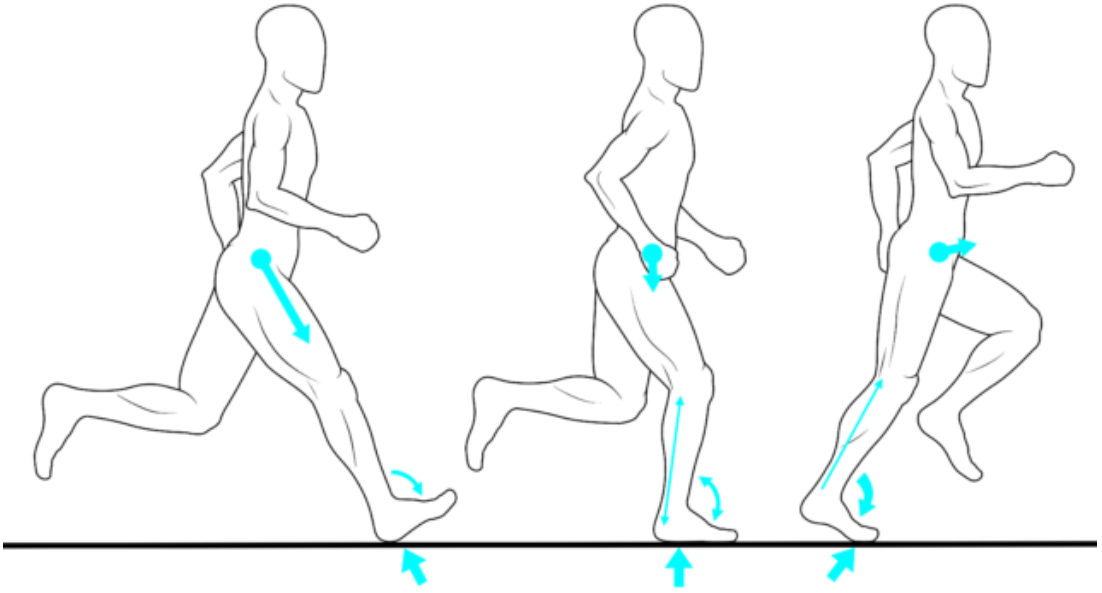
Dan Powell and Chris Held
IS 6850-002 Spring 2025

Observing the Unobservable: A Pipeline for Running Gait Analysis



Completed by Chris Held and Dan Powell
IS 6850-002 Data Engineering
Spring 2025

Project Introduction and Methodology

Our project originally aimed to address shortfalls in running gait analysis. We chose this subject area due to a recent episode of Data Skeptic, where a biologist discussed attaching IMU devices to small animals to predict migratory patterns and other unobservable behaviors. After understanding how an IMU device operated, we felt that it could also be applied to athletics - specifically to gait analysis during running. Currently, running gait analyses often rely on the subject to be on a stationary device like a treadmill, which can alter the subject's gait due to the difference in biomechanical demands of a treadmill versus running on outdoor surfaces. Additionally, the subject's increased awareness of their gait patterns while being watched and the observer's potential bias could further augment the conclusions drawn from traditional gait analysis techniques. However, streaming data from IMU devices during outdoor runs and enriching it with metrics related to exertion and training load could offer insights into a runner's injury risk that go beyond what a single gait analysis session can provide.

With this goal in mind, we then constructed a data pipeline that would stream in data from IMU (inertial measurement unit) devices while making batch API calls to the sports social media site, Strava, for information on training load, exertion, and summary statistics surrounding a particular activity. The streaming data from the IMUs would be received as a three-column dataset with X, Y, and Z coordinates corresponding to where the device was in space. As the data is being streamed in, utilizing PySpark, we would aggregate the data in sliding windows of 60 seconds to observe abnormalities in gait. These abnormalities would be calculated using pre-trained machine learning models or a simple threshold using standard deviation for each of the planes (X, Y, and Z). When a sustained gait abnormality is detected using the aggregated time windows, the corresponding window is stored in a temporary in-memory file, and a simulated

message—like one that would be sent to a wearable device—is triggered to alert the user that their gait may need correction. Finally, the streamed data would be appended to a dataset in temporary storage until streaming was complete. Once complete, the two streaming datasets, the overall aggregated dataset and the separate abnormality windows dataset, would be moved to permanent storage in Google BigQuery.

