An IoT System for Automated Fall Detection in Smart Homes

Benjamin Posnick, *bmp53@cornell.edu*
Arjun Bhalla, *ab2383@cornell.edu*

Cornell University
Dept. of Computing & Information Science
CS 5412: Cloud Computing and Smart IoT Systems

INTRODUCTION

A pervasive issue in elderly care is the potential for elderly persons living alone to experience a fall, thereby injuring themselves. In these situations, immediate detection and rapid response times are key. Therefore, after talking with Dr. Ashutosh Saxena of Caspar.AI, we pivoted from our original idea of building a home security system, to one of a fall detection system for a smart home. This proposal originally involved using thermal sensors placed around the home, conducting some initial processing of the raw data on the in-home IoT Device, passing parameters to the cloud to do the "heavier lifting" (so as not to violate privacy by passing raw data), and then returning whether or not the subject has fallen over – taking action as required (i.e. sending an email to a caretaker).

We have achieved this goal by creating a fully-functional system, with the slight caveat that the system learns from accelerometer sensor data instead of thermal images due to an issue procuring labelled thermal images. We begin by explaining the dataset, feature extraction, and the machine learning model used in classifying whether or not a fall occurs. We then explain the system architecture end-to-end, explaining the design choices made along the way to (1) improve latency, throughput, and scalability, and (2) achieve availability, consitency, and fault tolerance – including a quantitative analysis of the effects of these choices.

MACHINE LEARNING

Data Preprocessing and Feature Engineering

We used the "Localization Data for Posture Reconstruction" dataset from Kaluza *et al.*[1] This dataset consists of records of the x-, y-, and z-position of multiple sensors placed on a human subject while they engage in various daily activities. Each subject wore four sensors during data collection, with one on each of their: left ankle, right ankle, belt, and chest. Given that the raw positional data of a single sensor at any given point in time may not be enough to learn from, we chose to conduct feature engineering to improve the quality of data used for inference.

We read in from the dataset (which is sorted in order of the arrival time of the sensor data) and use every consecutive group of three data reports to construct a single example. Sensor data does not arrive in uniform time intervals and there is no guarantee of order. This unpredictability is not only present in the dataset, but is also a reality for a system in production, given the potential for network delays and sensor failure. Thus, we sought to create features that captured the variability in the data and to develop a machine learning model that is robust enough to learn from data that is not guaranteed to be clean nor predictable.

Table 1 and Table 2 depict how an example is created from three consecutive sensor data reports. From this set reports, we extract the individual x-, y-, and z-positions; the mean x-, y-, and z-positions; the number of reports in the given example that come from sensors 0, 1, 2, and 3 (where this mapping from sensor ID to integer is depicted in the codebase); and the trajectory of the set of x-, y-, and z-positions. We define trajectory as follows, where a, b, and c are three data points from the same dimension (i.e. x, y, or z) such that a happened before b and b before c:

$$\text{trajectory}(a,b,c) = \begin{cases} 1 & \text{if } a < b \text{ and } b < c \\ 0 & \text{if } a < b \text{ and } b > c, \text{ or } a > b \text{ and } b < c \\ -1 & \text{if } a > b \text{ and } b > c \end{cases}$$

Table 1: Raw sensor data. Note that UID has been omitted because it is not useful in feature extraction.

Person ID	Sensor ID	UID	Timestamp	X-Position	Y-Position	Z-Position
A01	020-000-033-111	*	27.05.2009 14:03:25:183	4.2919535636901855	1.7811404466629028	1.3444952964782717
A01	020-000-032-221	*	27.05.2009 14:03:25:210	4.359101295471191	1.8264557123184204	0.9688209891319276
A01	010-000-024-033	*	27.05.2009 14:03:25:237	4.0878353118896475	1.8799986839294436	0.4669834077358246

[1]B. Kaluza, V. Mirchevska, E. Dovgan, M. Lustrek, M. Gams, An Agent-based Approach to Care in Independent Living, International Joint Conference on Ambient Intelligence (AmI-10), Malaga, Spain, In press

Table 2: An training/testing example constructed through feature extraction using the data in Table 1.

