IMPERIAL

DESE71003 – Sensing and Internet of Things

Imperial College London

Dyson School of Design Engineering

Luma

Author:
Sophie Britz (CID: 02235059)

Code & Data URL:
https://github.com/sophiebritz/Luma

Coursework Video URL:

`https://youtu.be/-_lCCfsrfOs`

Date: October 10, 2025

Contents

1. Introduction

In response to growing urban congestion, noise pollution and rising CO_2 emissions, many cities are shifting towards zero-emission micro-mobility solutions [1]. The rise of commuter e-bikes and shared services such as Lime has made cycling more accessible and convenient [2]. With nearly half of 18–34-year-olds using them for weekly commutes, safety remains a significant concern [3]. Despite, 60% of London cyclists wearing helmets, the highest proportion in Europe, amongst shared e-bike users helmet usage drops to 10% [4]. Seasonal factors such reduced daylight hours and worse road conditions further increase the risk of cycling accidents [5]. In addition, shared e-bikes often lack automatic brake lights systems, and frequently suffer from mechanical faults with approximately one in twelve bikes reported to have defects [6].

Cyclists need smart solutions, visibility, brake and crash detection all conveniently in a helmet. Current smart helmets (Lumos £200, Livall £130) are expensive, and their existing crash detection systems produce false alarm rates exceeding 20% [7]. This highlights a clear gap in the market for an affordable, safe and reliable helmet solution.

The aim of this project was to:

'Design a smart cycling helmet system that automatically enhances rider visibility and safety'

Based on this research and project aim I developed Luma an Internet of Things (IoT) system integrating heterogeneous time-series data sources to provide 4 main functions with subfunctions, seen in Table 1.

Table 1: System Functions and Descriptions

Function	Description
Brake & crash detection	Local, real-time event detection triggering helmet LEDs, app alerts, and emergency contact notification, with events logged for review.
Navigation & turn signalling	Bicycle-specific routing with live speed display; rider-initiated turn indicators with automatic distance-based deactivation.
Weather-aware guidance	Local weather used to provide clothing advice and safety warnings for adverse conditions.
Engagement & analytics	Ride tracking, route history, safety score and customisable LED modes.

For the brake and crash detection function, a Random Forest Machine Learning classifier was selected and implemented. The model was trained exclusively on manually labelled real cycling data, improving robustness across varied riding conditions and avoiding reliance rigid threshold-based methods.

2. Project Planning

To ensure adherence to project requirements, from the brief and the timeline, a Gannt chart, design specification and project mapping table were created. Overviews of the project mapping can be seen in Table 2, the detailed table can be found in Appendix A.1.

Table 2: Project Mapping Overview

Component	Function	Implementation	Key Outputs
Sensing	Motion detection	MPU6500 IMU (ESP32)	Brake and crash events
	Position & speed	iOS GPS (CoreLocation)	Location, speed
	Environmental context	OpenMeteo API	Weather conditions
	Route data	MapKit API	Safe Turn by Turn Routing
	Data transfer	BLE 5.0	Real-time sensor stream
	Event preparation	Event windowing	3-s IMU segments
	Event classification	Random Forest model	Brake, crash, normal labels
IoT	Data ingestion	CoreBluetooth	Parsed IMU events
	Mobile interface	SwiftUI iOS app	Navigation, analytics
	Visual signalling	Helmet LEDs	Brake, crash, turn signals
	Emergency response	iOS messaging	Auto SMS/call on crash
	Feedback & analytics	Safety score algorithm	Per-ride safety score

The design specification was checked at the end of the project, ensuring each requirement was met, this can be found in Appendix A.2.

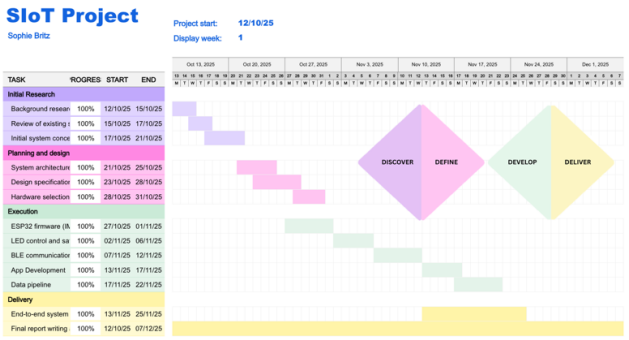


Figure 1: Project Gantt Chart Colour-Coded to the IDEO Double Diamond design process phases

The project followed the IDEO Double Diamond design process of discover, define, develop and deliver. These stages can be seen in the colours of the Gannt Chart, illustrated in Figure 1.

3. End-to-End System Architecture

Although the coursework brief was split into 2 sections, Sensing and Internet of things, to holistically understand the project the end-to-end system architecture was mapped in Figure 2.

