Collaborative Edge-Cloud Computing for Personalized Fall Detection *

Anne H. Ngu, Shaun Coyne, Priyanka Srinivas, and Vangelis Metsis

Texas State University, San Marcos, TX 78666, USA
{angu, spc51, p_s231, vmetsis}@txstate.edu

Abstract. The use of smartwatches as devices for tracking one's health and well-being is becoming a common practice. 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 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.

Keywords: Fall detection · Smart health · Model personalization · Deep learning · Edge computing · Wearables.

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's "hard fall" detection [1] or even more advanced deep learning models [12] have proven to be ineffective at covering all patterns of falls and ADL (Activity of Daily Living) data. Our previous work [5], 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 full user interface (UI) on the phone, with the watch used mainly for sensing of accelerometer data. This is problematic because elderly people have difficulties keeping up with devices that are not directly attached to them and may find

* This work is supported by the National Science Foundation under the awards CNS-1358939, CCF-1659807, CNS-1757893, at Texas State University, and the infrastructure was provided by the NSF-CRI 1305302 award.

phones difficult to retrieve from their pockets/purses after a fall. A watch's UI, on the other hand, would allow interaction with the SmartFall App at any time and anywhere.

The second issue is that creating a personalized model in the previous system 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 and leaves room for human error.

We aim to solve the above problems by automating the entire personalization process using a collaborative edge-cloud framework, from the user initially wearing the device/watch, to getting feedback or labeled ADL 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 personalized training dataset, and the validation and selection of the new model.

In this paper, we propose a solution that involves migrating the SmartFall App (UI and the prediction logic) to a single device (smartwatch) and using a robust and efficient Couchbase [6] 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 user's feedback data reliability including tracking the best personalized fall detection model for each user, the personalized training dataset for each user, and 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 on a commodity-based smartwatch. We describe how we can robustly collect labeled feedback data from the user in real-time, the automation of the pipeline, and the intuitiveness of the App's user interface on the watch. 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.
- An automated personalization pipeline including strategies used for re-training, and accurate offline validation of new models.
- Robust archiving of feedback data using Couchbase, a NoSQL database.
- The edge-cloud collaborative framework that enables optimization of limited resources of the watch, while achieving a robust fall detection performance.

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 young, healthy adult participants.

There has been a wide range of success levels, 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 [9] to location monitoring [11] all require wearable custom hardware [4]. It is not reasonable to set up 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 wearable device such as smartwatch which has unrestricted mobility. 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 the fall detection system is the high rate of false positives generated. A survey paper in [3], described the various strategies used to help combat false positives. However, this remains a challenging issue. This is partially due to difficulties in obtaining a 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 [3]. 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 [2, 8]. 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 movement continues to occur. These cases could be things such as Parkinson's, seizures, injury, or any instance in which the user is conscious 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. Solutions utilized some form of a generic model that was trained on a synthetic fall dataset from wrist-worn devices. The system in [14], 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 were 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. This personalized bag of words strategy 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 were 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. The authors in [13] also studied personalization of fall detection models using a traditional K-Nearest Neighbor (KNN) machine learning algorithm. 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%. Their data was collected using a smartphone while ours was via a smartwatch, which poses extra computational and user interaction challenges.

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 an 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.

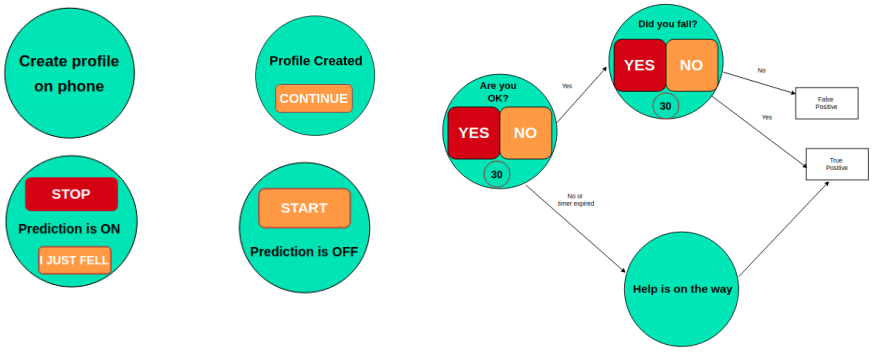
3 SmartFall with Edge-Cloud Collaboration

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. People can wear the watch and use it as a fall detection device without worrying about their mobility and privacy.

3.1 Overview of the personalization process

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 list of ADL activities. 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 1b.