Sample 1 ID	**Sample 1 X**	**Sample 1 Y**	**Sample 1 Z**
2.	4.29195356	1.78114045	1.3444953
Sample 2 ID	**Sample 2 X**	**Sample 2 Y**	**Sample 2 Z**
3.	4.3591013	1.82645571	0.96882099
Sample 3 ID	**Sample 3 X**	**Sample 3 Y**	**Sample 3 Z**
0.	4.08783531	1.87999868	0.46698341

# Sensor 0	**# Sensor 1**	**# Sensor 2**	**# Sensor 3**
1.	0.	1.	1.

X Trajectory	**Y Trajectory**	**Z Trajectory**
0.	1.	-1.

Mean 3 X	**Mean 3 Y**	**Mean 3 Z**
4.24629672	1.82919828	0.92676656

Classification Model

We formulate this as a supervised learning problem, in which the data belong to one of eight classes:

- **0** = Walking
- **1** = Falling
- **2** = In the process of lying down
- **3** = Lying
- **4** = Sitting
- **5** = In the process of sitting down
- **6** = On all fours
- **7** = In the process of standing up

When deciding how to label these newly formed examples based on the original raw training data, we chose the most frequent label from the set of three records. This works well in practice because the rate between sensor records is less than 0.1 seconds, and transitions from one activity to another are not abrupt (e.g. given consecutive sensor records labelled ”falling”, ”falling”, ”lying”, the example would be labelled ”falling” and one would not expect three different labels in such a short timespan).

We use a random forest classifier for classification of our data. It consists of 256 individual decision trees, each using information gain as the splitting criterion and having a maximum tree depth of 32, so as to minimize overfitting – a common pitfall of decision trees. Since random forests are a bagging classifier, the variance of the individual decision tree models is reduced. Given the unpredictability of the data coming in from sensors, such an ensemble method is well-suited for generalizing in this scenario. The test accuracy of this model is 68.05% – table 3 further breaks this down into accuracy by-class. Note that the size of the training dataset after feature extraction was relatively small (≤ 1250 per class) and was further limited by the uneven distribution of labels in the dataset.

Table 3: Accuracy of random forest classifier by-class, where class-to-integer mapping was defined above.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
66.43%	63.11%	62.32%	65.56%	71.33%	68.93%	92.19%	51.61%

SYSTEM DESIGN

System Architecture and Information Flow

For purposes of demonstration, we simulate receiving data from sensors by reading in raw data from text files. The raw data is processed on the IoT Edge device, which is a Linux virtual machine (running Ubuntu Server 18.04) to represent the controller in the proposed smart home. Each set of three consecutive sensor records is used to create examples in real-time in the previously described format. Inference is able to be done on these individual examples by our machine learning model, which is deployed as an Azure Container Instance and reachable via HTTP requests

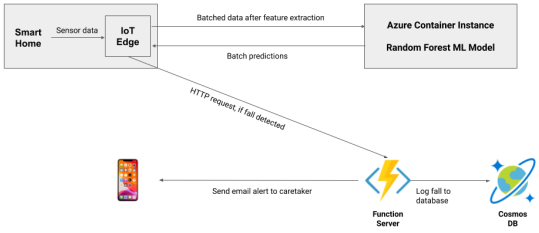

Figure 1: End-to-end system architecture.

to a REST service endpoint. Since we are sending sensor data as opposed to images or videos to the cloud, we are mitigating potential privacy concerns associated with data that is more sensitive than what the sensors provide.

To improve the throughput of our system and reduce the overhead of network delays, we batch our examples into a batch size of N – where N can be changed at ease (see "Performance Testing" for detailed analysis of optimal batch size). This allows inference to be conducted on N examples per POST request to the model endpoint, rather than just 1 example per request. This has the benefit of being able to leverage processor caches and linear algebra optimizations in the machine learning libraries, which aid in improving the speed of the system.

Once a batch of predictions returns from the model endpoint, the Edge device searches for a sequence of predictions in which the person transitions from falling to lying. The rationale is that there could be misclassification of single examples (i.e. false positives) that would otherwise cause alerts to be sent to a caretaker that the smart home's resident has fallen. By searching for a sequence of predictions, we (1) are able to know that the person transitions from a falling to lying state, which is more indicative of a fall than someone simply tripping (which may otherwise be classified as a fall and cause a false alarm), and (2) make our system more robust to potential of false positives.

If this sequence is found, an HTTP request is sent to the function server. After first checking that the message is well-formed, the function server will do two things. First, it initiates an SMTP server instance to send an email to the address specified in the HTTP request, informing the recipient that someone has fallen in that home. The second action it will take is to initiate a connection to a CosmosDB database on Azure and then log the time and location apartment number of the fall.

