Adaptive Fall Detection on the Edge

Christopher Calloway
Electrical Engineering
Stanford University
cmc2374@stanford.edu

Estelle Kao
Electrical Engineering
Stanford University
ekao5@stanford.edu

***Abstract*— Falls represent a considerable medical danger to the world's population. Many fall detection systems have been implemented to rectify this issue, such as the Apple Watch fall detection, Life Alert, and various smartphone IMU applications. However, the models implemented in these devices are either inference-only or cloud-based. Given the diversity of lifestyles and living conditions in the world, a model that can retrain from data it collects on the edge could provide users with a more robust fall prediction system. This report presents an experimental device setup and data collection to test such an adaptive fall detection model on the edge. The device is initially trained on data from the MobiAct dataset and then retrained on the edge using data collected from an LSM330DLC Accelerometer/Gyroscope [1]. Our neural network-based device retrained on the edge achieved an accuracy of 71.0% and 80.0% on two activities we taught our model to label as a non-fall with only 1 and 3 retraining events respectively. We also achieved an accuracy 94.0% on one new activity we taught the model to consider a fall with 4 retraining events.**

I. Introduction

According to the CDC, over 3,000,000 Americans over 65 go to the emergency room and over 300,000 are hospitalized for fall-related injuries each year [2]. Wearable technology offers a potential mitigation strategy to this problem. The wearable technology market has been projected to grow at an annual growth rate of 14.9% from 2022 to 2030 [17]. With the continued growth of wearable devices, it is only natural that fall detection software would become an important feature of such devices. The data show that there is a great incentive for fall detection software to alert and contact medical assistance quickly and prevalent hardware to do so. However, the current state-of-the-art fall detection systems, such as the Apple Watch, use inference-only models [3]. Abou L, Fliflet A et al. found the Apple Watch had a false negative rate of 95.3% in an experiment with able-bodied people falling out of wheelchairs [3]. By allowing retraining on the edge, a more robust fall prediction system can be put in place with models fine-tuned to a user's particular lifestyle.

In this report, we discuss an adaptive fall detection device that will retrain according to user-defined behavior. We used an LSM330DLC Accelerometer and Gyroscope to measure acceleration, angular velocity, roll and pitch. We created a 3-layer feed-forward neural network trained on the MobiAct dataset with a fall detection accuracy of 99.76%. We then ran live inference on an Intel UP Board and we provide a button that can self-label collected fall data.

With this device, we measured three new activity types to demonstrate experimentally that the adaptive edge model provides greater performance than an out-of-the-box model. We performed three types of falls: a knee slide, a back fall and a slide off a chair. A knee slide is a common action professional skateboarders take while bailing from a trick. The technique requires the users to land on their knees, lean back, and slide forward. We observed our model is able to learn to distinguish this event as a non-fall motion with 71.0% accuracy with only 1 retrain. A backward fall is modeled on a person falling backward onto a bed. The fall requires falling backward from a standing position onto a cushion about 16 inches off the floor. We observed our model is able to learn to distinguish this event as a non-fall with 80.0% accuracy with 3 retrains. Lastly, we considered a user sliding off the edge of a chair. This is modeled on a user falling out of a wheelchair, which could be a very low-impact but critical fall to detect. This fall requires

a user to slide out of the chair and hit the floor at varying degrees of impact. We observed our model is able to learn to distinguish this event as a true fall with 94.0% accuracy with 4 retrains.

These 3 retrained models show an accuracy of 98.13%, 96.19%, and 98.75% respectively when applied against the original MobiAct dataset showing our model is correctly distinguishing the new activities without overfitting.

The rest of the report is organized into the following sections. Section 2 discusses related work, papers, and methodologies for fall detection. Section 3 discusses our design setup, including specifics of the dataset, preprocessing, model architecture, and physical design. Section 4 discusses our data collection strategy and the experiments we ran with our adaptive fall detection device. Section 5 analyzes the results and model accuracy for these experiments. Section 5 also discusses a detailed power analysis of the device and the performance of an alternative model architecture. Section 6 summarizes our work and includes a discussion of our future work. Section 7, the Appendix, includes a link to our source code for this project as well as a summary of each author's contribution to the report.

II. Related Work

Most fall detection literature focuses on the mobile deployment of fall detection models on an edge-cloud framework [8] [9]. There are two main ways to collect input data. The first method is to collect video feeds to keep track of the participants' movements [9]. However, this limits the mobility to constrained spaces where the cameras are installed. The second method uses sensor data to collect accelerometer, gyroscope, and magnetometer information. This is a more portable solution and can be deployed commercially, therefore we will focus on discussing research done in this area.

