FALL DETECTION USING FEDERATED LEARNING FOR MODEL PERSONALIZATION AND ANOMALY DETECTION

by

Nader Maray, B.S.

A thesis submitted to the Graduate College of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Computer Science
May 2023

Committee Members:

Anne Hee Hiong Ngu, Chair

Vangelis Metsis

Chul-Ho Lee

DEDICATION

This project is dedicated to my advisor Dr. Anne Hee Hiong Ngu, as she has been an inspiration to me with her hard work on the fall detection project in general, her mentorship and guidance have been instrumental in shaping my academic journey at Texas State University and providing me with invaluable academic experience. I will always be grateful for all the research opportunities she has given me.

ACKNOWLEDGEMENTS

I want to express my gratitude for the exceptional learning experience that I have received during my master's program. I would like to thank the entire computer science department for providing me with an outstanding academic environment, resources, and guidance.

Special thanks to Dr. Anne Hee Hiong Ngu for every opportunity she has presented me, as well as my committee chair members Dr. Vangelis Metsis and Dr. Chul-Ho Lee for being generous and giving me their input, time and effort for the thesis. I would also like to thank Dr. Lu Wang for voluntarily helping me through both my thesis and my paper with insightful input.

The master's program has been a transformative experience for me, and I attribute much of my success to the education and mentorship that I received from the department's faculty and staff. Throughout the program, I have had the opportunity to learn from some of the most brilliant minds in the field, and I have been impressed by all of the lecturer's and researcher's dedication to helping students achieve their goals.

The resources provided by the department have been invaluable in helping me succeed in my studies. The library, computer labs, and other facilities have been well-equipped and maintained, allowing me to engage in my studies effectively.

I cannot thank the department enough for the support and guidance that I have received during my studies. The knowledge and skills that I have gained will be invaluable to my future career, and I will always cherish the experience of being a part of this esteemed institution.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

ABSTRACT

Falls in the elderly are associated with significant morbidity and mortality. While numerous fall detection devices incorporating AI and machine learning algorithms have been developed, no known smartwatch-based system has been used successfully in real-time to detect falls for elderly persons. We have developed and deployed a SmartFall system on a commodity-based smartwatch which has been trialled by nine elderly participants. The system, while being usable and welcomed by the participants in our trials, has two serious limitations. The first limitation is the inability to collect a large amount of personalized data for training. When the fall detection model, which is trained with insufficient data, is used in the real world, it generates a large amount of false positives. The second limitation is the model drift problem. This means an accurate model trained using data collected with a specific device performs sub-par when used in another device. Therefore, building one model for each type of device/watch is not a scalable approach for developing smartwatch-based fall detection system. To tackle those issues, we will focus on two datasets including accelerometer data for fall detection problem from different devices: the Microsoft watch (MSBAND), and the Meta Sensor device. We have previously achieved good success in solving the limitations through the use of transfer learning, however, false positives still remained as a problem, as well as real-time model testing. To solve the remaining issues, we will try to apply two main methods, the first method is building a federated learning framework of multiple edge devices for multiple people, where each edge device would have its own personalized fall detection model, as well as its own synthetically generated data and real life data, while the second method would treat the problem as an anomaly

detection problem, where the standard action would be an ADL (Activities of Daily Life) and the anomaly would be a fall, and vice versa. Federated learning experiments showed promising results, achieving an average F1-score of 0.94, while the anomaly detection experiment did not achieve results better than the previous model.

I. INTRODUCTION

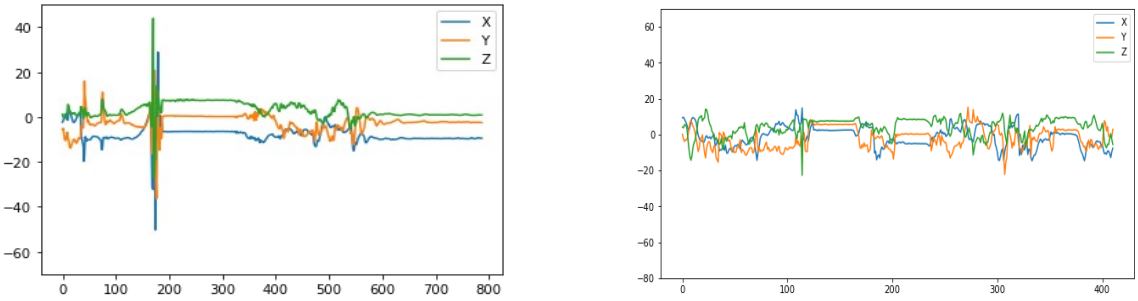
Falls are one of the leading causes of death and injury among the elderly population [1]. According to the U.S. Center of Disease Control and Prevention, one in four Americans aged 65 and older falls each year [2]. A recent CDC report also stated that around 28% of people aged over 65 lived alone [3]. In addition, the Agency for Healthcare Research and Quality reports that each year, somewhere between 700,000 and 1,000,000 people in the United States fall in the hospital alone [4]. The resultant inactivity caused by a fall in older adults often leads to social isolation and increased illnesses associated with inactivity including infections and deep vein thrombosis. Consequently, a large variety of wearable devices which incorporate fall detection systems have been developed [5, 6, 7, 8]. Wearable devices have the promise of bringing personalized health monitoring closer to the consumers. This phenomenon is evidenced in the articles entitled "Staying Connected is Crucial to Staying Healthy" (WSJ, June 25, 2015) and "Digital Cures For Senior Loneliness" (WSJ, Feb 23, 2019). The popularity of using a smartwatch, paired with a smartphone, as a viable platform for deploying digital health applications is further supported by release of the Apple Series brand of smartwatches [9] which has a built-in "hard fall" detection application as well as an ECG monitoring App. Apple also added car crash detection in the most recently version of Apple watches. An Android-Wear based commercial fall detection application called RightMinder [10] has been released on Google Play since 2018. One of the major sensors used in fall detection on a smartwatch is an accelerometer, which measures the acceleration of an object. Acceleration is the change in velocity with respect to time and velocity represents the rate at which an object changes its position. Acceleration data is commonly used in fall detection because accelerometer sensors are found in most smart devices, and a distinct change in acceleration

happens when a fall occurs. The clustered spikes in Figure 1a show a unique pattern in the acceleration data during one second when the fall occurs, which means that falls can be identified in acceleration data by that pattern.

Previously, we have developed a watch-based SmartFall App using Long Short-Term Memory neural networks (LSTM), an artificial recurrent neural network (RNN) with feedback connections, to detect falls based on the above pattern, by training it on simulated fall data collected using a Microsoft watch (MSBAND) [11, 12]. We have deployed this SmartFall system on a commodity-based smartwatch which has been trialled by nine sensor participants. Each participant was recruited under IRB 7846 at Texas State University to use the SmartFall system to collect their ADLs (Activity of Daily Living) data by just asking them to wear the watch for three hours per day over a seven day period. The user only needs to interact with the watch and provide feedback when false positives are generated by the system. Despite the system was welcomed by the participants in our trials, it still have several limitations: 1) fall detection models trained on simulated falls and ADLs performed by young, healthy test subjects suffer from the fact that they do not exhibit the same movement characteristics as the elderly population. For example, an elderly person typically has comorbidities that affect their movements including the effects of multiple medications, poor vision, stroke, arthritis, sensory neuropathies and neuro-degenerative diseases such as Parkinsonâs disease, all of which may contribute to their risk of falling [13]; 2) a sudden hand or wrist movement from some ADLs can interfere with the recognition of this pattern. For example, Figure 1b is the signal generated from a person putting on a jacket and has some cluster spikes which can be mistaken for a fall; 3) there is no guarantee that accelerometer data collected from different smartwatch devices is exactly of the same quality for fall detection since they have different hardware characteristics and API libraries.