Separately, we would pipeline data from Strava's API and transform it into a tabular format in storage. This would be done using a pull extraction method, meaning that data would only be refreshed when requested from the system. Once the API was called, JSON data with summary information on each activity would be returned. Then, once in memory, we could convert it into a pandas dataframe and then load it into more permanent storage in Google Cloud Storage for serving.

Once all data had been ingested and stored, we intended to utilize the three main datasets: abnormality data (windows from streaming data with consistent abnormal gait patterns), aggregated streaming data, and summary data from Strava's API. We then considered the possibilities of potentially training an ML model to recognize patterns of gait abnormalities for prediction and prescriptive purposes and enriching the summary data with information from the streaming data. This model would then be served primarily to an audience of kinesiologists, physical therapists and running enthusiasts to illuminate where injury risk may be increasing and enable interventions to keep the athlete healthy.

Figure 1 shows a flowchart of how we intended to construct the pipeline. However, as we will elaborate on in the next section, we did need to make some minor adjustments. The next section is divided into 5 stages: Extraction, Transformation, Streaming, Loading, and Analysis to reflect each step of the construction of our pipeline.

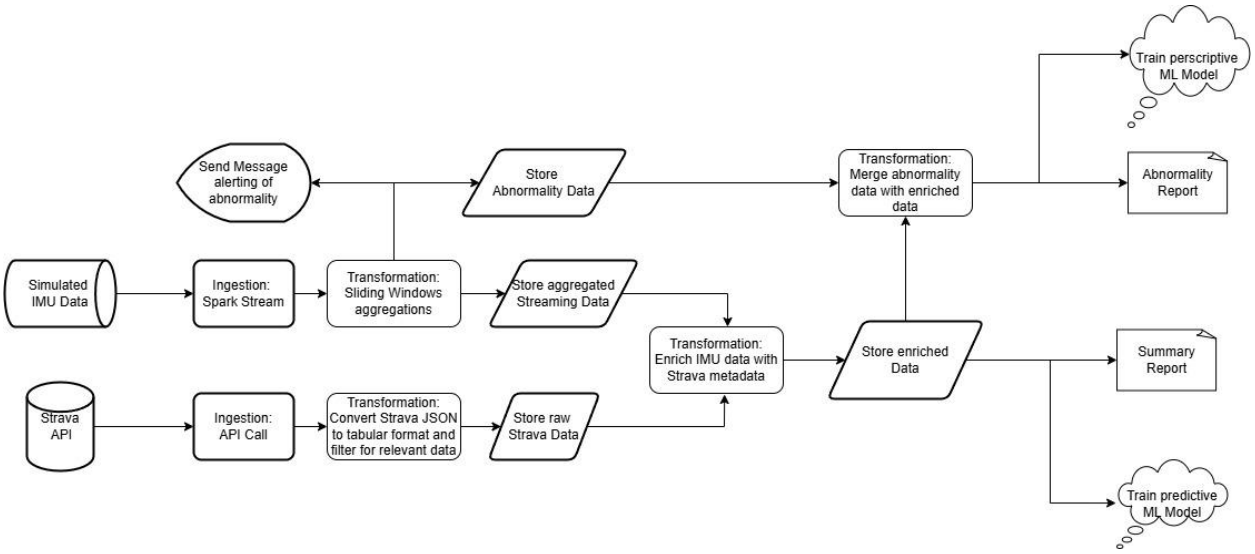


Figure 1: Initial Infrastructure Design

Project Work Review

Our infrastructure did evolve once we began implementing our project. Figure 2 below shows the structure of the pipeline after we had finished implementing all portions of the project.