Most work in this sub-area utilizes smartphones to collect sensor data to make predictions on-device as to whether a fall has occurred. A simple but effective model for fall detection uses the signal magnitude vector (SMV) as a threshold for falls [18]. The SMV is the root mean square of the accelerometer data given by

$$SMV_i = \sqrt{{x_i}^2 + {y_i}^2 + {z_i}^2} \tag{1}$$

where x_i, y_i, z_i are the acceleration values along the x, y and z-axis for the accelerometer at a given discrete time i. Within the MobiFall dataset such a model was able to achieve a 55% fall detection accuracy [14]. He et al. used another threshold-based implementation that considers the signal magnitude area, SMA_i and the tilt angle TA_i as well [15].

$$SMA_i = \frac{1}{i}(\sum_{u=1}^{i}|x_u| + \sum_{u=1}^{i}|y_u| + \sum_{u=1}^{i}|z_u|) \tag{2}$$

$$TA_i = sin^{-1}(\frac{y_i}{\sqrt{x_i^2 + y_i^2 + z_i^2}}) \tag{3}$$

In this approach, He et al. defined a hyperparameter that Eq. 2 had to be above to be considered an activity and not noise. Likewise, if this condition was met, another hyperparameter was defined such that the SMV Eq. 1 had to be above and yet another hyperparameter was defined so that Eq. 3 had to be above it. If all three equations were greater than their respective hyperparameters, an activity was labeled as a fall. This approach achieved a 71% accuracy on the MobiFall dataset [14]. Some approaches use more advanced fall detection algorithms, but result to more simple thresholds to reduce the processing power and computation time when possible [9].

The risk of these threshold implementations occurs when a threshold that is too high could make it so that only "hard" falls with large g-force will be detected. The situation may be exacerbated when the device is worn on the wrist and the user does not use their hands to stop the fall [11].

Several machine learning implementations have been implemented that have superior performance to threshold-based methods on the the MobiAct dataset [1]. Some edge-cloud systems have an interface to allow users to label false positives and send raw data to a cloud server [16]. The cloud server provides model parameter updates then redeploys the model onto the mobile device (Fig. 1) [16]. Artificial intelligence and machine learning algorithms implemented, such as support vector machine, k-nearest nearest neighbor, naive bayes classification, logistic regression, decision tree, and neural networks have all achieved accuracies greater than 90% on various fall/ADL

datasets. [12]. In particular, the authors of the MobiAct dataset achieved accuracies of 99.88% and 99.30% for k-nearest neighbors and decision trees respectively when applied to the MobiAct dataset [1].

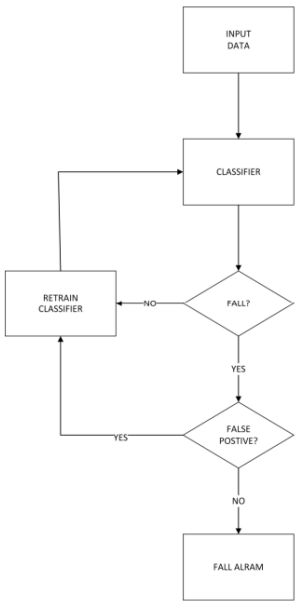

Fig. 1. Model retrain is triggered when a false positive is detected. [16]

While the k-nearest neighbors and decision trees algorithms achieved superb accuracies already, Zhu et al suggested that the accuracy may be further improved if using deep learning algorithms, such as CNN classifiers, instead of traditional ensemble classifiers [13]. Zhu et al created their own data with more advanced motions such as bicycling and swimming and showcased significant improvement using a CNN network instead of random forest and XGBoost. While an ensemble of CNN networks showed even better accuracy, the model size is also larger which may not be ideal for on-the-edge training and inferences. In our design, we used an MLP classifier and got an accuracy of 99.76% on the MobiAct dataset, which is on-par with the k-nearest neighbors and decision trees algorithms.

III. Design Setup

Our design was created to mimic the data collection established by the MobiAct dataset. In addition, we added extra hardware to retrain on the edge. An overview of our design can be seen in Fig. 6.

A. Dataset

The dataset we used is the second release of the MobiAct dataset (the first was called MobiFall) which contains 12 different types of activity of daily living (ADL) and 4 types of falls [4]. Table I and Table II list all these classes. The MobiAct dataset was collected from 66 participants with 3200+ trials and contains motions that involve sudden movements along z-axis changes, such as climbing the stairs, jumping, and jogging. To make the dataset more comprehensive and understand how we can utilize the MobiAct dataset to perform on-device training, we added three more types of fall. The falls were a knee slide, a backwards fall, and sliding off of a chair. The knee slide motion is common in skateboarding tricks, which we hope to designate as a non-fall, is most similar to the front-knee-lying fall position from MobiAct dataset. The backwards fall, which we also hope to designate as a non-fall, is most similar to the forward-lying position and is modeled on someone falling back into bed. The sliding out of a chair, which we intend to be trained as a fall event, is most similar to the back sitting chair fall and is modeled as a user sliding off the edge of a wheelchair.

In this report, we will demonstrate how our device is able to categorize the knee slide and backwards falls as a non-falling event after training while still successfully labeling the four MobiAct fall positions as a fall events. Likewise, we will demonstrate how our device is able to categorize the front edge chair fall as a true falling event without impacting our other fall/ADL category classification.

Both the MobiAct falls and the falls we collected were performed on hard mattresses of 5cm in thickness and captured on LSM330DLC inertial module which contains 3D accelerometer and gyroscope.