In addition, we find that a fall detection model trained with data collected using a specific device usually does not generalize well to similar data collected using a different device because of differences in hardware characteristics which result in the acceleration data being sensed and recorded with varying G units, sampling rates, and X, Y and Z orientations of the accelerometer data. For example, Huawei watch (which we do not use in this paper) specified that data can be collected in 32 ms, but in reality, the data is always collected in every 20 ms while MSBAND collects data in 32 ms as specified. To tackle the aforementioned issues, we propose to use transfer learning approach to solve the small dataset problem in smartwatch based fall detection system. More specifically, while collecting a large amount of ADL or fall data from the elderly population is an unrealistic task (*i.e.,* the target domain), collecting a small amount of everyday movement data from the elderly population is possible (*i.e.,* the source domain). Therefore, the obtained model in the source domain can be utilized and retained in the target domain. This will enable us to create a real-world smartwatch-based fall detection model usable by older adult where we only need to collect a small amount of data to train a model tailored to each of them. After achieving a substantial improvement using the aforementioned transfer learning approach in [14], we conclude that there still relies a problem in the amount of false positive classifications, as that remains the biggest obstacle in improving results to a near-perfect level. One glaring limitation of the transfer learning approach is that it relies on a single global model to make predictions for all clients or users. This approach can lead to suboptimal performance on an individual level, as the model may not be tailored to the client's specific needs or characteristics.

To address this limitation, a personalized machine learning framework can be developed, where each client has their own unique classification model. This approach involves training a separate model for each client based on their historical



(a) Acceleration from a fall.

(b) Acceleration from putting on a jacket.

Figure 1: Comparison of smartwatch accelerometer data.

data, preferences, and behavior. These personalized models can then be used to make accurate predictions and recommendations for each individual client, leading to improved performance on an individual user level.

The development of such a personalized machine learning framework requires careful consideration of various factors such as data privacy, model scalability, and computational resources. Nonetheless, this approach has the potential to significantly enhance the effectiveness of machine learning applications in a wide range of domains, by providing tailored solutions that meet the unique needs of each individual client, hence, to solve the issue of excessive false positive classification, while maintaining accuracies for the other types of classifications, as well as data privacy.

we propose a fall detection framework inspired by federated learning, in which every client has their own classification model, tailored to their data, while maintaining a global model for all users in the server. More specifically, we use a copy (one copy per client) of a pre-trained model as the base personalized model for all clients in the framework, and gradually re-train each personalized model using that client's data, while averaging all the models' parameters after each training iteration into the global model on the server.

Another approach of solving the fall detection task is by considering falls as an

anomaly from normal day to day actions, since intuitively, a fall action rarely happens and is an abnormality in one's day to day life. Hence, building a model that is very well acquainted with datasets of day to day activities of clients (which do not involve falls) and is able to distinguish these activities, will also hopefully be able to tell whenever abnormal, unfamiliar data is fed into it, or in this case, fall data. Such tasks in machine learning are known as Anomaly Detection tasks.

Anomaly detection in machine learning is the process of identifying unusual data points or patterns in a given dataset that do not conform to the expected behavior. It is an essential aspect of many real-world applications such as fraud detection, fault detection, intrusion detection, and health monitoring. Machine learning algorithms use statistical models and mathematical techniques to analyze large volumes of data and identify anomalies that may indicate potential problems or opportunities for improvement. We will attempt to apply many of these models in our experiments.

The success of anomaly detection in machine learning largely depends on the quality and representativeness of the training data. It is a very challenging task since anomalies can take various forms and can be difficult to distinguish from normal data points. For example, in our case, the acceleration of the wrist which is induced by a normal daily life task, such as waving one's hand, could have a very similar pattern to the acceleration induced by a fall, especially since wrist acceleration alone, without the acceleration of any body part, isn't the most informative measure. Which is why, in our experiments, we try a multitude of data arrangement and sequencing methods, in order to make the most out of the wrist acceleration measure.

II. RELATED WORK

In this section, we review articles for both model personalization through the use of a federated learning framework, and for anomaly detection using different models, and explain how their findings can be of help to the fall detection task we are trying to solve in this paper. For related work on transfer learning, we will refer to our previous paper [14].

II.I. Personalized Federated Learning

Federated learning (FL) is a distributed machine learning paradigm where multiple clients collaborate to train a shared model, without exchanging their data directly. This approach is particularly useful in scenarios where the data cannot or should not leave the clients' devices, due to privacy or security concerns, or where the data is geographically distributed or too large to be centralized. However, FL faces several challenges, such as heterogeneity among clients, communication and computation bottlenecks, and the need to reconcile conflicting updates.

Personalized federated learning (PFL) is a recent extension of FL that aims to address the heterogeneity challenge by allowing each client to learn a personalized model tailored to their local data and preferences, while still benefiting from the collective knowledge of the other clients. PFL can be seen as a compromise between the extremes of centralized learning, where all data is pooled and processed by a single server, and decentralized learning, where each client learns independently without coordination.

Several works have proposed and studied PFL in various settings and applications. For instance, Yang et al. [15] proposed FedPerf, a framework that enables PFL for mobile devices by leveraging reinforcement learning to optimize the

allocation of resources and the selection of models. FedPerf was shown to achieve higher accuracy and faster convergence than non-personalized FL on several benchmark datasets. Kairouz et al. [16] presented a comprehensive survey of PFL, discussing its motivations, challenges, algorithms, and applications, and highlighting open research directions. The authors also provided an empirical evaluation of several PFL methods on real-world datasets, showing their effectiveness and scalability.

Other works have focused on specific aspects or variations of PFL. For example, Wu et al. [17] proposed a personalized privacy-preserving FL approach, where each client encrypts their data using a different key and shares only a part of it with the server, while still allowing personalized model updates. Zhang et al. [18] introduced PFL with differential privacy, where each client adds noise to their model updates to preserve privacy while still contributing to the collective learning. Li et al. [19] studied the effect of different aggregation strategies on the performance and fairness of PFL, showing that personalized federated averaging can improve both aspects compared to non-personalized federated averaging.

Overall, the articles present a promising approach to improving the accuracy and efficiency of federated learning algorithms by personalizing them to individual users. It perfectly matches the structure of our project, as each user has their own data, collected on their own edge device, which makes it easily applicable, by having a personalized model for each client which runs on their edge devices, and maintaining a global model in the server whose parameters are the average of all the clients' personalized models. This framework achieves a personalized model for each user, while maintaining user data privacy, as expected of a federated learning framework.

II.II. Anomaly Detection

Anomaly detection is an important problem in many domains, including cybersecurity, finance, healthcare, and manufacturing. In recent years, there has been growing interest in anomaly detection for time-based tasks, where the goal is to identify unusual behavior or events that occur over time.

One popular approach to anomaly detection for time-based tasks is based on time series analysis. Time series are sequences of data points that are collected at regular intervals over time, and they can be used to model a wide range of real-world phenomena. In this context, anomalies can be detected by identifying patterns in the time series that deviate from normal behavior.

Several techniques have been developed for time series anomaly detection, including statistical methods, machine learning algorithms, and deep learning models. We are gonna be focusing on the following ones being Variational Auto Encoders (VAE), Isolation Forests, Local Outlier Factors (LOF), One-Class Support Vector Machines (OCSVM), and Minimum Covariance Determinant (MCD).

VAE is a type of deep learning model that can be used for unsupervised anomaly detection in time series data. VAEs are capable of learning a probabilistic representation of the data, and can be trained to reconstruct the input data with minimal reconstruction error. Anomalies can then be identified by comparing the reconstruction error to a threshold value.

Isolation Forests are another machine learning technique that can be used for time-based anomaly detection. Isolation Forests construct decision trees to isolate individual data points, and anomalies can be detected by measuring the path length required to isolate a given point. The shorter the path length, the more likely the point is an anomaly.

LOF is a local density-based outlier detection algorithm that can be used for

time series anomaly detection. LOF measures the degree of abnormality of a given data point by comparing its local density to the local densities of its neighbors. Anomalies are identified as data points with a significantly lower density than their neighbors.

OCSVM is a type of support vector machine that is trained on only one class of data points (i.e., normal data points). Anomalies are then identified as data points that fall outside the decision boundary of the OCSVM. OCSVM is a powerful technique for detecting anomalies in time-based tasks where the anomalous behavior is different from the normal behavior.

MCD is a robust estimator of covariance that can be used for time series anomaly detection. MCD is based on the idea of computing the covariance matrix using only a subset of the data that is least affected by outliers. Anomalies are then identified by computing the Mahalanobis distance between each data point and the mean of the subset.