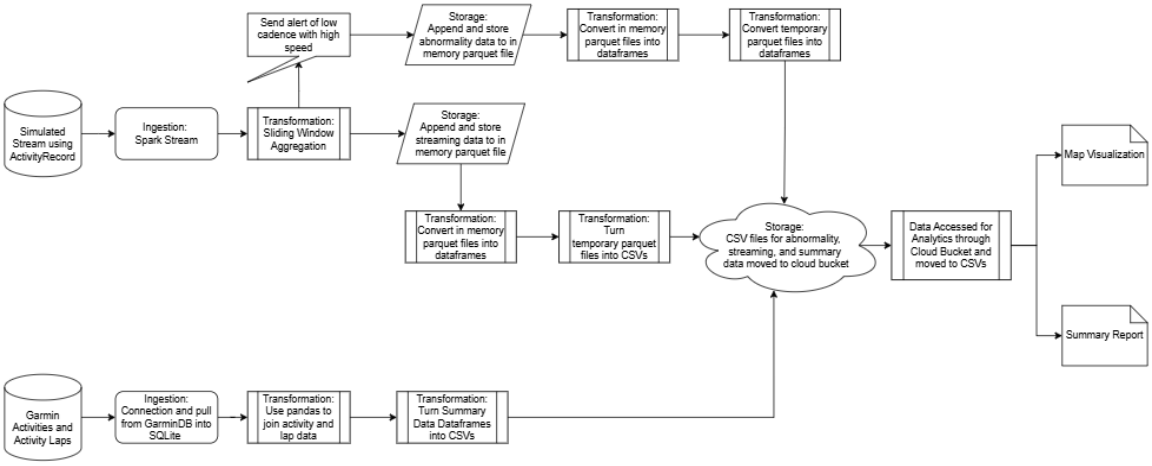


Figure 2: Final Pipeline Infrastructure

As can be noted, the pipeline has changed from the original format but has maintained its overall structure. This was due to a few adjustments that needed to be made in development, which we will elaborate on next.

Extraction of Data: Strava API, IMU JSON files, and GarminDB Package

Of all the processes displayed above, the most challenging portion of our project was importing the data and locating reliable sources for our data pipelines. The first challenge we ran into was with the Strava API. While searching for a video on how to import data from the Strava API, we came across a YouTube video discussing Strava's recent changes to their terms of service. Specifically, Strava no longer allows users to export data for "analytics." Not wanting to violate the terms of service, we turned our attention to the IMU data while pondering other sources. We were able to locate IMU data without purchasing hardware through a kinesiology study that had posted .json files for use via FigShare. While we were able to ingest these, we were left with the problem of the summary data. Now, even if we were able to pull the data, we would be unable to gather any relevant insights by connecting the summary data to the IMU data. This is because the IMU data was from a study participant and the summary data would be from one of our own Strava Accounts, making it impossible to associate events across datasets to create meaningful insights.

This is when we discovered GarminDB. Garmin DB is a python package that allows you to access your Garmin Data by providing JSON with credentials and parameters in a temporary storage file. This is then utilized by GarminDB to acquire your information from Garmin and parse .fit files, which are a unique file type Garmin utilizes to store activity information. Through GarminDB, three data sets became available to us: ActivityRecords, Activities, and ActivityLaps (see Table 1 descriptions). Each of these datasets were presented as objects, each of which

contained a .get() method, where you could provide the ID of an activity and retrieve attributes of the dataset through attribute calls (ex. activity.max_hr returns the maximum heart rate for that activity). This was different than expected, as usually API data is returned in .json format.

Granularity	GarminDB Dataset	Short Description
Low	Activities	Contained summary information for the whole activity. Included Columns such as max and average heart rates, max and average speeds, total distance, and pre-aggregated metrics.
Middle	ActivityLaps	Returns the laps for each activity, which is any time the lap button is hit on the garmin device or a mile is elapsed during a run. Contains aggregate information at the lap level, such as average and max speed, average and max heart rates, and other pre-aggregated data.
High	ActivityRecords	Returns every 3 to 5 seconds of an activity in snapshot format. Has details on heart rate, longitude and latitude, cadence, distance, speed, and timestamps.