TABLE I

ACTIVITIES OF DAILY LIVING FROM MOBIACT

Label	Activity
STD	Standing
WAL	Walking
JOG	Jogging
JUM	Jumping
STU	Stairs up
STN	Stairs down
SCH	Stand to sit(sit on chair)
SIT	Sitting on chair
CHU	Sit to stand(chair up)
CSI	Car-step in
CSO	Car-step out
LYI	Lying

TABLE II

FALLS FROM MOBIACT

Label	Activity
BSC	Back-sitting-chair
SDL	Sideward-lying
FKL	Front-knees-lying
FOL	Forward-lying

TABLE III

FALLS FROM THIS WORK

Label	Activity
KSL	Knee-slide
BFL	Backwards-fall
FEC	Front-Edge-Chair

B. Data Preprocessing

The LSM330DLC has two sensors, an accelerometer, and a gyroscope, for a total of 6 DoF. The accelerometer measures acceleration with a range ±2g and a sensitivity: 0.061 mg/digit, where g= $9.80665 m/s^2$. The gyroscope sensor measures the angular rotation of the LSM330DLC body axes with a range of ±250 dps and a sensitivity of 8.75 mdps/digit. Roll and pitch are also indirectly measured from the accelerometer and gyroscope data.

Roll and pitch can't be directly calculated to a high degree of accuracy due to signal noise and gyroscope drift. To get a confident estimate we employ sensor fusion on the two measurements. Specifically, we created the following filter seen in Fig. 2.

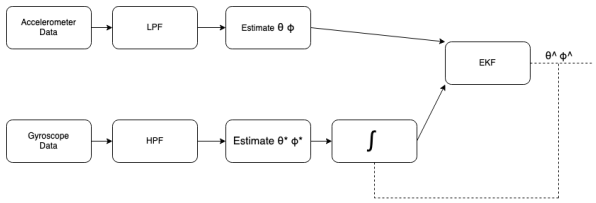

Fig. 2. UP Board and LSM330 I2C connection

A digital low-pass filter is applied to the accelerometer data, and then that data is used to calculate the first of two predictions for the roll and pitch. Likewise, a digital high pass filter is passed to the gyroscope data and it predicts the roll and pitch as a function of time. This is integrated with respect to the 6 second sampling window to give the gyroscope estimate of roll and pitch. To find the adequate weight of both filters both are passed to an extended Kalman filter. The final state vector of the extended Kalman filter is taken to be the roll and pitch data.

With the raw data of acceleration, angular rotation, roll, and pitch the following features are calculated:

1) Minimum Gyro
   (gyro_x_min, gyro_y_min, gyro_z_min)
2) Maximum Gyro
   (gyro_x_max, gyro_y_max, gyro_z_max)
3) Minimum Acceleration
   (x_min, y_min, z_min)
4) Maximum Acceleration
   (x_max, y_max, z_max)
5) Standard Deviation Acceleration
   (x_std, y_std, z_std)
6) Mean Acceleration
   (x_mean, y_mean, z_mean)
7) Slope Acceleration
   (x_slope, y_slope, z_slope)
8) Zero Crossing Acceleration
   (x_zc, y_zc, z_zc)
9) Maximum Mean Discrepancy Acceleration
   (x_mmd, y_mmd, z_mmd)
10) Slope Pitch/Roll
    (pitch_slope, roll_slope)

Where the features were chosen based on their

successful results in other ML models implemented in the MobiAct dataset [1]. These features are then all normalized. Finally, the normalized features are passed into the model for inference, and the results are then indicated over the device's 3 LEDs. The details for the latter two steps are discussed below.

C. Model Architecture

Our classification model was a binary-class MLP classifier. The model contains a total of 3 layers, where the hidden layer uses a ReLU (rectified linear unit) activation function and the output layer uses a sigmoid function. There are a total of 29 input nodes and 100 hidden nodes as demonstrated in Fig. 3. The model was created and trained using the Scikit-learn Python library. The source code was forked from an existing repository that provided a convenient framework for loading in the MobiAct dataset [19]. The model was able to converge quickly and shows a loss function of 3e-3 within 100 iterations with a model size of 80 kilobytes (Fig. 4).

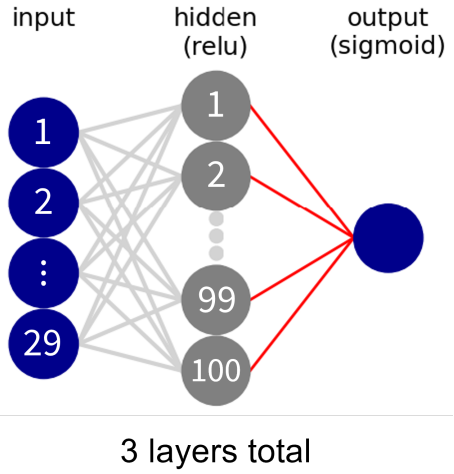

Fig. 3. Neural network architecture used in the model.

