38 | 38 |
| 4 | 19 | 15 | 19 | 19 | 17 | 17.8 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 |

### 4.4. *Battery Performance*

We tested the performance of the battery by recording the average time the battery lasted from a full charge until the watch runs out of battery when SmartFall App is running. The user is not using any other App on the watch during this battery testing period. As shown in Table 3 which records the battery performance, at the end of the first hour, the percentage of battery left was around 80%. At the fifth hour, the battery runs out before the end of that hour. In summary, the watch lasted an average of five hours (taken from five test runs where the watch was worn continuously from 100 percent to 0 percent). However, if the wifi on the watch is left on, the battery on the watch lasted only around 3.5 hours with the SmartFall App running. Note that our experiment is run on a five-year old Huawei smartwatch. When the App runs in background mode, further battery power can be conserved. For the system to be of practical value, we would like the battery to last for 12 hours or more. We believe as more advanced smartwatches come to market with better battery technology, this problem can be solved.

### 4.5. *Scalability of the pipeline*

We tested the system's ability to generate personalized models for multiple users. We tested this by uploading feedback data from 21 different users for personalization at the same time. This means the system must complete the re-training of 21 models overnight. We confirmed that new models were generated for all of the 21 users the next day. It took 11.18 hours to complete the personalization process for 21 users' models. All models were trained on on a Dell Precision 7820 Tower, 256 GB RAM with one GPU (GeForce GTX 1080). In summary, the system can process 21 personalization requests within an 11-hour period. With an increase in the number of GPUs to five, our system can scale to a hundred users. This is sufficient for a mid-sized nursing home which is where we want to deploy the system first.

## 5. Usability of SmartFall App

The watch-based SmartFall App has recently been used by 9 participants who were recruited under IRB approved protocol 7846 at Texas State University. Each participant was asked to wear the watch for 7 continuous days. The participant was expected to wear the watch for around 3 hours when he/she was relatively active during each of the 7 days. Each participant completed 3 surveys during the 7-day period. Each participant was also asked to complete a post-study interview, conducted via zoom by a sociologist specializing in gerontology.

The first survey assessed baseline altitudes towards technology and participant fall risk. The second survey was to check the ability of the participant to use the SmartFall App independently and correctly after the initial training. The third survey was to gather the feedback on the usability of the SmartFall App after using it for a week, including information regarding the UI, the comfort of wearing the watch, efficacy of the fall detection, interest in using the App or recommending it to others. Each participant was paid a gift card of $25 for participating in this study.

Table 4 summarizes the profile of the participants. Four of them (44.4%) were men, and five (55.6%) were women. The mean age of the sample was 71.4 years, and only 2 of them (22.2%) had ever used a smartwatch. All except 1 reported of being at fall risk, with 4 (44.4%) having fallen within the last 12 months. A third (3, 33.3%) reported having experienced balance issues. Two thirds of the participants expressed concerns about falling, specifically while taking a bath or shower (66.7%), going up or down stairs (55.6%), walking on a slippery surface (88.9%), or on an uneven surface (88.9%). Nevertheless, most were independent, with few having problems with ADL's. All of the participants reported that they were very comfortable with using computers and reported no trouble with charging and putting on the watch each day. In fact, 7 (77.8%) of them used a computer daily. The majority of them (66.7%) stated that they had no concerns with having their vitals monitored by a smart device. Yet, a smaller number of participants (5, 55%) were willing to wear devices that could record videos such as a body camera.

All 9 participants reported feeling comfortable wearing the fall detection device, with as many as 5 (55.6%) of them confirming that the App was easy to use after the initial training. Given that only 2 out of the 9 participants had used smartwatches before, this result is very positive regarding the design of the App. Overall, one third (33.3%) of the participants indicated that they were satisfied with the App, and 44.4% would recommend it to a friend and felt that they would be able to use it after a fall.

Themes that emerged from post-study interviews concern: 1) learning to use the App, 2) convenience of wearing a watch, 3) false positives, 4) potential benefits of the App, 5) improving the App, and 6) developing an attitude about something new. First, participants talked about how they had to learn to use the App, but after some initial difficulty, they all found it easy to use. For example, one participant mentioned that there were a lot of instructions to read and understand first: "It was not easy to use in the beginning, but it got easier after that." Second, one of the advantages of the App seems to be related with to convenient it is to have it on a watch, which is not intrusive at all. One stated, "..,compared to the other ones that are in the market, like the one you wear around your neck, this is a lot more convenient." Third, all complained that there were too many false positives. Although the personalized fall detection model has reduced false positives, participants reported that the false positive rate needed to be reduced further for the App to be useful for their daily life. Fourth, one participant stressed the need of a fall detection device like this App, especially for those living alone with the fear of falling. Another interviewee stated, "Someone will come to help you if you are unconscious." Fifth, participants offered various recommendations for improving the App, such as reducing false positives, making it simpler to use, making the screen more sensitive to touches, reordering the questions to ask "Have you fallen?" first, and repeating the question about the fall: "Did you say you have fallen?" Five of the nine participants said that the initial prompt of "Are you OK?" was confusing. The App should ask whether they have fallen or not. If the answer is "No," it should go back to the home screen. In addition, 2 participants said that the vibration sound of the watch alerting them of the prediction was too soft to be heard. This suggests the need to choose a smartwatch that can take into consideration of diminished hearing abilities of older adults. Finally, one participant brought up the importance of priming users of the App by helping them develop a good attitude toward something new such as a new technology or device.