Analysis of System Performance, Scalability, and Adherence to C.A.P. Principles

IoT Edge and the Function Server

All stream updates from each home are sent to the central IoT Hub. This is easily scalable, as the code is stored in a container registry, which we can then replicate on new IoT devices as and when they are spun up. They are not necessarily fault tolerant, as they are not replicated anywhere; however, they do have the ability to run offline in limited capacity, which can be effective to provide an alert to the resident to fix the system. The function server is fault tolerant and highly-scalable as it uses the default serverless function in Azure.

Azure Container Instance Endpoint

The ACI endpoints are configured to handle loads ranging from a few up to 200 concurrent requests. Given that we are not currently deploying this application to more than 200 simulated smart homes, one endpoint suffices. However, our system is built in a modular fashion such that deploying more endpoints (using the same inference model for each) is a trivial task. The load could be balanced across these endpoints by having each smart home choose one of K endpoints uniformly at random each time it requires another batch prediction. The model endpoint is hosted by a virtual machine running on an NVIDIA Tesla K80 GPU with 56GB RAM. To ensure fault tolerance, the Azure infrastructure will automatically move the virtual machine that facilitates our endpoint to another physical node in the datacenter in the event that the original node crashes due to a hardware failure.[2]

[2]https://docs.microsoft.com/en-us/archive/msdn-magazine/2015/september/microsoft-azure-fault-tolerance-pitfalls-and-resolutions-in-the-cloud

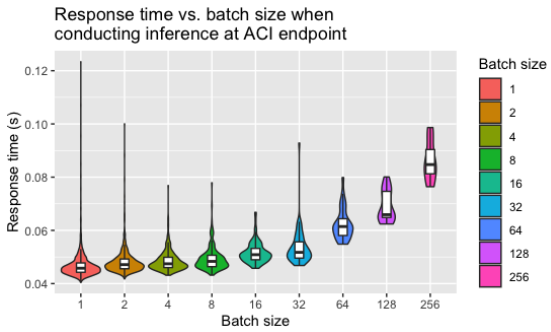

Figure 2: Violin plots depicting the distribution of response times for various batch sizes.

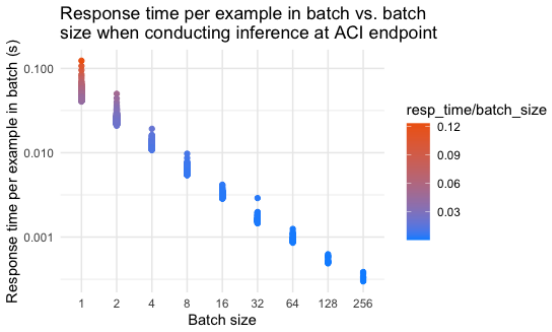

Figure 3: Scatter plot depicting the response time per example in a batch of a given size for various batch sizes.

Database

CosmosDB provides global availability of our data (99.999% availability of reads and writes) and latencies of indexed reads and writes of less than 10ms. CosmosDB also supports elastic scaling of the database's throughput and storage, giving our application the ability to scale with the number of requests. The database we employ has a maximum throughput of 4000 request units per second (RU/s). (One request unit is equivalent to reading 1KB of data, and 10 RU/s are required to write 1GB of data.[3]) The database uses the apartment number as the partition key so as to distribute storage in a logical and relatively even manner.

The database is configured to use a consistent prefix level of consistency. Such a consistency guarantee helps to provide low write latencies, high availability, and high read throughput similarly to eventual consistency guarantees, while also preserving the order of writes.[4] Write order is important because if the fall is serious, we want to ensure that whomever is querying the information in the database (e.g. potentially a 911 operator) can react in a manner aligned with the ordering of the writes, which represent the order of the falls. Stronger consistency guarantees would allow all nodes to see the same data at the same time, at the expense of higher latency. However, in a potential life-threatening situation, lower latencies are preferred, even if that means some nodes see writes before others.

Performance Testing

Response Time vs. Batch Size when Conducting Inference at ACI Endpoints

We define the response time as the time for a batch of predictions to be sent from the IoT Edge to the ACI endpoint, classified by our model, and returned to the Edge. In figure 2, we show the distributions of response times for various batch sizes. Note that smaller batch sizes generally experience greater variance in response time, whereas large batches have few (if any) outliers. We also see that the mean response time tends to increase as batch size increases. This is an intuitive conclusion, given that a larger batch implies more computation, as well as large message sizes. However, figure 2 does not tell the whole story.