This small 3-layer network was chosen for a number of reasons. From Section 2, we observed that simple analytical models could achieve accuracies as high as 71%. Furthermore, very basic machine learning algorithms achieved accuracies above 90%. A neural network approximates the optimal function, where more layers allow for greater approximation capability. However, in our resource-constrained embedded system, we want to minimize the model size, and hence minimize the number of layers. Given the strong performance of simple models, we chose a 1 hidden layer design as an optimized tradeoff between size and performance. In section 5 we will defend this choice by comparing the performance of this model against a 5-layer (3 hidden) network. Additionally, we avoid convolutions to maximize the connections for potential retraining.

To retrain the model on the device, we use sklearn's partial_fit function which performs one epoch of stochastic descent on all data collected in a certain window. This window is defined as the period of either five self-labeled readings or sufficient variability in the currently observed data. Variability is measured as follows.

$$similarity_k = \arg\min_j \frac{\sum_i A_k \sum_i B_{j,i}}{||A_k||||B_j||} \quad (4)$$

$$variability = \sum_i similarity_i \quad (5)$$

Where A_k is the vector of features indicated in section 3, and B_j is a representative vector of features corresponding to a class from the ADLs and Falls in Table I and Table II respectively. We hold a FIFO of self-labeled and non-labeled vectors of features A. For each A_k, we match the most similar vector B_j using cosine similarity, as shown in Eq. 4. We sum up all such similarities and if it thresholds above a certain hyperparameter α we force a retrain event.

Once the dataset is fitted into the model, we clear the FIFO A and wait until at least another five self-labeled data are seen or sufficient variability to trigger the retraining of the model.

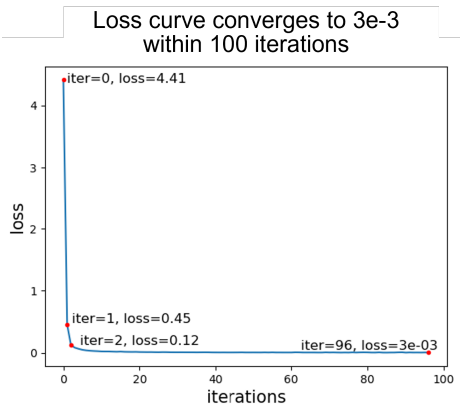

Fig. 4. Loss curve of the vanilla 3-layer neural network trained offline.

D. Physical Design

The LSM330DLC chip we used is packaged as part of a printed circuit board with convenient I2C connection terminals. The LSM330DLC accelerometer/gyroscope communicates with the UP Board over an I2C bus. This is shown in Fig. 5. The UP Board takes the raw data and programmatically applies the preprocessing discussed in Subsection B. The data is passed to the model discussed in Subsection C where inference is done. The result of the inference is displayed on the 3 LEDs.

The green LED indicates no fall detected. The red LED indicates a fall was detected. The yellow indicates that retraining has been initiated. A push button was used for self-labeled data. The LEDs and the pushbutton are controlled by a Raspberry Pi Pico and then sent over a serial bus to the UP Board. This allows the button and LEDs to operate completely in parallel to the fall detection script. All these electrical components were stored tightly in a cardboard box with the LEDs and push button exposed. A belt was strapped through to allow a person to wear the device. The resulting device is shown in Fig. 6.

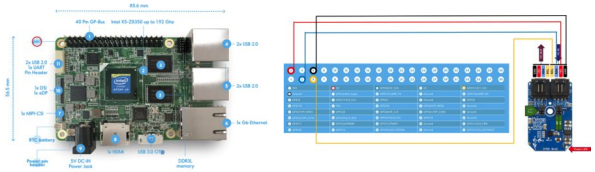

Fig. 5. Sensor Fusion Filter of Gyroscope and Accelerometer

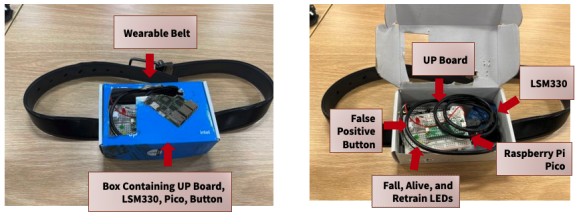

Fig. 6. UP Board Physical Design

IV. Data Collection and Experiments

To understand how on-device training can affect model adaptation, we simulated two fall-like positions, and one non-fall-like position. In a knee slide position, the participant accelerated forward and glided with the knee in contact with the ground with the center of weight is toward the heels (Fig. 7. In the backwards fall position, the participant falls on their hips with all limbs and the back flat on the raised cushion at the end of the fall (Fig. 8. In the front edge chair case, the participant slides off the front edge of the chair and makes contact with the floor with relatively low impact (Fig. 9).

For the fall-like positions, we used two sets of our hardware setup strapped onto the midsection area around the abdomen. The top is with model retrain enabled and the bottom is without. Every 5ms, the device collected accelerometer and gyroscope data. In 6s chunks, the device analyzed the features listed in Section 3 in the 6s time frame. Based on these calculations, the model made predictions as to whether the event was a fall. If a fall was predicted the red LED of the board turned on. If this was considered a false positive by the user, the button was pressed to mark the event as a non-fall (false positive).

The hardware with retrain enabled collects the post-processed calculations in a queue. When there are five self-labeled data collected or sufficient variability in the queue, the model retrains and clears the queue. Note that because the two hardware implementations are sitting on top of each other, the raw accelerometer and gyroscope readings are not identical. Therefore, we see that the two may provide different fall predictions using the same model.