The relatively low endorsement of the App indicates that there remains plenty of room for im-

Table 4: Background of Participants.

| | Race | Gender | Age | Fall Risk | Use of Smartwatch |
|---|---|---|---|---|---|
| P1 | White | M | 44 | Yes | No |
| P2 | White | M | 54 | Yes | Yes |
| P3 | White | F | 89 | Yes | No |
| P4 | White | F | 76 | Yes | No |
| P5 | White | M | 79 | Yes | No |
| P6 | Black | F | 77 | Yes | No |
| P7 | White | F | 76 | Yes | Yes |
| P8 | Asian | M | 78 | No | No |
| P9 | White | F | 71 | Yes | No |

provement. This is a research prototype, and there are hardware issues. For example, the watch screen is sometimes insensitive, and the vibration tone can be too low for some users with diminished hearing abilities.

The total activity data collected from the 9 participants amounted to 495MB with each participant wearing the watch for 3 hours each day for 7 days. If we scale that to 12 hours each day, this will generate around 2Gb for nine users for the same group of users. These data are not yet compressed. This suggests that the system only requires a moderate amount of storage. We conclude that the storage required for continuously monitoring of fall in a mid-sized nursing home is manageable.

In summary, this study with real-world participants demonstrated that the SmartFall App was valued and usable by the targeted population group of older adults with fall risk. Older adult participants did not feel the device was intrusive. More research with a larger and more diverse sample is, however, needed for optimizing the personalized fall detection algorithm to further reduce false positives for the SmartFall App to be acceptable by the target population.

## 6. Discussion

Our work demonstrates the feasibility of running a personalized real-time fall detection application on a commodity-based wearable device. A robust automatic personalization process can be achieved using an edge-cloud collaborative framework where the computational intensive re-training of a new model can be done on the cloud and the real-time detection can be performed on the edge device Management of each user's model, feedback data, and personal test dataset is achieved by using a NoSQL Couchbase database which is scalable for many users and where data can be versioned and exported as CSV files for re-training.

The real-time requirement of our prediction is satisfied despite the need to smooth over 20 predictions because the model we used is a simple LSTM model with two dense layers. The prediction is performed on the watch where the data is sensed, there is no latency being incurred due to data transmission. Data is sampled at 32 ms, and a total of 54 unique samples are used for the 20 predictions. That amounts to 32*54 = 1728 ms. So, in the worst case a fall will be detected 1.7 seconds after it has happened. However, often the inference average probability of the last 20 will exceed the threshold before the entire alpha queue is updated. Thus, a fall will be detected in less than 1.7 seconds.

We tested the personalization pipeline with three young and healthy users over a total of 31.17 hours and with all three users (two females and one male) able to obtain a model that achieve an average spike score of 0.99 with 0.76 recall.

We demonstrated the feasibility and usability of the system for the target population by recruiting 9 older adults to wear the watch Although the majority of the participants have never used a smartwatch before, all of them including an 89-year-old participant reported that they were comfortable in using SmartFall. We are able to collect a watch-based dataset of ADL data from these 9 older adults by just asking them to wear the watch for 3 hours each day over a seven-day period. This dataset will be used for further fine-tuning of future fall detection algorithms.

The most significant drawback of the personalization strategy, when scaling up for multiple users, is the long training time involved in creating and validating a new model (around 30 minutes for each user), although such training can take place offline, on a server. One of our future directions is to explore more efficient re-training strategies such as more accurate incremental re-training or transfer learning and more efficient validation, and also look into less computationally intensive network architectures such as a combination of one-dimensional convolutional layers (1D-CNN) and transformers for less battery consumption.

Our model is trained on data collected from 14 subjects. This is still considered a small dataset. One of our future work is to explore the use of a data augmentation technique based on Generative Adversarial Networks (GAN) to augment our dataset. Initial exploration of this data augmentation technique is published in.[21]

The usability study indicates that user satisfaction was relatively low. To improve user satisfaction and consequently the probability of long-term adoption, the current system needs to be enhanced in several aspects: (1) reducing false positives, (2) making it simpler to use, (3) increasing the sensitivity of the screen to touches, (4) reordering the questions, and (5) increasing the vibration sound of the watch. Some of those issues have already been addressed in our second prototype which will be trialed in early 2023.

## 7. Conclusion

This work paves the way for creating a fall detection system that can be tailored to each person. The infrastructure for collecting and labeling data is reliable and secure while preserving user privacy. The personalization process requires no intervention only requires a brief period of calibration. The system is set up to provide a robust way of deploying personalized fall detection models with the ability to immediately review performance and generate new models without the need for experienced programmers.