The article "Anomaly Detection in Time Series Sensor Data for Machine Health Monitoring" by Lin et al. [20] proposes a machine learning-based approach for detecting anomalies in time series sensor data from manufacturing machines. The proposed method uses various anomaly detection models, including an Autoencoder, and One-class support vector machine to classify data points as normal or anomalous. The authors also evaluate the proposed methods using two publicly available datasets and compare its performance with four other existing methods. The results show that the proposed method outperforms the existing methods in terms of accuracy, precision, recall, and F1-score.

"Anomaly Detection in Time Series Data: A Survey and Evaluation" by Chandola et al. [21] - This paper provides a survey of existing methods for anomaly detection in time-series data and evaluates their performance on several benchmark datasets. The authors evaluate several methods, including LOF and Isolation Forest,

and compare their performance with other methods.

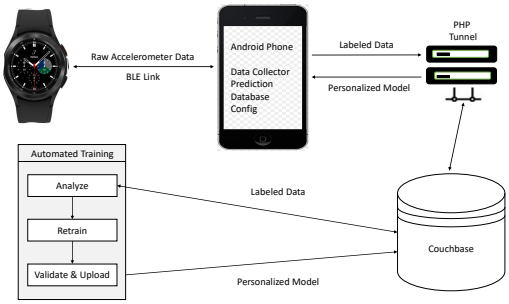
"Detecting Anomalies in Time Series Data via Minimum Covariance Determinant Estimation" by Rousseeuw et al. [22] - This paper proposes a method for detecting anomalies in time series data via Minimum Covariance Determinant (MCD) estimation. The authors evaluate the proposed method on several synthetic and real-world datasets and compare its performance with other existing methods.

Overall, there are many machine learning techniques that can be used for detecting anomalies in time-based tasks. The articles mentioned above managed to achieve good results using the aforementioned anomaly detection models, applying them on time-based tasks, which matches with the settings of the task at hand

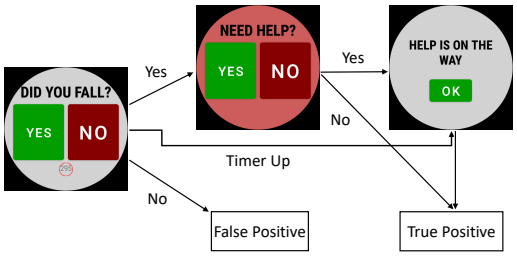
III. SMARTFALL SYSTEM ARCHITECTURE

We implemented a three-layered architecture which has the smartwatch on the edge, the smartphone in the middle layer, and the cloud server in the inner most layer. This is one of the most flexible architectures for IoT applications as discussed in [23] and is a practical choice for our prototype. Microservice is a particular implementation of the service-oriented architecture (SOA) that enables an independent, flexible, and distributed ways of deployment of services on the internet. Applications designed with microservices contain small, modular, and independent services which communicate via well-defined APIs. As compared to the 3-layer architecture of our SmartFall, microservices are more agile, flexible, and resilient. However, each microservice must be hosted in a container and connected to a cloud framework. Moreover, the portability of an edge container is not proven yet. Currently, there are no Docker-compatible containers that can run on an edge device like an Android phone. We have explored a microservice-based architecture called Accessor-based Cordova host for edge devices in [24].

Figure 2a gives an overview of the SmartFall fall detection system. The major software components developed on a smartphone are (a) the *Config* module which



(a) An overview of the SmartFall system.



(b) Watch's user interface display after a fall is detected.

Figure 2: Architecture of SmartFall system.

manages the parameters, version of the deep learning model used by a particular user, the chosen personalization training strategy, and the chosen cloud server for data storage and re-training; (b) the *Database* module which manages all the data sensed, the uploading of the collected data to the cloud, and the downloading of the best re-trained model for a user; (c) the *Data Collector* module which manages the transfer of sensed data on the smartwatch to the smartphone using different communication protocols. Our smartwatch and smartphone currently communicate using BLE. The smartwatch and the server communicate using HTTP. Our system is designed to leverage multiple communication protocols; and (d) the *Prediction* module, which manages different machine learning models used for fall detection. For example, the system can be configured to run an ensemble recurrent neural network (RNN) or a single RNN model. On the cloud, additional software components for analysis, re-training and validation of the re-trained models are implemented. Our system is designed to be flexible for using different personalization strategies as and when they become available.

The smartwatch's UI is designed to start with just the "YES" and "NO" buttons so as to overcome the constraint of small screen space (see Figure 2b). If the user answers "NO" to the question "DID YOU FALL?", the data is labelled as a false positive and stored as "FP" in the Couchbase database in the cloud. If the user answers "YES", the subsequent screen will prompt "NEED HELP?". If the user presses "YES" again, it implies that a true fall is detected and that the user needs help. The collected data will be labeled and stored as "TP" and "HELP IS ON THE WAY" screen will be displayed. If the user presses "NO", it suggests that no help is needed and the collected data is still labelled as "TP". If the user did not press either "YES" or "NO" after a specified period of time following the question "DID YOU FALL?", an alert message will be sent out automatically to the designated caregiver.

Our system is structured such that all user-identifying data is only stored

locally on the phone to preserve privacy. Real-time fall prediction is performed on the phone to reduce the latency of having to send data to the cloud for prediction. The training/re-training of the prediction model is done offline in the cloud server. The UI interface is designed such that there is no need to interact with the App unless the system detects that a fall has occurred, in that case, the watch will vibrate to alert the user that a prediction has occurred and the UI in Figure 2b will appear. The ability to interact with the system when a false prediction is generated allows the system to collect real-world ADL data and fine tune the fall detection model.

The ultimate goal is for the system to detect falls accurately, i.e. not missing any falls and not generating too many false positive prompts. Collecting data and training a new model from scratch is labour intensive, hence, we aim to have one model that can generalize well across different smart devices. When a new device is added, by using a small amount of feedback data collected by the user wearing the device for a short period of time, a new model can be trained with a transfer learning strategy and uploaded to the device to use in real-time. The following sections describe the transfer learning experiments we conducted to support our vision in this SmartFall system.

IV. METHODOLOGY

IV.I. Dataset Collection

We first collected two datasets which can be used in the transfer learning experiments. Those datasets are comprised of accelerometer data collected from the Microsoft watch (MSBAND) watch, and the Meta Sensor device. MSBAND data was collected in units of 1G on the left wrist only, while Meta Sensor data was collected in units of 2G on both the left and right wrists. The sampling rate is 32 Hz for MSBAND watch while Meta Sensor data is collected with the sampling rate of 50 Hz. Figure 3 shows the three different devices we used for the data collection process.

Figure 3: The three different hardware used for data collection.

The MSBAND dataset was collected from 14 volunteers each wearing a MSBAND watch. These 14 subjects were all of good health and were recruited to perform a mix of simulated falls and ADLs (Activity of Daily Living). Their ages ranged from 21-55, height ranged from 5 ft to 6.5 ft. and weight from 100 lbs to 230 lbs. Each subject was told to wear the smartwatch on his/her left wrist and perform

a predetermined set of ADLs consisting of: walking, sitting down, picking up an object, and waving their hands. This initial set of ADLs were chosen based on the fact there were common activities that involved movement of the wrists. Those data were all labelled as "NotFall". We then asked the same subjects to perform four types of falls onto a 12-inch-high mattress on the floor; front, back, left, and right falls. Each subject repeated each type of fall 10 times. We implemented a data collection service on an Android phone (Nexus 5X, 1.8 GHz, Hexa-core processors with 2G of RAM) that paired with the MSBAND smartwatch to have a button that, when pressed, labels data as âFallâ and otherwise âNotFallâ. Data was thus labelled in real-time as it was collected by the researcher holding the smartphone. This means when the user was walking towards the mattress before falling down or getting up from each fall, those duration of data will be labelled as 'NotFall". However, the pressing of the button can introduce errors such as the button is being pressed too late, too early, or too long for a fall activity. To mitigate these errors, we post-processed the collected data to ensure that data points related to the critical phase of a fall were labeled as âFallâ. This is done by implementing an R script that will automatically check that for each fall data file, the highest peak of acceleration, and data points before and after that point, were always labeled as âFallâ. After this post-processing of the collected data, we have a total of 528 falls and 6573 ADLs. The MSBAND watch was decommissioned by the vendor in May 2019. This dataset is available at http://www.cs.txstate.edu/ hn12/data/SmartFallDataSet.zip.