For the non-fall-like positions, greater precision was needed due to the low-impact nature of the fall. For this reason, a single UP Board was used that ran inference on two models simultaneously.

As was done for the fall-like positions, the device analyzed the features listed in section 3 in the 6s time frame. Based on these calculations, the model makes predictions as to whether the event was a fall. However, instead of using the button to mark true positive falls after a non-fall has been detected, we preemptively press the button to indicate that the next activity will be a fall. The reasoning behind this is that we cannot expect an end user to self-label a true fall. If an end user truly falls it is unlikely they will think to press a self-labeling button. The intention of the fall labeling design is so that a professional could do these types of falls ahead of time to create a model for the end user. For instance, in the case of a wheelchair user, a professional would collect wheelchair fall data with this method and retrain the model as opposed to letting the wheelchair user do this for themselves at high health risks.

Fig. 7. Knee Slide Fall position

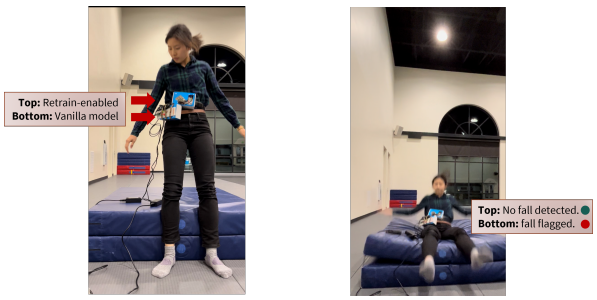

Fig. 8. Backfall position

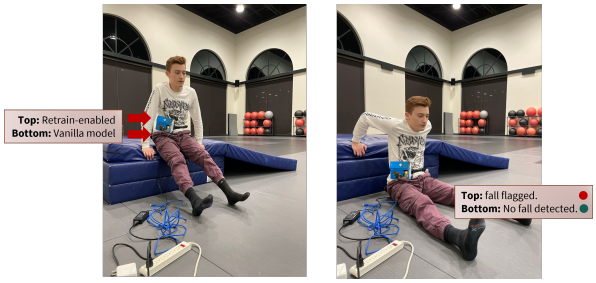

Fig. 9. Front-Edge-Chair position

V. Results and Analysis

A. Model Accuracy

For the fall-like positions, in the very first iteration in which the model has not retrained yet, both hardware flagged the fall-like motion as a fall event. Here, we call the number of retrains as the epoch iteration. Note this is different from the standard definition of epochs in machine learning models. Fig. 10 shows that in a knee slide experiment, the very first iteration (Epoch 0) shows that all five knee slides were flagged as falls in both systems. After the model is retrained, we see that 10 out of 14 knee slides were no longer classified as falls. Similarly, backward falls were retrained successfully by Epoch 3. Epoch 0 should produce same results on both hardware sets, however the position of where the hardware is strapped affected the raw data collection; the two hardware sets were positioned on top of each other. Therefore, we saw a deviation from the predicted results although both were using the same model at Epoch 0. Fig. 11 shows how the retrained model was able to classify the knee slide and back falls as a non-falling event after one retrains and three retrains respectively.

As for the front-edge-chair position, we collected data after each retraining is triggered. For the 1-hidden layer neural network, the model achieved an accuracy of 94% after four retrains. In addition to the 1 layer model, we also implemented a 3-hidden layer model to see if model size had an effect on our retraining ability (the specifics of this procedure are discussed in the alternate design analysis subsection). We collected the data in the same manner as the 1-hidden layer network just with a different model loaded into the script. The 3-hidden layer neural network converged much quicker and was able to achieve 100% accuracy

after two retrains. Fig. 13 illustrates that if a wheelchair user were to use the vanilla model without retrain, the accuracy will hover around 60% with more than 90% of it being false negatives where a fall has occurred but not detected. From the comparison of the 1-hidden layer and 3-hidden layer model, we are confident that the size of the neural network should not affect the model accuracy. However, smaller networks will take more iterations to train with the benefits of lower latency and power consumption when making predictions and model updates on device. The model accuracy 1-hidden layer network for the three fall positions is summarized in Table IV

We observed that the greater number of retraining events there were, the better the end accuracies were. While this result is intuitively obvious, it poses a challenge to improving models with false positive labeling. Namely, after a retrain event, the script requires another 5 false events in order to trigger a retrain. However, for a model that has already been retrained it can be challenging to reach this threshold. For instance, in the case of the knee slide after 1 retrain the model was correctly predicting most but not all, knee slides correctly. It is difficult to get enough data to trigger another retrain event. This is partially mitigated by using feature vector similarity as another condition for retraining.

We also tried cross-validating the models with the rest of the dataset. We loaded the pre-trained knee slide model and performed backfalls. Because the knee slide model was a medium-impact event, the model still flagged the high-impact back falls as falls (Fig. 12). When we loaded the backfall pre-trained model, the knee slide being a lower impact event was no longer flagged as a fall.