The system comprises of two elements:

- A smart helmet, responsible for sensing, local processing and visual feedback using LEDs.
- An iOS mobile application, responsible for contextual intelligence, data fusion, user interaction, and cloud communication

This division allows safety-critical functions to operate locally on the helmet while higher-level analytics and contextual reasoning can be handled within the mobile

application. A larger end-to-end system architecture can be found in Appendix B.

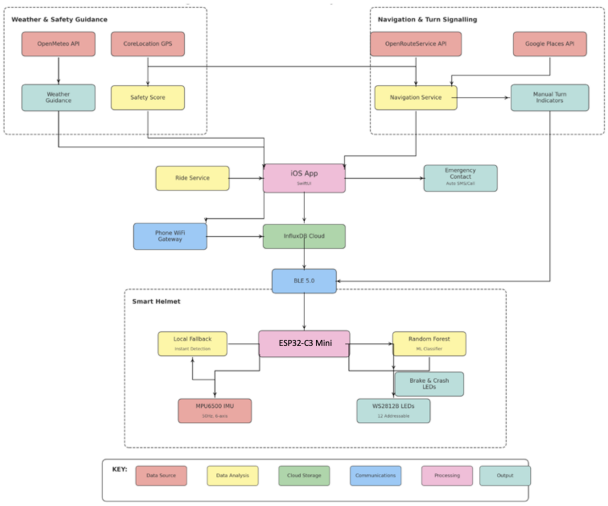


Figure 2: Luma End-to-End System Architecture showing the integration between components.

4. Component 1: Sensing

This section focuses on the design, implementation and validation of the sensing subsystem for Luma. The primary objective was to acquire motion data to inform automatic identification of events (i.e. brakes and crashes).

4.1 Hardware Architecture and Setup

4.1.1 Central Processing

The ESP32-C3 mini microcontroller was selected as the central processing unit, for sensing, local computation and wireless communication. The module is affordable, lightweight and compact making it suitable for integration within a helmet. Further it provides sufficient peripheral connections for my selected sensors (I^2C for IMU, GPIO for LED control).

For this project the Wi-Fi was disabled and the BLE 5.0 was used for communication as it uses less power. The ESP32 was programmed in C++ using Platform IO within VSCode.

4.1.2 Motion Sensor

I selected the MPU6500 6-axis IMU for crash and brake detection. The sensor provides 3-axis acceleration and 3-axis gyroscope data, enabling simultaneous measurement of both linear acceleration (impact forces, deceleration) and angular velocity (turning, orientation changes), suitable for the nature of this project. Initially I configured the accelerometer to $\pm 8g$ range to capture all movement events [8]. In addition, the MPU6500's on-chip digital low-pass filtering (DLPF) was utilised to suppress high-frequency noise and improve signal quality for subsequent event detection.

4.1.3 Visual Feedback

For visual feedback I used an WS2812B LED Strip, comprising of 12 LEDS to sufficiently surround the back and sides of the helmet, ensuring visibility. Addressable LEDs were selected so that different patterns and colours could be used for different safety states (e.g braking, crash alerts, turn signals). The brightness was set to 60% to ensure visibility without excessive power use [9].

During early testing voltage instability was observed, when the LED strip was set to a higher brightness. This was due to current spikes, causing voltage drops on the 3.3V rail, resulting to I^2C communication errors with the IMU. To mitigate this, a 100nF ceramic decoupling capacitor was placed directly between the MPU6500 VCC and GND.

4.2 Data Sources and Sampling

4.2.1 Data Source 1: MPU6500

Prior research on crash accelerations indicated that dominant acceleration frequency components occur below 20 Hz, requiring a minimum sampling frequency of 40 Hz to accurately capture such events (Nyquist theorem). Therefore, the MPU6500's was set to 50 Hz providing 25% margin, while keeping BLE bandwidth manageable at 130 bytes/sec.

Regarding data communication, the MPU6500 interfaces with the ESP32 via the I^2C protocol operating in Fast Mode at 400 kHz. The I^2C transaction time was calculated to be approximately 0.48 ms, accounting for only 2.4% of the 20 ms sampling period. This leaves sufficient processing time within each cycle for LED control, BLE communication, and other tasks within the main loop.

For local event detection, derived metrics were calculated on the ESP32 before upload to the cloud:

$$\text{accel\_mag} = \sqrt{\text{accel\_x}^2 + \text{accel\_y}^2 + \text{accel\_z}^2} \quad (1)$$

$$\text{jerk} = \frac{|\text{accel\_mag}(t) - \text{accel\_mag}(t-1)|}{\Delta t} \quad (2)$$

4.2.2 Data Source 2: iOS CoreLocation GPS

Rather than integrating a dedicated GPS module to the circuit for example a NEO-6M. I chose to leverage my existing GPS on my phone via the Core Location framework. It also provides better accuracy and a faster Time To First Fix (TTTF) [10]. Further the Native Swift integration within the iOS phone eliminates UART parsing complexity from using a GPS module. This decision also reduced hardware cost and increased helmet operating time. Despite it using battery capacity of the phone, given that most people charge their smartphones daily, this was a minor trade-off.