## Bibliography

1. T. Theodoridis, V. Solachidis, N. Vretos and P. Daras, Human fall detection from acceleration measurements using a recurrent neural network, *International Conference on Biomedical and Health Informatics*, Springer, (Greece, 2017), pp. 145–149.
2. A. H. Ngu, V. Metsis, S. Coyne, B. Chung, R. Pai and J. Chang, Personalized fall detection system, *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, (Austin, 2020), pp. 1–7.
3. M. A. I. Hubail, A. Alsuliman, M. Blow, M. Carey, D. Lychagin, I. Maxon and T. Westmann, Couchbase analytics: NoETL for scalable NoSQL data analysis, *Proceedings of the VLDB Endowment* **12**(12) (2019) 2275–2286.
4. A. Gigantesco, A. Ramachandran and A. Karuppiah, A Survey on Recent Advances in Wearable Fall Detection Systems, *BioMed Research International* **2020** (2020).
5. F. Riquelme, C. Espinoza, T. Rodenas, J.-G. Minonzio and C. Taramasco, eHomeSeniors Dataset: An Infrared Thermal Sensor Dataset for Automatic Fall Detection Research., *Sensors (Basel, Switzerland)* **19**(20) (2019).
6. M. C. Shastry, M. Asgari, E. A. Wan, J. Leitschuh, N. Preiser, J. Folsom, J. Condon, M. Cameron and P. G. Jacobs, Context-aware fall detection using inertial sensors and time-of-flight transceivers, *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, IEEE, (USA, 2016), pp. 570–573.
7. G. Demiris, S. Chaudhuri and H. J. Thompson, Older Adults' Experience with a Novel Fall Detection Device., *TELEMEDICINE AND E-HEALTH* **22**(9) (2018) 726–732.
8. A. Fanca, A. Puscasiu, D.-I. Gota and H. Valean, Methods to minimize false detection in accidental fall warning systems, *2019 23rd International Conference on System Theory, Control and Computing (ICSTCC)*, IEEE, (Romania, 2019), pp. 851–855.
9. V. Mirchevska, M. Luštrek and M. Gams, Combining domain knowledge and machine learning for robust fall detection., *Expert Systems* **31**(2) (2014) 163–175.
10. I. Chandra, N. Sivakumar, C. B. Gokulnath and P. Parthasarathy, IoT based fall detection and ambient assisted system for the elderly., *Cluster Computing: The Journal of Networks, Software Tools and Applications* **22**(Suppl 1) (2019) p. 2517.
11. P. Van Thanh, D.-T. Tran, D.-C. Nguyen, N. D. Anh, D. N. Dinh, S. El-Rabaie and K. Sandrasegaran, Development of a real-time, simple and high-accuracy fall detection system for elderly using 3-DOF accelerometers, *Arabian Journal for Science and Engineering* **44**(4) (2019) 3329–3342.
12. J. R. Villar, E. de la Cal, M. Fañez, V. M. González and J. Sedano, User-centered fall detection using supervised, on-line learning and transfer learning., *Progress in Artificial Intelligence* **8**(4) (2019) p. 453.
13. P. Tsinganos and A. Skodras, A smartphone-based fall detection system for the elderly, *Proceedings of the 10th International Symposium on Image and Signal Processing and Analysis*, (Slovenia, 2017), pp. 53–58.
14. K. Arulkumaran, M. P. Deisenroth, M. Brundage and A. A. Bharath, Deep reinforcement learning: A brief survey, *IEEE Signal Processing Magazine* **34**(6) (2017) 26–38.
15. A. Rosenfeld and J. K. Tsotsos, Incremental learning through deep adaptation, *IEEE transactions on pattern analysis and machine intelligence* **42**(3) (2018) 651–663.
16. T. R. Mauldin, M. E. Canby, V. Metsis, A. H. H. Ngu and C. C. Rivera, SmartFall: A Smartwatch-Based Fall Detection System Using Deep Learning, *Sensors* **18**(10) (2018).
17. H. M. Salman, W. F. W. Ahmad and S. Sulaiman, Usability Evaluation of the Smartphone User Interface in Supporting Elderly Users From Experts' Perspective, *IEEE Access* **6** (2018) 22578–22591.
18. N. A. Hee, W. Yeahuay, Z. Habil, P. Andrew, Y. Brock and Y. Lina, Fall Detection Using SmartwatchSensor Data with Accessor Architecture, *International Conference for Smart Health (ICSH)*, (China, 2017), pp. 81–93.
19. T. Mauldin, A. H. Ngu, V. Metsis, M. E. Canby and J. Tesic, Experimentation and Analysis of Ensemble Deep Learning in IoT Applications, *Open Journal of Internet Of Things (OJIOT)* **5**(1) (2019) 133–149.
20. Couchbase Server Architecture *https://www.youtube.com/watch?v=bgFyBEgTBc4* .
21. *X. Li, V. Metsis, H. Wang and A. H. H. Ngu, Ttsgan: A transformer-based time-series generative adversarial network,* arXiv preprint arXiv:2202.02691 *(2022).*