The labeled feedback data is stored locally on the watch and is uploaded periodically to the cloud storage if consent to upload is available from the user. If the user did not give consent, only the generic model trained with the combination of aggregated ADL data from multiple users is used on the watch. On the cloud, during the night or when a certain number of false alarms have been generated, re-training of the model is initiated, and a new model is created. This model is validated and automatically pushed onto the watch if it is deemed to be a better model. An overview of the SmartFall system with personalization is shown in Figure 2. Our system is structured such that all user-identifying data are only stored locally on the watch to preserve privacy. The real-time fall prediction is performed on the watch. The training and personalization of the prediction



(a) Various UI smartwatch screens. (b) User interface display when a fall is detected.

Fig. 1: Smartwarch screens displayed at different states of the fall detection App.

model is done offline in the cloud server. All archived data are de-identified and indexed by a randomly generated key which is only known to the watch wearer and the caregiver. Archived data are configured to expire periodically to manage the finite cloud storage space.

3.2 Watch-based Fall Detection App (SmartFall)

Fig. 1 shows a few of the screens displayed on the smartwatch fall detection application. As part of the activation process of the SmartFall App, the user needs to create a profile on the phone. After the watch and the phone are 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" as shown in Figure 1a. After the profile is created, it is pushed to the watch and the watch will respond with "Profile Created". When the user presses the "continue" button on this UI, the SmartFall App is activated. At any time, if the user pressed the "stop" button in Figure 1a, the SmartFall App will be deactivated.

When a fall is detected, the top left watch's UI screen in Figure 1b is displayed. It is designed to start with just two "yes" and "no" buttons, asking the user if they are okay. If the user pressed "yes", the next screen will ask the user to confirm whether it is a fall or not. If the user did not press either "yes" or "no" on this screen, after a specified period of time (e.g. 30 sec), an alert message will be sent automatically to the designated caregiver. If the user responded with "yes" on the second screen, the sensed data will be labeled and saved as true positive data. If the user responded with "no", the data will be saved as false positive. The "I just fell", button on Figure 1a should be pressed if the system missed the fall. This data sample will be saved as a false negative. We followed the best practices advocated in [10] for the design of the UI for the elderly.

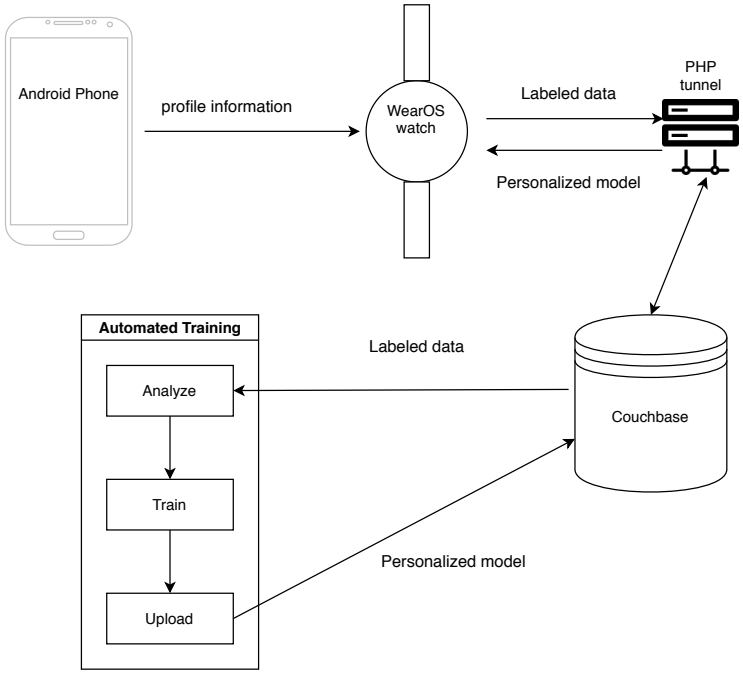


Fig. 2: Overview of the system and data flow between different components.

3.3 Real-time Fall Detection

The fall detection is made using a pre-trained recurrent deep learning model created in TensorFlow 2.0 as described in [7]. The fall prediction is made on a sliding window of data that is 35 samples (time steps) in length. This window size corresponds to a little over one second of accelerometer data which aligns with the duration of a typical fall. Each window is classified as positive (fall) or negative (not fall). 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.

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. If the averaged probability reaches a threshold, we determine that a fall has occurred. The threshold of 0.3 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.

If the system infers that a fall has occurred (threshold is > 0.3), we empty the queue by saving the data to the 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 is saved immediately as it could contain true positive fall data which is 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.