To ensure we are not overfitting the model to backfalls, kneeslides, and chair edge slides, we then used the three models to predict the results of the MobiAct dataset. Although there was a slight degradation on model accuracy, the pre-treained knee slide model, back fall model and front-edge chair model still achieved 98.13%, 96.19% and 98.75% accuracy, respectively (Table. V).

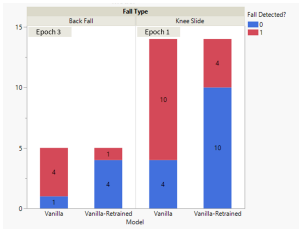

Fig. 11. Fall detection on last epoch of on-chip training data we collected.

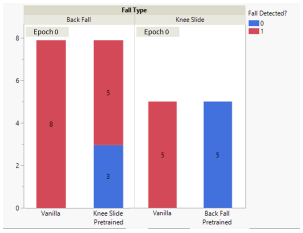

Fig. 12. Cross validation of the pre-trained models.

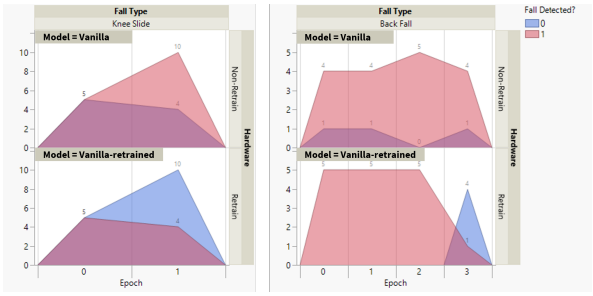

Fig. 10. Fall detection results with model retrain disabled and enabled.

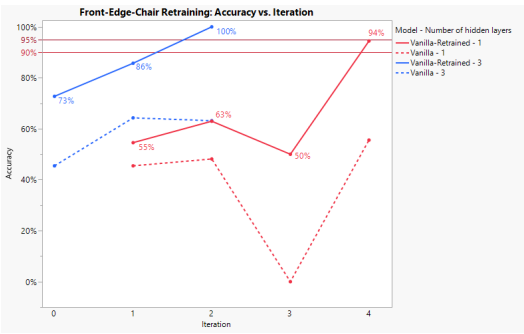

Fig. 13. Model retrain is triggered when a false positive is detected. [16]

TABLE IV
MODEL ACCURACY AGAINST ACTIONS PERFORMED
FOR A 1-HIDDEN LAYER NETWORK

Model	Vanilla	Vanilla-Retrained
Back Fall*		
Number of retrains	0	3
Accuracy	20%	80%
Knee Slide*		
Number of retrains	0	1
Accuracy	28%	71%
Front-Edge-Chair**		
Number of retrains	0	4
Accuracy	60%	94%

*Back Fall and Knee Slide consists the intended fall actions only.
**Front-Edge-Chair action consists of 50% wiggling in the chair and 50% falling lightly out of the chair.

TABLE V
MODEL ACCURACY AGAINST MOBIACT DATASET

Model	Vanilla	Knee Slide	Back Fall	Chair Edge Fall
Accuracy	99.59%	98.13%	96.19%	98.75%

Fig. 14. UP Board Physical Design

Fig. 15. UP Board Physical Design

B. Power Analysis

Power consumption analysis is critical to edge device deployment, in particular for a device that is meant to assist users in case of emergency. The state-of-the-art standard Apple watch model has a battery life of 18 hours [5]. The UP Board device does not have a way to directly measure power consumption. To model this, we rely on the datasheet provided for the UP Board by Intel. Intel claims that the device has a 13 watts power consumption under typical conditions [6]. Typical conditions are not defined by the datasheet. To approximate it, we compare Fig. 14 and Fig. 15 which was generated by the tool TUI.

The figures show the core temperature, average core frequency, and average core utilization for the UP Board with and without the fall prediction program running, respectively. We can see that the UP board running the program has a higher average core frequency and utilization compared to the idle UP Board. However, this deviation is not considerable and leads us to believe that the fall prediction script running can be considered as typical system conditions. Thus we approximate a power consumption of 13 watts.

With a 5V, 20000mAh power bank, this is a 2.6A drawing current. Using 3A as a conservative estimate, with this power bank we would have a battery life of 5 hours. This is unacceptable for a real deployment. However, in Section 3 we showed our model architecture was a fairly simple 3 layer neural network. It stands to reason that a board much less complicated than the UP Board could run this. Indeed we implemented a reduced scale version on a Raspberry Pi Pico, which only draws 0.4325 watts. With a 3V supply, this is a battery life of 138 hours. The problem with the Pico is that it only has 80 kilobytes of storage, a limitation we will further discuss in the next section.

C. Alternative Design Analysis

In Section 3 we discussed the model architecture we used for our experiments. Namely, the model was a 1 hidden layer multi-level perceptron neural network that was approximately 80 Kilobytes. We claimed that the 1 hidden layer was sufficient for retraining purposes and offered a strong tradeoff in terms of size. Indeed our results in Section 5 corroborate the claim that it meets sufficient accuracy. In this section, we will discuss an alternative

larger model architecture and compare the results against the smaller model from our experiments. In particular, we use a 3 hidden layer multi-level perceptron, with 150, 100, and 50 nodes in the hidden layers. The model size for this network is approximately 580 kilobytes.