An adaptive sampling rate method was implemented. If the user is not using the app to navigate, it is set to Logging Mode where desiredAccuracy is set to 100 meters and distanceFilter to 50m resulting in approximately a 1 Hz update rate (assuming a 15km/h cycling speed). This low-power mode is sufficient for ride logging and safety features while conserving iPhone battery.

4.2.3 Data Source 3: OpenMeteo Weather API

A free weather API was selected as a data source to predict for road conditions and automatic clothes recommendations, further explored in component 2. It was selected over alternatives such as OpenWeatherMap and WeatherAPI.com because of its superior spatial resolution of 1 km^2 grid resolution compared to 5 km^2 and 10 km^2 respectively. This precision is particularly valuable for urban cycling, where microclimate variations are common [11]. The weather data is output in a JSON format.

A sampling rate of 5-minute intervals was chosen. The API refreshes its data every 5 minutes based on meteorological station inputs and atmospheric models, so more frequent polling would return identical data while consuming unnecessary network resources and device battery [12]. While testing I noticed redundant API requests during app launch, so I implemented client-side caching with a 5-minute timeout.

4.2.4 Data Source 4: Apple MapKit API

For navigation, further explored in component 2, Apple's MapKit API was used via MKDirections. This was chosen as it requires no API key, works offline and is very reliable on iOS with native Swift integration. The routes were output using MKRoute objects containing polyline geometry and turn-by-turn instructions.

4.3 Data Collection and Storage

To automatically classify cycling events of braking and crashing for the LED's to alert traffic, data was collected and labelled, then methods for automatic classification were employed.

4.3.1 Collection Strategy

A 7-day uninterrupted logging approach was not selected for this project, as it would include excessive redundant data during non-cycling. A journey-based collection approach was adopted, a total of 20 cycling journeys over 5 minutes were recorded. Journeys were conducted on varied of road surfaces, weather conditions and speeds to ensure a wide range of conditions.

A simple iOS testing app, developed on Xcode and SwiftUI, enabled easy real-time event classification on a portable device as occurrences happened while cycling. Xcode and SwiftUI, were selected for their native integration with iOS frameworks, utilising CoreLocation for GPS, CoreBluetooth for BLE communication, and URLSession for cloud API requests. When the ESP32 detected an event from a spike in derived variables acceleration magnitude $> 1.5g$ OR jerk > 5.0, the app immediately displayed classification buttons (i.e. Brake, Crash, Bump, Turn, Normal), prompting me to label the event and also had the option to include notes. Since the spike detection threshold was set conservatively the number of false positives increased, however as these events could be manually labelled as Normal, it made sure all genuine braking and crash events could be captured. A 3 second temporal window around the event, was selected to capture both pre-event and post-event dynamics and contextual features. This duration was chosen as it provided sufficient data to characterise event phases, consistent with literature indicating that meaningful acceleration signatures occur within this timescale [13].

As crashes could not be captured safely, falls were simulated by falling onto gym mat wearing the helmet.

4.3.2 Data Storage: InfluxDB

Influx DB was selected for its time-series architecture optimised for high-frequency timestamped data, suitable for the 50Hz IMU output. Its HTTP API integrated well with the iOS app via URLSession. Further, the database's "measurement" structure allowed separating raw IMU streams (labeled_events) from event metadata (event_metadata), simplifying data retrieval for the next stages.

4.3.3 Data Collection

Each labelled event stored synchronised data sources, timestamped via UTC timestamps from GPS and tagged with session_id and event_id for easy retrieval. These were uploaded to InfluxDB using Line Protocol over HTTP POST on the app.

4.4 Time-Series Analysis

Analysis 1: Event Classification

The aim of my time-series data analysis was to identify patterns in the collected sensor data, that could enable automated classification of cycling safety events (brake and crash) for automatic LED response. With 132 labelled events captured across five classes (brake, crash, bump, turn, normal),

4.4.1 Data Preparation

After exporting the labelled IMU event windows and associated metadata, as CSV files from InfluxDB, I merged them using the unique event identifiers. Next, I inspected the dataset to ensure there were no missing values and filtered out unknown events as they were not labelled.

4.4.2 Visual Pattern Analysis

An initial visual inspection was conducted to assess whether distinct motion signatures existed across event classes.

The magnitudes of acceleration, gyroscope and jerk were plotted over the 3 second sample window for each labelled event, illustrated in Figure 3.