3.4 Data Archiving

Couchbase [6] is chosen as the storage system in both the watch and the cloud. Couchbase is an open-source NoSQL document-oriented database. Couchbase was chosen for its ability to scale up easily, fast I/O, and compact JSON format. All data are stored using Couchbase's document structure.

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 queue's data in the memory is saved to the 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 in temporal order, and each sample is unique. After obtaining the actual feedback from the user from the UI's prompt, the saved data is updated with the correct label.

The final type of accelerometer data that needs to be archived is 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 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 consists of only timestamp and user-id to facilitate the re-labeling process later.

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 are downloaded to the watch, newer tracker documents are created to track which set of feedback data was recorded with which model. Tracker document contains the *ID* of the first and last data sample recorded when using the model. It also contains the *model* 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. 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.

3.5 Database Synchronization

Synchronizing data collected on the watch to the cloud database is a core part of achieving automation in the personalization of fall detection. We upload 20 saved documents in batches periodically to avoid continuous usage of the watch's Wifi connection which can drain the battery. There is also a limit on the size of a file that can be uploaded over HTTP Post. Uploading in chunks of 20 documents avoids the data file being too large to be posted.

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 periodically, but never deleted from the watch. User's profile information is never uploaded to the cloud database to preserve the user's privacy.

All archived data is associated with a user-id (UUID) which is a 32-character string that is generated at random during profile creation. Each time the user starts the SmartFall App, the App queries the cloud's database for the best model using the UUID of the user. If the watch does not already have the best model, it is then downloaded to the watch. Fall detection then proceeds 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 if the system needs to train a new model. The second part of the system handles re-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.

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 is 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 retrain. Our evaluation metrics put emphasis on retaining high recall performance, i.e. our model should not miss a true fall and still have reasonable precision.

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 lifestyles (e.g. active subjects vs. less active ones), we leverage a new custom metric for measuring false positives for evaluation in our prior work [7]. This 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.

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 *Spike Score* value:

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

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 a large amount of low acceleration data if the user is not active. Adding too much low acceleration data to the training 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 is 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 data is 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 for this user. 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.

Part Two

Model Generation. Creating models in TensorFlow 2.0 using the TFS (Training From Scratch) method has been described in our previous work [5]. 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 testing dataset path is used to store the detail 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 that is set at 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 existing model with a pre-determined 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 3a shows a red box highlighting the best precision for the personalized and 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.

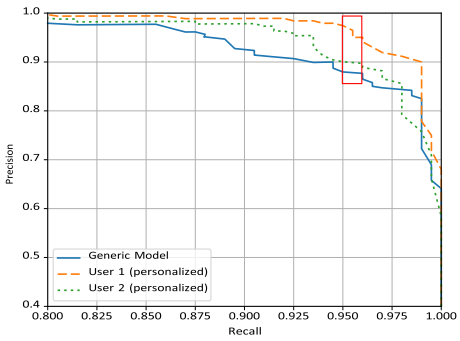
4 Evaluation

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[1] 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 ADLs. Whenever a fall is predicted, if it is false positive, the volunteer will label that. All the labeled data will be used to train the personalized

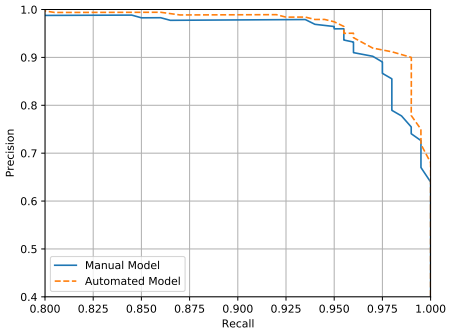
[1] This dataset is available from `http://www.cs.txstate.edu/~hn12/data/SmartFallDataSet` under the smartwatch folder.

model at the end of this personalization period. We tested the two personalized models using the test dataset. Figure 3a 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.

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 [5]. 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 3b shows the comparable Precision-Recall (PR) curves of both models.



(a) Performance of Generic vs Personalized model created using the automated pipeline.



(b) Automated model vs Manual model.

Fig. 3: Precision-Recall curves comparing the performance of different models at different thresholds.

As stated in Section 3.6, 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 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.

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 is used for the initial round of personalization:

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 performs a set of prescribed activities for half an hour. This is like a calibration phase.
3. The user provides feedback when prompted during the calibration phase.
4. The user performs the simulated fall test to record the recall of the generic model at the end of the calibration phase.
5. The user will press the "STOP" button to deactivate the SmartFall App at the end of the calibration phase. The recorded feedback data will be uploaded to the cloud's database automatically at this point.

In the cloud, the system analyzes the feedback data and computes the spike score. If the spike score is high (above .98), no new model is generated. This means no personalization is needed. Otherwise, a new model is generated and validated as described in section 3.6.