As was discussed in the Model Accuracy subsection, our 3 hidden layer model was able to achieve an accuracy of 100% whereas our 1 hidden layer model was able to achieve an accuracy of 94%. Furthermore, we observed that it took less training iterations to get to our final model for the 3 hidden layer model.

Table VI shows the results of the trained models when applied back to the MobiAct dataset. We can see that neither the single layer nor the multi-layer offers a considerable improvement over the other, nor do either have a result that degrades considerably from the Vanilla model. Thus we conclude that while we get slightly better accuracy and training time with the larger model, the gains in accuracy are not enough to warrant the larger model size. This we conclude that the optimal model choice is indeed the smaller one.

TABLE VI
MODEL ACCURACY AGAINST MOBIACT DATASET FOR ALTERNATE MODEL ARCHITECTURE

Model	Vanilla	Vanilla-Retrained
1 Hidden Layer	99.59%	98.75%
3 Hidden Layers	99.93%	97.13%

A lingering question concerns the hardware of the design. The Intel UP Board we used had a storage space of 32 Gigabytes and 4 Gigabytes of RAM [6]. A model size of 80 kilobytes or even 580 kilobytes is hardly a concern for a system with these specifications. However, our goal is for future iterations of this design to move away from the UP board. As we discussed in the power analysis section the UP Board draws 13 watts in typical conditions. Furthermore, it is quite physically large, at least impractical as a true commercial wearable. Thus a new hardware platform is sought.

One potential option is the Raspberry Pi Pico. As previously discussed, it only consumes 0.4325 watts and is physically much smaller. Indeed this was the original hardware that was intended for this report. However, the Raspberry Pi Pico has a maximum storage space of 64 kilobytes. This means we would need to fit our model to 64 kilobytes, in addition to storing the program itself and any necessary libraries. We implemented a version of this that recreated many NumPy functions to avoid installing NumPy on the Pico. However, due to space constraints, this method very quickly became infeasible. Thus the Pico has too small a storage constraint to be used in this application.

Therefore, a future design would like need to be somewhere between a Pico and an UP Board. A reasonable choice would be to use a smartphone. The reasoning is threefold. First, most people carry around smartphones in their pockets so this removes the need for the device to a literal wearable. Two, the device is small and already engineered to optimize battery life. Three, most smartphones have IMUs built in (although not all of them use LSM330DLC chips), and additionally also have magnetometers. These 3 reasons reduce the space and power needed for our design. A challenge of this approach is consistency. Not all phones have the same IMU, battery life, our even API. Nonetheless, the benefits of using a smartphone outweigh the many issues of our current wearable device.

VI. CONCLUSION AND FUTURE WORK

Overall, we showed our model is able to learn to distinguish knee-slide events as a non-fall with 71.0% accuracy with only 1 retrain, back fall events as a non-fall with 80.0% accuracy with 3 retrains, and chair edge falls as a true fall with 94.0% accuracy with 4 retrains. Furthermore, we showed these models have an accuracy of 98.13%, 96.19%, and 98.75% respectively when applied against the original MobiAct dataset showing our model is correctly distinguishing the new activities without overfitting.

We observed that the greater number of retraining events, the better the final accuracies were. We noted that this somewhat obvious conclusion poses a problem for our self-labeling system. Namely, once our model has been retrained once it is difficult to get enough data to reach the next retraining threshold. Our current system of feature vector

similarity mitigates this issue, but more work is needed to come up with a more robust method to aid in this data starvation.

Furthermore, we observed that our current hardware setup, namely the UP Board, is grossly overkill for the task at hand. We concluded that smartphones are a far better hardware platform for future iterations.

Ultimately, we conclude that personalized fall detection does provide better performance than out-of-the-box models and that with a few adjustments to the above approach can be achieved at a broad scale.

ACKNOWLEDGMENT

The authors would like to thank the EE 292D teaching staff for their help and guidance on this project.

REFERENCES

[1] Chatzaki, Charikleia, et al. "Human daily activity and fall recognition using a smartphone's acceleration sensor." International Conference on Information and Communication Technologies for Ageing Well and e-Health. Springer, Cham, 2016.

[2] CDC. (2020, December 16). Keep on your feet-preventing older Adult Falls. Centers for Disease Control and Prevention. Retrieved December 9, 2022, from https://www.cdc.gov/injury/features/older-adult-falls/index.html

[3] Abou L, Fliflet A, Hawari L, Presti P, Sosnoff JJ, Mahajan HP, Frechette ML, Rice LA. Sensitivity of Apple Watch fall detection feature among wheelchair users. Assist Technol. 2022 Sep 3;34(5):619-625. doi: 10.1080/10400435.2021.1923087. Epub 2021 May 25. PMID: 33900885.

[4] Chatzaki C., Pediaditis M., Vavoulas G., Tsiknakis M. (2017) Human Daily Activity and Fall Recognition Using a Smartphone's Acceleration Sensor. In: Rocker C., O'Donoghue J., Ziefle M., Helfert M., Molloy W. (eds) Information and Communication Technologies for Ageing Well and e-Health. ICT4AWE 2016. Communications in Computer and Information Science, vol 736, pp 100-118. Springer, Cham, DOI 10.1007/978-3-319-62704-5.