Table 1: GarminDB Dataset Descriptions

Transformation and Storage: SQLite and GarminDB

Since the data was only accessible through attribute calls, we utilized a for loop to load the data into SQLite databases. First in this process, we used list comprehension to get each of the activity_ids for the last 30 days from the Activity dataset.We did this by using the .get_all() method that would call all activities within the timeframe specified by the configuration file mentioned earlier. Then, we used these IDs in a for loop to load the data from the GarminDB object into three SQLite tables, one for each of the datasets described in the table above. Once this is done, we could join the tables for ActivityLaps and Activities to give us a lap-level view with summary statistics for the full activity. We also pulled from the new ActivityRecords table and created a csv that was utilized by PySpark to simulate the data stream.

Streaming: PySpark

In the PySpark stream simulation, we found that we could send one row at a time and use the sleep method from the time package to simulate a five second delay between each record. However, the activity we were interested in simulating was an hour and a half long run, which means the streaming simulation would take just as long to run. Instead, we switched to having each batch come into the PySpark stream function ten rows at a time. To achieve this, we created an inner function that would create a temporary csv with each batch of ten records and then have spark read that csv using the spark.read.csv method. We then pass the batch in spark dataframe form to an outer function that processes each batch. In the outer function, we specify our sliding window duration (1-minute) and the slide duration (5-seconds). We then converted the timestamps to unix to define the window specifications with our specified duration. We then union the batch to the rest of the stream and then calculate the average speed for each sixty-second window as a demonstration of inflight aggregations with streaming data. We also set up a messaging alert to print a message any time the cadence drops below a specified level (85) and the speed is above a certain level (8 miles per hour) for a batch. As part of the conditional that detected the abnormality in the batch, we also union the batch to a parquet file in temporary storage holding all the abnormality data. Finally, we append the aggregated data to its own data store, which is also a parquet file in memory. At the end of the streaming simulation, we have two parquet files with data, one that contains the batches with abnormalities in gait and speed, and the other contains the streaming data with sixty-second window aggregations for speed. We then load the parquet files back into spark dataframes and then into pandas dataframes to ensure all data has loaded as expected. In a fully-realized data pipeline with actual streaming data, we

would likely publish the parquet files directly to our permanent storage while streaming instead of temporary storage.

Loading: Google Cloud Storage

At this stage, we now have three datasets in memory, ready to be sent to more permanent storage (Abnormality Streaming Data, Aggregated Streaming Data, and the summary data). For permanent storage, we utilize a Google Cloud bucket. Here we create a client and a bucket to hold our data for analysis later. Each file is then given a blob and then uploaded to the blob to be stored in Google Cloud. Once completed, we have three CSVs stored in the cloud corresponding to each of the datasets we created. While the process was performed manually for this project, a production-grade implementation would use an orchestration tool such as Airflow to automate appending data from the two streaming datasets to CSV files in cloud storage. Additionally, data from the Garmin API would be programmatically retrieved, used to construct the summary table, and then appended to the existing CSV files in Google Cloud. However, since we did not want to make repeated API calls through GarminDB, we decided to keep the process manual for this project.

Analysis: Running Map and Bar Plot

Finally, our last stage was analysis. For the first analysis, we pulled the aggregated streaming data back from Google Cloud Storage and into a Google Colab notebook. We then used every ten observations' longitude and latitude to place points onto a folium map, which is a python package for making map visualizations. We then used a simple mean to get the speed, heart rate, and cadence recorded between two points. This metric is then displayed on a tool tip that appears when the user clicks the line. We also used color coding on the lines to indicate

faster speeds, higher heart rates, and faster cadences. Finally, we applied 3 layers - one for each of the metrics we utilized (heart rate, cadence, and speed) to allow users to toggle between metrics on the map. This allows users to view a map of an activity and investigate areas of interest in regard to cadence, heart rate, and speed (See figure 3).