Meta Sensor was developed by MBIENTLAB in San Francisco (mbientlab.com). It is a wearable device that offers continuous sensing of motion and environment data. It can sense gyroscope, accelerometer and magnetometer, and it provides easy-to-use open source APIs for fast data acquisition. Data can be stored locally on the phone or in a cloud server provided by MBIENLAB. The Meta Sensor we used is the MetaMotionRL. The sensor has a weight of 0.2 oz and can be recharged via

USB port. By embedding the meta sensor in an appropriate wrist band, it can serve as a wrist watch for easy collection of ADLs and simulated fall data. The collected data can be exported into multiple file formats. We recruited 8 participants (3 male and 5 female) with ages from 22 to 62 for data collection. Each participant is asked to perform four types of fall (front, back, left and right), five time each on an air mattress, and a prescribed list of ADLs. These are walking, waving hand, drinking water, wearing a jacket, sitting down and picking stuff from the floor.

The Meta Sensor fall data was first programmatically labeled by a Python script that identifies a set amount of peak magnitudes based on the amount of trials per file and a uniform width of 35 data points (1.12 seconds) per fall. Plotting programmatically labelled Meta Sensor data in Microsoft Excel showed that labels were often placed around peaks caused by noise rather than actual falls and did not capture the distinct pre-fall, fall, and post-fall activity that accompanied an actual fall. To ensure that we have a set of accurately labelled Meta Sensor data to experiment with, we decided to manually relabel all Meta Sensor data using Excel plots as a basis for fall window placement. We choose fall windows with a width of 100 data points in attempt to capture both pre-fall and post-fall activities. To minimize noise, we trimmed non-fall data in between each fall. Since an ADL activity could last much longer than a fall, we label the non-fall data in ADL files to the smallest multiple of 100 data points per trial that could capture the entire activity being performed. The collected Meta Sensor data has 202 falls and 492 ADL samples, and is available at
http://www.cs.txstate.edu/hn12/data/Meta_sensor_7030.zip.

IV.II. Experimental Settings

IV.II.I. Federated Learning

Federated learning is a machine learning approach that allows multiple devices to collaborate on the training of a shared model, without sharing the raw data that they hold. This is achieved by training the global model on the local data of each device, and then sending only the model updates back to a central server, which aggregates them to improve the global model. In our case, since we want to focus on model personalization, we maintain a copy of the best performing model for each user on that user's edge device, and use that model instead of the shared global model for training on the local data of that user, meaning that not only do we maintain a shared global model which is updated by the local data of all users, but for each user we maintain their own personalized model on the edge device. For ease of understanding, we select one of our experiments to explain how the federated learning strategy works in this study.

As mentioned before, in our Meta Sensor dataset, we have the data of 8 different people. For the sake of our experiment, each person's data will be split into what we call 4 data cycles. Each persons Fall and ADL data will be split evenly into those 4 cycles without any data intersection between cycles. Each person will have their own personalized model, which initially is a base model trained on the entireity of the MSBAND dataset, as well as a 9th global model. We then would iterate 4 times (as many times as there are data cycles) where on the i'th cycle, for each user, we would test the performance of that user's personalized model against the global model on that user's i'th data cycle, and whichever model performs better will be set as the new personalized model for that user, and it will be trained on the i'th data cycle of that user. After going through all 8 users in the i'th cycle, re-setting each user's personalized model, and training it on that user's i'th data

cycle, we would average the parameters of all 8 personalized models, weighted by their F1-scores, into the new global model, and go on to the next iteration.

The full federated learning process is described in algorithm 1. In the algorithm, we have the base model (trained on the MSBAND dataset), the users' data, and the number of federated learning iterations as inputs. We start off by organizing the data into cycles for each person and initializing all the personalized models for each user, as well as the global model, to have the same parameters as the base model. We then start iterating through the federated learning process, for each iteration, comparing each personalized model's performance to that of the global model on the iteration's data cycle, and updating the personlized model to be the one of the two which performed better, and then training that model on the data cycle, and eventually, after doing that to all the personalized models, we update the global model to be the weighted average of all the personalized models, and move on to the next iteration. All our experiments are conducted on a Dell Precision 7820 Tower, 256 GB RAM and one GeForce GTX 1080 GPU using TensorFlow.

Algorithm 1 Our Federated Learning Structure

Input: ***Base_ Model***, Data of multiple users ***Fed_ Data***, Number of federated learning cycles ***Num_ cycles***

For Each User, Organize Their Data From ***Fed_ Data*** Into ***Num_ Iterations*** Cycles

For Each User i, Initialize The Personalized ***User_ i_ Model*** to ***Base_ Model***

Initialize ***Global_ Model*** to ***Base_ Model***

START LOOP, Iterating ***Num_ Iterations*** Times, For Each Iteration Do:

For Each User, Compare F1-Score of Personalized Model to Global Model On Current User Cycle Data, And Update Peronalized Model To Be The Better Performing Model, Then Train Model On Current User Cycle Data

Update Global Model to be The Weighted Average of All Personalized Models

END LOOP

IV.II.II. Anomaly Detection

Anomaly detection is a machine learning technique used to identify observations or data points that differ significantly from the norm or expected behavior. The goal is to identify rare events or patterns that deviate from the typical or normal behavior of a system or process, which could indicate potential problems, fraud, or other anomalous activity. As mentioned before, this relates to our problem in that the norm or expected behaviour is what we call ADL (Activities of Daily Life) actions, and the anomaly is the Fall action.

For our experiments, we will have a variety of anomaly detection models. In each experiment, we will be exposing the anomaly detection model to only one type of data as its training phase, either the ADL or the Fall data. Whichever type of

data the model gets exposed to will be considered as the "norm" or "expected behaviour", and the other type of data will be the anomaly. After the training phase, we will use an unseen dataset (test set), containing both ADL and Fall data, to evaluate the model's F1-score. This implies that we have 2 experiments per model, in one of them, Falls are considered an anomaly, and in the other, ADL actions are considered an anomaly. Another experiment we'll be showing is similar to what's described above but a model gets trained on only a specific set of ADL data, such as training strictly on data from waving hand, and then testing on both waving hand and fall data.

IV.III. Model Training and Parameters Tuning

IV.III.I. Federated Learning Structure and Tuning

As mentioned before, we used a simple LSTM neural network structure for our model, as not only does that fit the time-series task well, but it is also a viable option for real-time classification that operates on the edge device without having the need to communicate to the cloud. Our classifier had many different hyperparameters, as well as different options for layer structuring, all of which needed extensive tuning in order to find which permutation of these hyperparameters and structures gives the best result. The main hyperparameters for our classifier are:

- **Window_Size:** The number of consecutive data entries that will be fed to the LSTM classifier at once. For example, if the window size is 35 (meaning the length of a single input block is 35 time-consecutive data entries), then the classifier will be fed a tensor of the shape 35x3 (since we have 3 coordinates for acceleration for each entry) to give a single classification for. This snapshot of a particular window size represents one sample of time series data as shown

in Figure 1a.

- **Step_Size:** The difference between two consecutive data **blocks** (each block comprised of **Window_Size** data entries). For example, say we have 37 data entries, with a **Window_Size** of **35** and a **Step_Size** of **1**, then, we would have 3 different data blocks, them being [1, 35], [2, 36] and [3, 37], which means we have an overlap of 34 entries between each 2 consecutive data entries. If **Step_Size** was 2, then we would have 2 different data blocks, them being [1, 35] and [3, 37] (the middle block would be skipped since our step is 2), with an overlap of 33 entries between each 2 consecutive entries (**Window_Size** - **Step_Size** is the general number of overlapping entries).

- **Smooth_Window:** The way we have our model make a final prediction is by predicting over the last **Smooth_Window** data blocks, and then average (take the median of) the predictions and use that average as the final fall probability. The motivation behind the smooth window is to take into account a wider scope of predictions, better covering pre-fall and post-fall data points. This will also ensure that we do not miss any clustered spikes related to fall and we do not just take a single spike as a fall prediction.

- **Fall Threshold:** After having the averaged fall probability from the most recent smooth window, if its value is greater than **Fall Threshold**, then we classify the window as a fall, otherwise we classify it as a non-fall.