In figure 3, we measure the response time per example in a given batch as a function of the overall batch size. We see that as the size of the batch increases, the response time per example tends to logarithmically decrease. Tying together the information from both figures, it is clear that although larger batch sizes require longer response times, they are capable of completing requests faster on a per-example basis. Thus, since an event like falling generally takes a few seconds to occur, smaller batch sizes may not be necessary because we are not operating in a domain where seconds matter – more so on the scale of minutes. Thus, larger batch sizes will reduce the overhead and load on our system, allowing for better scalability as more homes are added to the system.

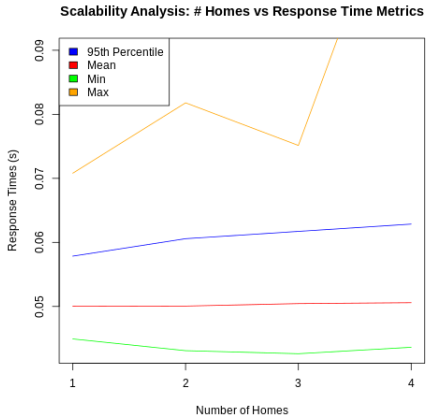

Figure 4: The maximum, minimum, mean, and 95th percentile response times for varying numbers of simulated smart homes concurrently communicating with the same ACI endpoint for inference using batches of size 10.

Scalability Tests

We attempted to run experiments within the scope of our project – subjected to the amount of Azure credits allocated to us – to determine the scalability of the system that we built. Figure 4 shows various response time metrics (as defined in the previous subsection) against the number of homes simultaneously querying the ACI endpoint.

We can observe that while the max time varies significantly (alluding to the idea of response times being modelled by a Zipf distribution, as discussed in lecture), the mean time is nearly unchanged, with a variation between 1 home and 4 homes being ~ 0.001s. This is likely explained by how Azure has designed the ACI, being able to handle up to 200 concurrent requests and thus on average not affecting the response times of the homes at this scale. (Unfortunately, we did not have enough Azure credits to spin up 200 instances of virtual machines running separate IoT Edge devices.) Moving to the next metric, we see that the times for the 95th percentile exhibit logarithmic behaviour. This could likely be attributed to the manifestation of the increased server load that the ACI endpoint would be facing.

CONCLUSION

Overall Evaluation

In our project plan, we set out to evaluate and test our system around four major pillars: unit testing, machine learning model accuracy, performance testing, and integration testing. Given this, we have succeeded unequivocally in achieving this goal, as we built a robust, privacy-protecting, scalable system to rapidly process and conduct fairly accurate inference on streamed data to determine whether or not a person has fallen in their home.

There were numerous difficulties that were overcome due to unforseen circumstances such as obtaining data, credit limits, virtual machine set up – just to name a few. Despite these challenges, we produced a functional and performant system, as evidenced by the previous sections, which is modular enough that it can be altered with ease (e.g. changing the form of data used for inference).

We find that while the speed of the system is fast and the architecture is scalable, until it begins computing on a large scale (perhaps in the thousands of homes), we cannot tell with certainty what our upper-bounds are on scalability. It is important to note that our code could have been further optimized for performance - the choice to use Python as a language was due to its ubiquity within the world of machine learning and accesibility within the Azure IoT ecosystem. If we were to revise our system in the context of multiple IoT devices in the same home, perhaps Scala, C#, or C++ would be better-suited for performance. Design-wise, however, decisions were made to ensure that this architecture struck a balance between the efficiency of the system and the subject's privacy.

[3]https://docs.microsoft.com/en-us/azure/cosmos-db/request-units
[4]https://docs.microsoft.com/en-us/azure/cosmos-db/consistency-levels

Future Work

Of course, there is still much that could be done to improve and expand upon the system that we have, including the following:

- Switch to a more performant language
- Collect thermal image data, update the Edge to read in this image feed, and design a machine learning model to process such data to better improve the accuracy of inference
- If given access, see the ecosystem in which this would live (e.g. a Caspar home) to get a better idea of how this would interplay with the rest of the smart home and adjust design choices accordingly
- Generalize and apply this architecture to other use cases (although our code and design is structured so that this is fairly simple to do), such as home security