After this initial round, we 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 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, User1 requires two rounds, and User2 required four rounds to achieve a spike score of $\geq$0.97. All users performed simulated test falls at the start and 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 trade off 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

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

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.

Table 2: Comparison of recall before and after personalization, when the required spike score is set to 0.98.

	User1	User2	User3	Average
Recall with Generic model	0.95	0.85	0.85	0.88
Recall with Personalization	0.85	0.7	0.75	0.76

5 Conclusion

Our work demonstrates the feasibility of running a personalized real-time fall detection application on a commodity-based wearable device. 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 patient privacy. The personalization process requires no intervention from any programmer and only requires a brief period of calibration and willingness to wear the watch and activate the SmartFall App.

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 in the cloud and the real-time detection can be performed on the edge without loss in prediction accuracy and delay. Management of each user's model, feedback data and personal test data is achieved by using a NoSQL database which is scalable for many users and where data can be versioned.

The complete automation of the personalization process demonstrates the feasibility of collecting a dataset of accelerometer data from the users (e.g. elderly people) by just asking them to wear the watch for a period of time in hope of being able to generate real datasets for future fall detection algorithms.

References

1. Apple watch series 4. http://www.apple.com/apple-watch-series-4/activity/, accessed: 2019-04-18
2. Chandra, I., Sivakumar, N., Gokulnath, C.B., Parthasarathy, P.: Iot based fall detection and ambient assisted system for the elderly. Cluster Computing: The Journal of Networks, Software Tools and Applications **22**(Suppl 1), 2517 (2019)
3. Fanca, A., Puscasiu, A., Gota, D.I., Valean, H.: Methods to minimize false detection in accidental fall warning systems. 2019 23rd International Conference on System Theory, Control and Computing (ICSTCC), System Theory, Control and Computing (ICSTCC), 2019 23rd International Conference on pp. 851 – 855 (2019)
4. Gigantesco, A., Ramachandran, A., Karuppiah, A.: A survey on recent advances in wearable fall detection systems. BioMed Research International (2020)
5. H.Ngu, A., Metsis, V., Coyne, S., Chung, B., Pai, R., Chang, J.: Personalized fall detection system. In: The proceedings of the 5th IEEE PerCom Workshop on Pervasive Health Technologies, Austin, TX (March 2020)
6. Hubail, M.A., Alsuliman, A., Blow, M., Carey, M., Lychagin, D., Maxon, I., Westmann, T.: Couchbase analytics: Noetl for scalable nosql data analysis. Proceedings of the VLDB Endowment **12**(12), 2275–2286 (2019)
7. Mauldin, T., Ngu, A.H., Metsis, V., Canby, M.E.: Ensemble deep learning on wearables using small datasets. ACM Trans. Comput. Healthcare **2**(1) (Dec 2021). https://doi.org/10.1145/3428666, https://doi-org.libproxy.txstate.edu/10.1145/3428666
8. Mirchevska, V., Luštrek, M., Gams, M.: Combining domain knowledge and machine learning for robust fall detection. Expert Systems **31**(2), 163 – 175 (2014)
9. Riquelme, F., Espinoza, C., Rodenas, T., Minonzio, J.G., Taramasco, C.: ehomeseniors dataset: An infrared thermal sensor dataset for automatic fall detection research. Sensors (Basel, Switzerland) **19**(20) (2019)
10. Salman, H.M., Ahmad, W.F.W., Sulaiman, S.: Usability evaluation of the smartphone user interface in supporting elderly users from experts' perspective. IEEE Access **6**, 22578–22591 (2018)
11. Shastry, M.C., Asgari, M., Wan, E.A., Leitschuh, J., Preiser, N., Folsom, J., Condon, J., Cameron, M., Jacobs, P.G.: Context-aware fall detection using inertial sensors and time-of-flight transceivers. In: 2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). pp. 570–573. IEEE (2016)
12. Theodoridis, T., Solachidis, V., Vretos, N., Daras, P.: Human fall detection from acceleration measurements using a recurrent neural network. In: Precision Medicine Powered by pHealth and Connected Health, pp. 145–149. Springer (2018)
13. Tsinganos, P., Skodras, A.: A smartphone-based fall detection system for the elderly. In: Proceedings of the 10th International Symposium on Image and Signal Processing and Analysis. pp. 53–58 (Sep 2017)
14. Villar, J.R., de la Cal, E., Fañez, M., González, V.M., Sedano, J.: User-centered fall detection using supervised, on-line learning and transfer learning. Progress in Artificial Intelligence **8**(4), 453 (2019)