As mentioned above, the hyperparameter tuning process needed an extensive amount of experimentation, and for each hyperparameter we tried a multitude of different numbers from lower to higher values. In this part of the sub-section, we will be describing the experimentation process for each hyperparameter and mentioning what the optimal value is with the reasoning behind it. The hyperparameter turning process was validated on the MSBAND and Meta Sensor

datasets, for each dataset separately, by splitting that dataset into a training set, which consisted of 70% of the data, and a test/validation set, which consisted of 30% of the data. For each choice of hyperparameters, we would train our classifier on the training set, and then calculate the **F1 score** of the trained model on the test set. In the results tables, we show the scores of 5 different values as the other values' results were similar to the value closest to them in the table.

- **Window_Size:** We tried a multitude of different values, and found that the optimal value is the same as the number of data entries sensed within 1 second (the duration of a fall), meaning that the optimal value for the MSBAND model was **32**, as the MSBand is at 32 Hz, and the optimal value for the Meta Sensor model was **50**, as the Meta Sensor is at 50 Hz. This seemed to be the sweet-spot that captures enough data for an accurate classification, any value below that gave a worse classification accuracy, and any value beyond that did not increase the classification accuracy by a noticeable amount.

Table 1: **Window_Size** tuning for MSBAND and Meta Sensor datasets respectively

Value	15	20	**32**	40	50
F1-Score	0.8	0.85	**0.93**	0.91	0.92

Value	30	40	**50**	60	70
F1-Score	0.75	0.76	**0.81**	0.81	0.8

- **Step_Size:** Out of all the values, a step of 1 seemed to perform the best, which indicates that high overlap and small increments between the consecutive data blocks is important for a good performance, as all the higher values gave worse results.

Table 2: **Step_Size** tuning for MSBAND and Meta Sensor datasets respectively

Value	**1**	3	5	7	9
F1-Score	**0.93**	0.9	0.87	0.88	0.86

Value	**1**	3	5	7	9
F1-Score	**0.81**	0.77	0.79	0.75	0.73

- **Smooth_Window:** As explained before, we want to capture the notion of both pre-fall and post-fall occurrence in order to help us better classify falls and have less false positives, and exactly matching that intuition, a broader smooth window of about 2 seconds of sensed data entries (64 for MSBand and 100 for Meta Sensor) out-performed both shorter and longer smooth windows.

Table 3: **Smooth_Window** tuning for MSBAND and Meta Sensor datasets respectively

Value	20	40	**64**	80	100
F1-Score	0.83	0.89	**0.93**	0.86	0.87

Value	20	60	**100**	130	160
F1-Score	0.69	0.75	**0.81**	0.75	0.78

- **Fall_Threshold:** Different values in increments of 10% were tried, starting from 10% and ending at 90%, and the fall threshold of 40% performed the best as it had the best balance of accurate true-positive classification while avoiding as many false-positives as possible. This value wasn't picked solely through experimentation, but also by looking at the prediction probability of the classifier over the test set, we can see that for the fall data, the classifier predicts values above 40%, and for non-fall data, it predicts values below 40%.

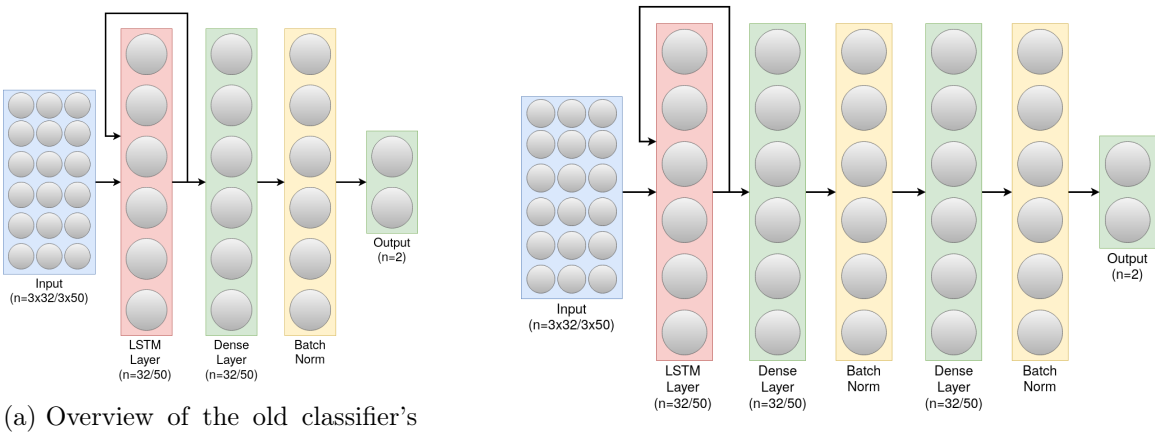
Table 4: **Fall_Threshold** tuning for MSBAND and Meta Sensor datasets respectively

Value	0.1	0.3	**0.4**	0.7	0.9
F1-Score	0.68	0.85	**0.93**	0.81	0.67

Value	0.1	0.3	**0.4**	0.7	0.9
F1-Score	0.6	0.76	**0.81**	0.73	0.65

As we have mentioned, not only did we tune the hyperparameters of the network, we also tried several structures for the network itself, mainly following the LSTM layer, as a part of our model tuning. Previous work's benchmark model is illustrated in Figure 4a.

As we can see, the model consisted of an LSTM layer, followed by a dense layer, batch normalization and ended off with another dense layer. It worked well as



(a) Overview of the old classifier's architecture.

(b) Overview of an improved LSTM classifier.

Figure 4: Comparison of classifier architectures.

is, however, through examining the training accuracy during the training process, the accuracy value seemed to plateau earlier than desired, which is what led to experimenting with the network structure by adding more, but not too many, additional dense layers, up to a point where it wouldn't impact the classification time, and enough to be able to overcome the training accuracy plateau as well as achieve better test accuracy. And indeed, after thorough experimentation, a more optimal structure was achieved, one that had more parameters (from 13,601 to 16,351 parameters), hence more potential for knowledge gain, while maintaining relatively quick classification speed. The new structure simply had 2 additional layers, a batch normalization layer followed by a dense layer. The structure of the new model can be seen in Figure 4b. It is worth noting a few things that are consistent between our model and the previous work's model:

- All layers are fully connected, using drop-out/convolution layers made the performance of the model slightly worse, hence we do not use any of those layers.
- The activation function of the dense layers is Relu, and the last layer uses Sigmoid which is commonly used for binary classification.

- The default Keras Library's Binary Cross-Entropy loss function as well as the default Adam optimizer were used as the loss function and optimizer of the network, as those two worked well in our older version of classifier.

- The number of neurons in the LSTM layer, as well as the output dimensions of the Dense layers were always set to the number of data entries sensed in one second, similarly to **Window_Size**, as that generally gave the best result.

IV.III.II. Anomaly Detection Structure and Tuning

As for Anomaly Detection, we used 4 anomaly detection methods being:

- sklearn: Variational Auto Encoder, Local Outlier Factor, Isolation Forest, One Class SVM

- pyod: Minimum Covariance Determinant

The hyperparameters' optimal values did not make a big difference in the result, hence we ended up using the default value for most of the hyperparameters of the 4 models. For that reason, we will not be elaborating on hyperparameters tuning process in the paper.

V. EXPERIMENTS AND RESULTS

In this section, we present our experimental results on federated learning for personalization, as well as our results on anomaly detection. The datasets we use for the experiments are the MSBAND and Meta Sensor datasets, as well as synthetically generated data off of the Meta Sensor dataset. In our federated learning experiments, we always start off by building a model from scratch on the full MSBAND dataset, and then, we use that as a base model for our federated learning part of the experiment, meaning that, for each of the 8 subjects involved in the Meta Sensor dataset, we create their own personalized model, and use a copy of the trained model as a base for that subject's personalized model, and we also use the same base model for the base global model, and proceed with the experiment from there.

In our anomaly detection experiments, we used a basic train/test split for evaluation. For each experiment, we noted one type of data (Fall or ADL) as the anomaly, and trained the anomaly detection models on the normal data, and then tested the model on a test dataset comprised of both normal and anomaly data, meaning that per model, we conducted two experiments overrall, one where fall data is perceived as an anomaly, and the other is where ADL data is perceived as an anomaly. The reason why we did not use anomaly detection for personalization is because we wanted to achieve good preliminary results using basic evaluation methods first, as we did with deep learning, which we unfortunately weren't able to achieve.