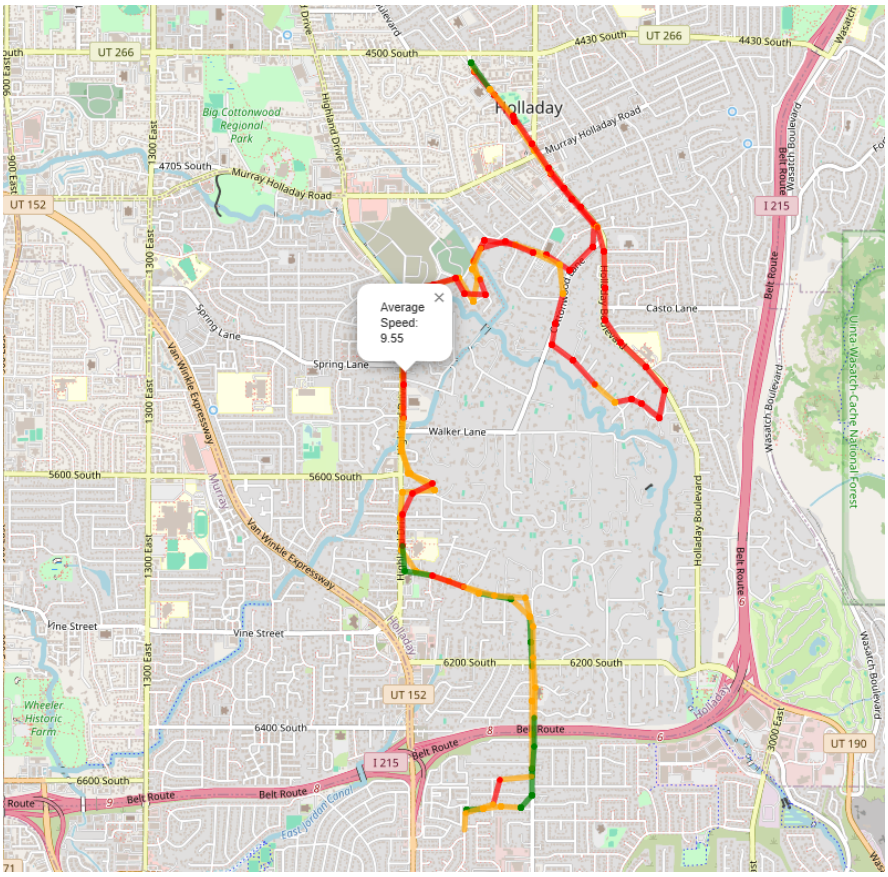


Figure 3: Output from the map visualization

Our other visualization was a more standard analysis showing maximum heart rate by date over time. This involves little transformation on the part of the analyst, as they simply need to load in the data from the Google Cloud Storage Bucket, truncate the date time column into a single date, and then create the visualization using matplotlib or another software. Figure 4 shows a bar plot we produced to demonstrate a visualization of maximum heart by date over the course of 3/17/2025 to 4/3/2025.