[5] Apple. (n.d.). Apple Watch - Battery. Apple. Retrieved December 9, 2022, from https://www.apple.com/watch/battery/

[6] Intel. (n.d.). Up specifications. UP Bridge the Gap. Retrieved December 9, 2022, from https://up-board.org/up/specifications/

[7] Harari, Y., Shawen, N., Mummidisetty, C.K. et al. A smartphone-based online system for fall detection with alert notifications and contextual information of real-life falls. J NeuroEngineering Rehabil 18, 124 (2021). https://doi.org/10.1186/s12984-021-00918-z

[8] 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), 2020, pp. 1-7, doi: 10.1109/PerComWorkshops48775.2020.9156172.

[9] T. Vaiyapuri, E. L. Lydia, M. Y. Sikkandar, V. G. Díaz, I. V. Pustokhina and D. A. Pustokhin, "Internet of Things and Deep Learning Enabled Elderly Fall Detection Model for Smart Homecare," in IEEE Access, vol. 9, pp. 113879-113888, 2021, doi: 10.1109/ACCESS.2021.3094243.

[10] Harari, Y., Shawen, N., Mummidisetty, C.K. et al. A smartphone-based online system for fall detection with alert notifications and contextual information of real-life falls. J NeuroEngineering Rehabil 18, 124 (2021). https://doi.org/10.1186/s12984-021-00918-z

[11] Mauldin, T. R., Canby, M. E., Metsis, V., Ngu, A. H. H., & Rivera, C. C. (2018). SmartFall: A Smartwatch-Based Fall Detection System Using Deep Learning. Sensors (Basel, Switzerland), 18(10), 3363. https://doi.org/10.3390/s18103363

[12] Chatzaki, C., Pediaditis, M., Vavoulas, G., Tsiknakis, M. (2017). Human Daily Activity and Fall Recognition Using a Smartphone's Acceleration Sensor. In: Röcker, C., O'Donoghue, J., Ziefle, M., Helfert, M., Molloy, W. (eds) Information and Communication Technologies for Ageing Well and e-Health. ICT4AWE 2016. Communications in Computer and Information Science, vol 736. Springer, Cham. https://doi.org/10.1007/978-3-319-62704-5_7

[13] R. Zhu et al., "Efficient Human Activity Recognition Solving the Confusing Activities Via Deep Ensemble Learning," in IEEE Access, vol. 7, pp. 75490-75499, 2019, doi: 10.1109/ACCESS.2019.2922104.

[14] Vavoulas, George, et al. "The MobiFall dataset: An initial evaluation of fall detection algorithms using smartphones." 13th IEEE International Conference on BioInformatics and BioEngineering. IEEE, 2013.

[15] Y. He, Y. Li, C. Yin, "Falling-incident detection and alarm by smartphone with multimedia messaging service (MMS)," E-Health Telec. Syst. Netw., vol. 1, pp. 1–5, 2012.

[16] P. Vallabh, N. Malekian, R. Malekian, T. Li. "Personalized Fall Detection Monitoring System Based on Learning from the User Movements." ArXiv.org, 21 Dec. 2020, https://arxiv.org/abs/2012.11195.

[17] Wearable Technology Market Size, Share amp; Trends Report 2030. (n.d.). Retrieved December 14, 2022, from https://www.grandviewresearch.com/industry-analysis/wearable-technology-market

[18] Sposaro, Frank, and Gary Tyson. "iFall: an Android application for fall monitoring and response." 2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE, 2009.

[19] Shizume, T., amp; Salem, W. (n.d.). TTSHIZ/CS539_FALL_PREDICTION. GitHub. Retrieved December 14, 2022, from https://github.com/ttshiz/CS539_Fall_Prediction

VII. Appendix

A. Chris Calloway's Contributions

- Created initial breadboard model.
- Wired and programmed (Micropython) the Raspberry Pi Pico to control buttons and LEDs in parallel to fall prediction script.
- Experimented with neural network model and characterized model performance and power consumption.
- Implemented 3 hidden layer alternative neural network model.
- Setup LSM330DLC to properly read Gyroscope data.
- Implemented sensor fusion and extended Kalman filter to calculate pitch and roll.
- Brainstormed ideas on what experiments to conduct to test for false positives and false negatives.
- Implemented cosine similarity alternative retrain sequence to mitigate data starvation.

B. Estelle Kao's Contributions

- Installed OS Image and set up UP Board for edge deployment.
- Wrote preprocessing code for MobiAct dataset including data type conversion and feature extraction.
- Wrote binary classification neural network training script.
- Set up accelerometer/gyroscope collection channel on UpBoard and prototyped initial on-device training model.
- Set up proper code sequence for utilizing retraining on the edge.
- Compared accuracy of experiments and analyzed how it progressed through more retrains.
- Plotted all data recorded.

C. Source Code

- Source code available at https://github.com/estellekao/EE292D_Project.