V.I. Federated Learning

For our federated learning experiments, we tried a multitude of different experiments, all of which operated according to the algorithm we described in section IV.II.I., and have a goal of building a personalized model per subject (8 total subjects + the global model) of the Meta Sensor data. For all experiments, including our benchmark score from [14], as mentioned above, we used a model trained on the full MSBAND dataset as our base model for all 9 models. All of our models used the optimal hyperparameter choices and the optimal network structure described in the section IV.III. and figure 4b.

For evaluation, which we conducted on the Meta Sensor dataset, we split each subject's data into 4 parts, each part containing and equal amount of fall and ADL data. We then would run our federated learning framework for 4 cycles using the 4 data sections we created. As described in algorithm 1, each federated learning cycle, we compare the performance of each personalized model with the global model, and set the personalized model to be whichever one of the two performed better on the current cycle's data, we then train that model on the current cycle's data (the same data it was tested on), and average out all the personalized model to create a new global model, and move on to the next cycle. Our main evaluation metric will be the average F1-score of all 8 subject's personalized models on the 4th and final federated learning cycle, as that is the cycle in which the models have trained on the most amount of data (around 70% of the data), hence being at their most "personalized" stage, while there remains unseen data to test on. The reason we chose 4 federated learning cycles mainly comes from the size of our dataset, as it isn't big enough to be split into many more cycles, since we would have too little data in each cycle, making 4 the highest number of cycles we can use while maintaining a reasonable amount of Fall and ADL data in each cycle. And anything

less than 4 cycles would be too little as we want the process of personalization and averaging the models to happen a sufficient amount of times to be effective.

Before we present the experiments, it is worth noting that we tried 2 additional experiments, one involved multitask learning for personalization where the data subjects would be considered as seperate tasks, and the other involved model clustering based on height and weight where subjects would be clustered into the same group based on similarities in height and weight, and each group would be considered as a task with its own model "personalized" to the entire group. Neither of the experiments managed to produce results than the baseline benchmark (F1 scores of 0.85 and 0.82 respectively), which is why we did not include them.

Detailed below are the several experiments and their comparisons:

1. **Federated Learning Starting From MSBAND Model:** In this experiment, We trained a model on the full MSBAND dataset and used it as a base to perform the evaluation experiment using federated learning on the Meta Sensor dataset, which is described in the beginning of the current section. The PR Curve and Predictions are presented in Figure 5 under the titles starting with "FL No Synthetic".

2. **Federated Learning Starting From MSBAND Model + Synthetic Data:** In this experiment, We trained a model on the full MSBAND dataset, and then, for each subject in the Meta Sensor dataset, we generated synthetic data which consisted of both fall and non-fall data using Gretel AI (Gretel.ai), which uses a GAN for time series data generation [25], where for each subject, we used their personal data from the Meta Sensor dataset as the input for the generative model. We then would train a personalized model for each subject using the subject's synthetic data, and use that model as a base to perform the evaluation experiment using federated learning on the Meta Sensor dataset, which is described in the beginning of the current section. The PR

Curve and Predictions are presented in Figure 5 under the titles starting with "FL Synthetic".

Figure 5 contains the results of the two mentioned federated learning experiments, as well the transfer learning evaluation results from our previous paper as a benchmark. There is one row per experiment for all 4 experiments. In each row, on the left, there is the ROC Curve plot, with the F1-Score on the top right corner of the plot, and the AUC mentioned in the title above, and on the right, there is the prediction probabilities plot, with the x axis being the time (number of entry), the y axis being the prediction threshold, the blue data representing the falls (at threshold 1.0), and the red data representing the prediction probabilities, meaning that whenever the red line crosses the 0.4 threshold (fall prediction threshold on the y axis), the data entry is predicted as a fall, and otherwise it is predicted as a non-fall.

In our previous paper, we split the Meta Sensor data into a 70% training dataset, which contained the same data used in the first 3 cycles of our federated learning data, and a 30% test dataset, which contained the data from the 4th remaining federated learning cycle of all the subjects combined.

The first row of plots in figure 5, whose titles start with "Non TL", presents the evaluation results of building a model from scratch using the 70% training dataset, and evaluating it on the 30% test dataset. As we can see, it achieved an F1 score of 0.81 with an AUC of 0.81 as well. If we look into the prediction probability plot, we can see that fall prediction was for the most part successful, however, there were quite a few false positive predictions, with the red line going over the 0.4 threshold mark several times along the x axis where there were no falls, which was the main thing holding the model back.

The second row of plots in figure 5, whose titles start with "TL", presents the evaluation results of building a model using the full MSBAND dataset, and transferring it as a base model to train on the 70% training dataset, and evaluating

it on the 30% test dataset. As we can see, it achieved quite an improvement comparing to the previous experiment, with an F1-score of 0.93 and an AUC of 0.85. If we look into the prediction probability plot, we can see that fall prediction was for the most part successful, and the false positive classification decreased significantly in comparison with the previous experiment, with just a few false positive classifications remaining.

The third row of plots in figure 5, whose titles start with "FL No Synthetic", presents the evaluation results of the first federated learning experiment of this paper. The plots are taken from a random subject's evaluation on the 4th data cycle, while the F1 score is the weighted-averaged F1 score of the 4th data cycle of all the subjects. As we can see, we managed to achieve a slight improvement with the F1-Score reaching an average of 0.94, as well as a slight AUC improvement of 0.88. If we look into the prediction probability plot, we can see that there were no false positive classifications, and fall prediction was for the most part successful, however, we can see that in a few places, it got dangerously close to the prediction threshold. This was a pattern that we observed not just in the subject's presented prediction plot, but also other subject's plots as well. The classifier was not very aggressive in fall prediction probability values, which suggests that while we reduced the number of false positives, we may have sacrificed our low rate of false negatives, which could be the reason why the F1 score isn't much higher than the previous experiment.

The fourth row of plots in figure 5, whose titles start with "FL Synthetic", presents the evaluation results of the second federated learning experiment of this paper. The plots are taken from the same subject's data as above on the 4th data cycle, while the F1 score is the weighted-averaged F1 score of the 4th data cycle of all the subjects. As we can see, Both the F1-Score and AUC were near complete, both being at around 0.98, and the same applies for the prediction probability plot, with the red line being near-perfect. As good as the results are, they look almost too

good to be true in a sense, which very likely suggests that the synthetic data that we generated may just be really similar, if not a copy of the real data, which made the personalized models overfit the data before testing on it.

Overall, if we compare the results of our federated learning experiments to the previous benchmarks, the first thing we can see is the weighted-average F1 scores of the federated learning experiments are higher than the F1 scores of the TL benchmark. It is worth noting that this an important measure, as the test dataset of the TL benchmark is comprised of the data of all the subjects in the 4th federated learning cycle, which means that we managed to out-perform the previous benchmark after combining the personalized models' prediction results. That can mainly be attributed to the decrease in false positive classifications that were achieved by the new federated learning models. However, that decrease came at another cost which is a rise in false negative classifications, as in the third experiment we observed that the model was less aggressive hence made more false negative classifications, which is the only obstacle in its way of achieving a near-complete classification model.