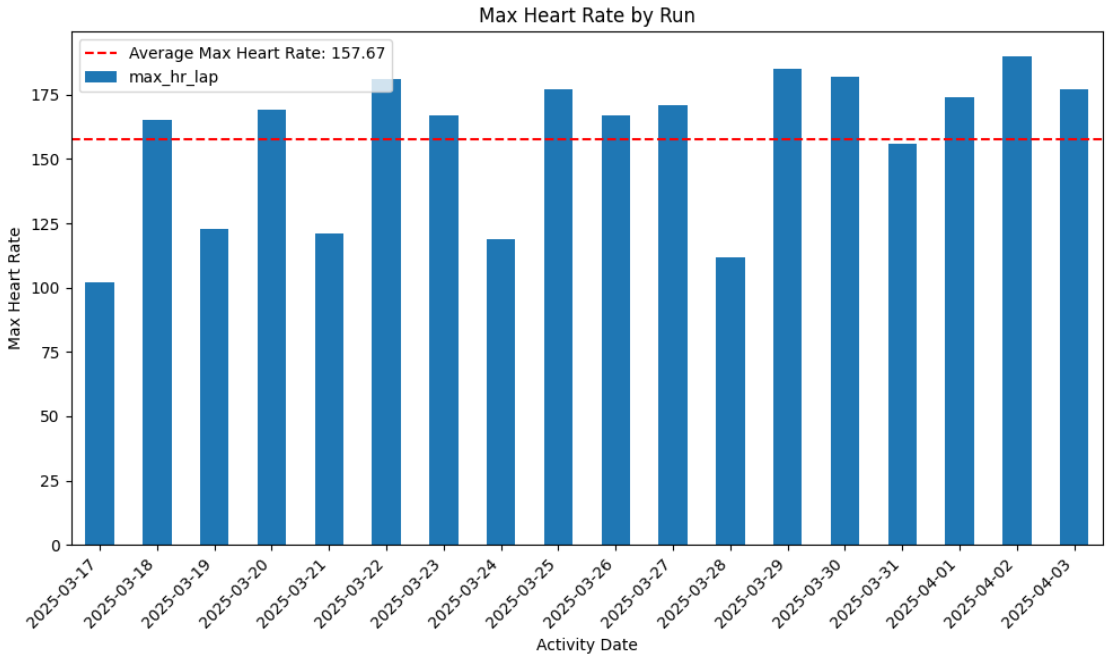


Figure 4: Maximum Heart rate over time with an average max heart rate.

Retrospective: Lessons Learned and Future Direction

To begin with a more positive note, a lot did go right with this project. Despite the roadblocks with the Strava and IMU data, we were still able to simulate streaming via the GarminDB package. To elaborate, the GarminDB package provided both data that we could use to simulate streaming data (ActivityRecords dataset) and data very similar to what we expected to acquire from Strava (Activities and ActivityLaps datasets). Good program management was shown on the part of both group members as duties were split when the roadblock was encountered. Specifically, Chris focused on trying to see if we could still utilize the original data, while Dan investigated alternatives. This allowed for optimal use of time, as Dan eventually turned up the Garmin data which we ultimately used rather quickly once discovering its capabilities.

However, there was still room for improvement with our project. As mentioned before, a realized application would likely implement orchestration software (ex. Airflow and/or Kafka) to automatically pipeline the data into Google Cloud and append it with the existing datasets. Additionally, we did have to cut the machine learning models we initially planned to implement simply due to complexity and time constraints. While an abnormality detection model could be constructed using any variety of supervised classification algorithms, we decided instead to focus our attention on creating the running map. We believed this would have more of a "wow" factor on our audience than a complicated machine learning algorithm as it is interactive and demonstrates the value of the streaming data. Regardless, we still produced the bar plot to demonstrate that critical visualizations could be created for analysis utilizing the summary data we pipelined. Finally, and perhaps a somewhat uncontrollable factor, we had to change our project's original aim to utilize IMU data to address gait issues to one that utilizes more basic metrics like cadence, speed, and heart rate. While there is plenty of research to justify that these more basic metrics may be enough to predict injury risk, as some studies have shown a relationship between cadence and effort with injury risk, we still would have liked to pursue this novel avenue of gait analysis and injury prediction. However, the time required to learn how to implement and understand IMU data was not feasible for this project.

Even so, we learned many technical skills from this project. The use of folium opened up doors for additional applications of streaming data with longitude and latitude to create maps for analysis in the future. It also has inspired us to look more into how IMUs work, and potentially to invest in some as a hobby project to continue to build out this project. With the infrastructure we have built so far, we believe that it may not be infeasible to alter our current code to support the original intended project with some adjustments, including utilizing Garmin's data instead of

Strava's and conducting additional research on how to extract data from IoT devices. Additionally, we became more comfortable with packages such as PySpark which we utilized to simulate streaming data, create micro-batches, and create in-flight aggregations and alerts. Finally, pivoting from individual record ingestion to micro-batches in our streaming data demonstrated two fundamental methods of how streaming data can be extracted, especially in cases where perhaps there is a poor internet connection, or the device sends multiple observations in at predetermined intervals. Overall, this project not only expanded our technical abilities but also sparked ideas for future innovation in wearable health tech.