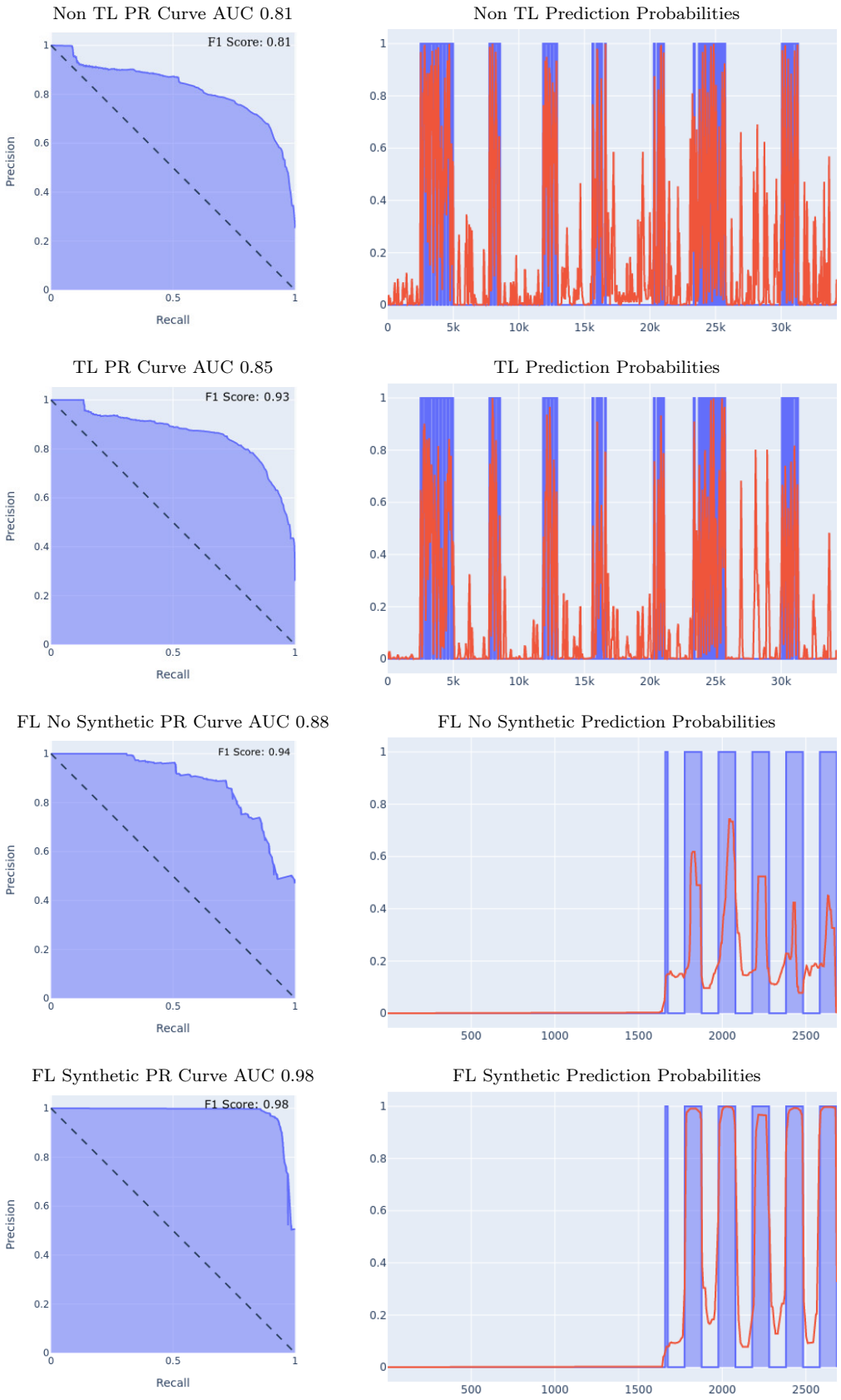


Figure 5: This figure compares the results of our previous paper with the two federated experiments we conducted in this paper.

V.II. Anomaly Detection

As previously described, we conducted two experiments per anomaly detection model using only the MSBAND dataset. In one experiment, falls were considered as anomalies, and in the other, ADLs were considered as anomalies. For both experiments, we used 70% of the **non-anomaly data** as training data, and tested on the rest of the dataset (including the anomaly data). Presented in the following tables are the results of the experiments:

Table 5: Results For the Experiment of Anomaly Fall data

	TP	TN	FP	FN	F1-Score
VAE	0	86001	0	5025	0
Isolation Forest	2535	76645	9356	2490	0.3
Local Outlier Factor	4390	3410	82591	635	0.1
MCD	0	86001	0	5025	0
One Class SVM	4212	1596	84405	813	0.1

Table 6: Results For the Experiment of Anomaly ADL data.

	TP	TN	FP	FN	F1-Score
VAE	86001	0	5025	0	0
Isolation Forest	4772	150	85851	253	0.1
Local Outlier Factor	1821	50402	35599	3204	0.1
MCD	75	85780	221	4950	0
One Class SVM	4714	1384	84617	311	0.1

As we can see, no method managed to achieve any significant results whatsoever. All of the algorithms had an excess of false positive predictions or an excess of false negative predictions, and the best F1-Score result we managed to

reach was 0.3, achieved by the Isolation Forest algorithm in the first experiment. All other F1-scores were equal to or below 0.1.

Our main speculation as to why the results are this bad comes from the similarity of non-fall data wrist acceleration to the fall data wrist aceeleration, up to the point where it isn't distinguishable using the anomaly detection techniques that we tried.

Another reason for that could be our hyperparameters choice, as even though we tried a multitude of different hyperparameters per anomaly detection model, we may have still not made the right choice when it comes to both the model hyperparameters, as well as the data hyperparameters (Window Size, Smooth Window, etc...).

One final speculation we had was that there is a specific type of ADL data which is too similar to fall data which is causing the confusion between ADL and fall data. It is worth mentioning that we tried using data from only one ADL task along with the rest of the falls data for anomaly detection, in an attempt to uncover an ADL action whose data acceleration pattern may be hard to distinguish from a fall for the anomaly detection algorithms, to no avail (all different ADL actions gave bad results).

VI. CONCLUSION AND FUTURE WORK

We presented an approach for fall detection based only on the acceleration data coming from an off-the-shelf wearable edge-device on the wrist of the subject. Fall detection using acceleration data coming strictly from a wearable on the wrist is challenging for the reason that there is a lot of room for false positives, as many activities of daily living (ADL) produce acceleration spikes similar to those of a fall. We collected and presented 2 different types of wearable wrist accelerometers, i.e., the MSBAND smartwatch and the Meta Sensor device. Each device has its own hardware specifications, hence making acceleration datasets produced from these 2 devices differ in many aspects, such as sampling frequency, acceleration unit, axis orientation, etc. Not only are the differences in data between devices a problem, but also, fall data in general is very scarce, as it is very time consuming to collect, leaving us with small datasets across different hardware accelerometers.

In order to overcome the problems detailed above and build a model that is robust to dataset size as well as changes in hardware specifications, we used part of our transfer learning methodology from our previous paper [14], where we would train a base model from scratch using one device's dataset, and then used the trained model as a basis for model personalization on a different device's dataset. Specifically, to solve the target dataset's task, we would not start training from scratch on the target dataset, but we would use a model which has already been trained on a source dataset of a similar (but not identical) feature space to the target data set, and then, building a federated learning framework for model personalization, in which each personalized model, as well as the global model, would utilize the trained model, and have its weights adapt and personalize to the target dataset's subjects, We would have effectively transferred the source dataset's knowledge to the personalized models of the target dataset. And to further add

knowledge to the base models, prior to conducting any evaluation, we generated synthetic data of the target dataset's subjects, and trained each personalized model on the synthetic data, having it effectively pre-adapt to the subjects' data prior to training on any real data.

Indeed, we found out through our experiments, that building personalized models using transfer learning, synthetic data, and federated learning produces better results than our previous benchmark of only using transfer learning, which outperforms training a model from scratch, as the former model out-performed the two latter in the experiments we conducted in the paper. This is encouraging as we can improve the fall detection by adding more senors in a scalable way. There is no need to re-collect a new set of dataset with all the existing sensors and re-train everything from scratch when a new sensor is added. We just need to collect a small amount of data using the new sensor, optionally generate synthetic data to aid in building a better base model, leverage a pretrained model with transfer learning to generalize to the newly sensed data, and finally, add the new model to the personalized federated learning framework. Not only does this achieve very good personalization results, it also autoamatically inherits all the pros of federated learning, such as data privacy. We also experimented with building anomaly detection models using many different anomaly detection algorithms, however, we were not able to achieve any significant results due to the inability of the models of distinguishing between fall and ADL data.

Even though were not able to achieve good results using either model clustering or multitask learning, we still believe that these two approaches have potential, as model clustering still can be improved with better, more detailed user specific data in which better and more accurate clusters can be created, such as exact height and weight, gender, arm length, etc, and multitask learning offers an approach of parameter sharing in which all models have preliminary shared layers, which could

have advantages over global model averaging. Also, even though we managed to achieve good results, we managed to do so on a small dataset, and it would still be interesting to see our framework's performance on bigger datasets with a bigger variety of users.

One immediate direction for future work is the further use of data augmentation method, for further solving the small training dataset problem. Data augmentation method is a process of artificially increasing the amount of data by generating new data points from existing data that does not require substantial training data, including Synthetic Minority Oversampling Technique (SMOTE) [26], Transformers [27], Auto-Encoder [28], Generative Adversarial Network (GAN) [29], and even though our synthetic data usage was successful, it resulted in our model over-fitting, which might indicate that the generated data is basically a copy of the real data.

Finally, we also intend to explore other models, for further improving the accuracy performance of fall detection. Currently, there are many time-series prediction models, such as neural ODEs [30], CT-RNN [31], Phased LSTM[32] and Transformer [33].

APPENDIX SECTION

APPENDIX A: Technologies

- Python 3
- TensorFlow 2.4.1
- Plotly
- Pandas
- Numpy
- Scikit-Learn
- Pyod
- Platform: linux-64

APPENDIX B: Setup

1. Install Python libraries in Technologies (using either Pip or Anaconda)
2. Download MSBAND
   http://www.cs.txstate.edu/ hn12/data/SmartFallDataSet.zip and Meta
   Sensor Datasets (available in the git branch of Spring 2023 linked in section C)
3. Build the model architecture described in figure 4b using Tensorflow
   sequential and the optimal hyperparameters detailed in section IV.III.
4. Sequence the data into blocks as per Window_Size in IV.III.
5. Run Algorithm 1

APPENDIX C: Other Information

- miniconda donwload on linux:
  - wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
  - bash Miniconda3-latest-Linux-x86_64.sh
  - rm Miniconda3-latest-Linux-x86_64.sh
- Code will be available on the Smartfall Services Spring 2023 branch on TXST github, with full data and full instructions on how to run the project https://git.txstate.edu/hn12/Automated_Smartfall_Service/tree/Spring-2023
- For GPU development and model training, Texas State University provides its students with access to the Venus and Ganymede remote servers. The Venus machine is equipped with a NVIDIA GeForce GTX 1080, as of May 2022, and can only be accessed through the Texas State public servers (Zeus and Eros). For ease of development, we recommend working on an up-to-date version of Visual Studio Code, which allows for editing remote files through SSH targets.

REFERENCES

[1] "Falls are the leading cause of death in older americans." https://www.cdc.gov/media/releases/2016/p0922-older-adult-falls.html. Accessed: 2019-6-17.

[2] "Facts about falls." https://www.cdc.gov/falls/facts.html. Accessed: 2019-6-17.

[3] "2017 profile of older americans." https://acl.gov/sites/default/files/Aging and Disability in America/2017OlderAmericansProfile.pdf. Accessed: 2019-9-7.

[4] "preventing falls in hospitals." https://www.ahrq.gov/professionals/systems/hospital/fallpxtoolkit/index.html. Accessed:2919-11-18.

[5] C. Tacconi, S. Mellone, and L. Chiari, "smartphone-based applications for investigating falls and mobility," in 2011 5th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth) and Workshops, pp. 258–261, May 2011.

[6] L. Chen, R. Li, H. Zhang, L. Tian, and N. Chen, "Intelligent fall detection method based on accelerometer data from a wrist-worn smart watch," *Measurement*, vol. 140, pp. 215 – 226, 2019.

[7] "Medical Life Alert Systems." http://www.lifealert.com.

[8] "Mobilehelp smart." https://www.mobilehelp.com/pages/smart. Accessed: 2019-11-18.

[9] "apple watch series 4." http://www.apple.com/apple-watch-series-4/activity/. Accessed: 2019-04-18.

[10] "rightminder - fall detection for android smartwatches and android phones." https://mhealthspot.com/2017/03/rightminder-android-wear-app-seniors/. Accessed: 2022-12-14.

[11] T. R. Mauldin, A. H. Ngu, V. Metsis, and M. E. Canby, "ensemble deep learning on wearables using small datasets," ACM Trans. Comput. Healthcare, vol. 2, Dec. 2021.

[12] T. R. Mauldin, M. E. Canby, V. Metsis, A. H. Ngu, and C. C. Rivera, "smartfall: A smartwatch-based fall detection system using deep learning," Sensors, vol. 18, no. 10, 2018.

[13] N. Seraji-Bzorgzad, H. Paulson, and J. Heidebrink, "neurologic examination in the elderly," Handbook of clinical neurology, vol. 167, pp. 73–88, 2019.

[14] N. Maray, A. H. Ngu, J. Ni, M. Debnath, and L. Wang, "transfer learning on small datasets for improved fall detection," Sensors, vol. 23, no. 3, 2023.

[15] K. Yang, W. Yang, T. Zhang, Z. Li, Q. Liu, and M. Hua, "personalized federated learning for mobile devices: a reinforcement learning approach," in Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2369–2379, 2020.

[16] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, J. Cao, G. Diamos, P. Emma, *et al.*, "advances and open problems in federated learning," Foundations and Trends in Machine Learning, vol. 14, no. 3, pp. 239–390, 2021.

[17] Y. Wu, J. Liu, J. Zhang, and J. Zhou, "personalized privacy-preserving federated learning with non-iid data," IEEE Transactions on Information Forensics and Security, vol. 16, pp. 2173–2186, 2021.

[18] Y. Zhang, S. Cao, Z. Liu, Z. Wu, J. Wang, and H. Xiong, "personalized federated learning with differential privacy," in Proceedings of the 28th ACM International Conference on Information and Knowledge Management, pp. 1313–1322, 2022.

[19] Y. Li, Z. Liu, Q. Liu, W. Cao, and J. Guo, "aggregation strategies in personalized federated learning," IEEE Transactions on Cognitive Communications and Networking, vol. 7, no. 3, pp. 844–858, 2021.

[20] J. C.-W. Lin, E. J. Keogh, L. Wei, and S. Lonardi, "anomaly detection in time series sensor data for machine health monitoring," IEEE transactions on knowledge and data engineering, vol. 22, no. 4, pp. 507–518, 2010.

[21] V. Chandola, A. Banerjee, and V. Kumar, "anomaly detection in time series data: A survey and evaluation," Data mining and knowledge discovery, vol. 29, no. 1, pp. 18–61, 2015.

[22] P. J. Rousseeuw, M. Hubert, and K. Van Driessen, "detecting anomalies in time series data via minimum covariance determinant estimation," Journal of Machine Learning Research, vol. 19, no. 94, pp. 1–31, 2018.

[23] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "iot middleware: A survey on issues and enabling technologies," IEEE Internet of Things Journal, vol. 4, no. 1, pp. 1–20, 2016.

[24] A. H. H. Ngu, J. S. Eyitayo, G. Yang, C. Campbell, Q. Z. Sheng, and J. Ni, "an iot edge computing framework using cordova accessor host," IEEE Internet of Things Journal, vol. 9, no. 1, pp. 671–683, 2022.

[25] X. Li, V. Metsis, H. Wang, and A. Ngu, TTS-GAN: A Transformer-Based Time-Series Generative Adversarial Network., vol. 13263 LNAI of *Lecture Notes in Computer Science*. Texas State University: Springer Science and Business Media Deutschland GmbH, 2022.

[26] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "smote: synthetic minority over-sampling technique," Journal of artificial intelligence research, vol. 16, pp. 321–357, 2002.

[27] V. Kumar, A. Choudhary, and E. Cho, "data augmentation using pre-trained transformer models," arXiv preprint arXiv:2003.02245, 2020.

[28] I. Kuroyanagi, T. Hayashi, Y. Adachi, T. Yoshimura, K. Takeda, and T. Toda, "anomalous sound detection with ensemble of autoencoder and binary classification approaches," tech. rep., DCASE2021 Challenge, Tech. Rep, 2021.

[29] G. Mariani, F. Scheidegger, R. Istrate, C. Bekas, and C. Malossi, "bagan: Data augmentation with balancing gan," arXiv preprint arXiv:1803.09655, 2018.

[30] P. Kidger, J. Morrill, J. Foster, and T. Lyons, "neural controlled differential equations for irregular time series," Advances in Neural Information Processing Systems, vol. 33, pp. 6696–6707, 2020.

[31] R. Hasani, M. Lechner, A. Amini, D. Rus, and R. Grosu, "liquid time-constant networks," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 7657–7666, 2021.

[32] Y. Liu, C. Gong, L. Yang, and Y. Chen, "dstp-rnn: A dual-stage two-phase attention-based recurrent neural network for long-term and multivariate time series prediction," Expert Systems with Applications, vol. 143, p. 113082, 2020.

[33] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan, "enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting," Advances in neural information processing systems, vol. 32, 2019.