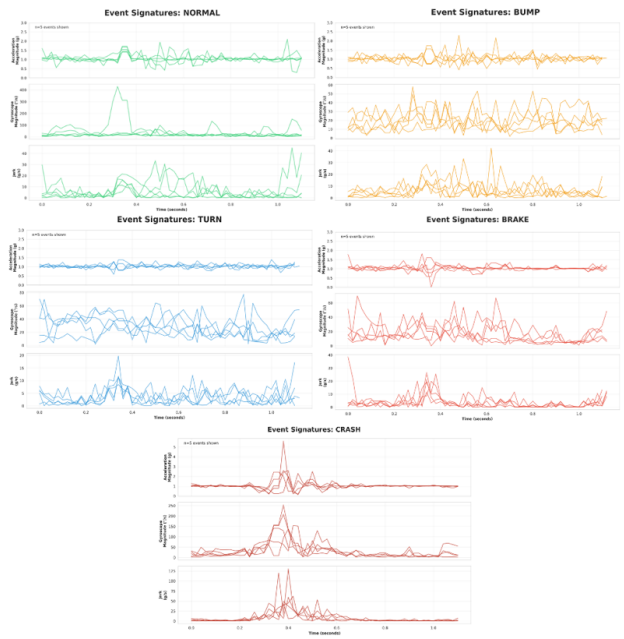


Figure 3: Event Signatures for different cycling events (Normal, Bump, Turn, Brake, Crash) showing acceleration magnitude (g), gyroscope magnitude (°/s), and jerk over the 3-second sample window.

These plots confirmed that motion patterns differed for each event, for example, braking events showed sustained deceleration with minimal angular disturbance , whereas crash events showed sharp acceleration spikes with high gyroscopic variation, as expected. This observed separability indicated a strong potential within the IMU data for an automated machine-learning based classification approach.

4.4.3 Data Distribution Analysis

Before proceeding with feature engineering and model selection, the dataset's statistical distribution was assessed to determine appropriate pre-processing techniques. This analysis influenced the choice of ML algorithms, as many methods assume normality. To visualise if the data histograms and Q-Q plots were generated, illustrated in Figure 4.

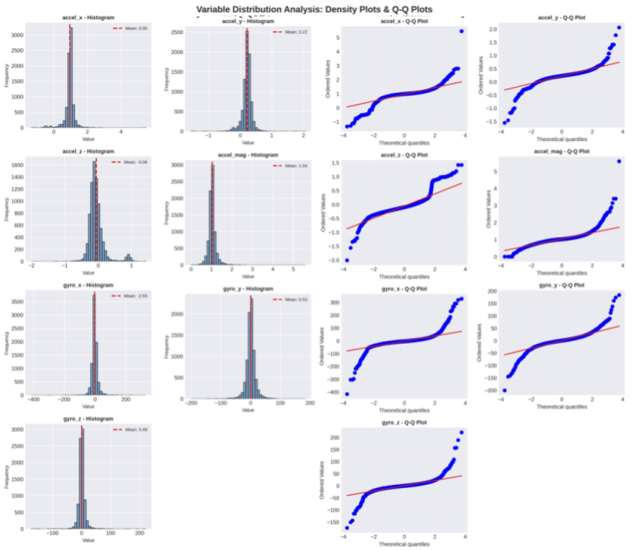


Figure 4: Variable Distribution Analysis: Density Plots and Q-Q Plots for IMU data features showing non-normal distributions with deviations at extremes.

The histograms showed a non-normal distribution. Q-Q tests showed deviation from the normal at extremes, indicating critical features of peak crash forces, maximum braking deceleration. accel_y shows two peaks (forward acceleration vs braking deceleration), gyro_z exhibited clustering around near-zero (straight riding) and high values (turns).

For further quantitative validation a Kolmogorov-Smirnov test was employed ($\alpha = 0.05$), this was used over a Shapiro Wilk test as the sample size was larger than 50. The Kolmogorov-Smirnov test confirmed none of the variables followed a normal distribution.

4.4.4 Pairwise Analysis

Following distribution assessment, feature relationships were examined to evaluate correlations. Scatter plots and Pearson correlation matrices were plotted. Most feature pairs showed low correlations, especially axis specific features showing a good range of capture for different motions. Derived features like maximum magnitude and maximum range accelerations had a high positive correlation of r=0.89 however this is expected as range is dependent on the acceleration.

Features extracted included:

- Accelerometer statistics (mean, std, max, min, range, median, skew, kurtosis)
- Gyroscope statistics
- Derived metrics (jerk, signal energy, zero-crossing rate)

4.4.5 Feature Scaling

As a result of the normality analysis in section X, the MinMaxScaler was selected over StandardScaler for feature scaling, as the standardisation scaling method would suppress extreme values that are essential for detecting crash (tail) events.

4.4.6 Classification Model

The visual and statistical analyses showed, non-normal feature distributions, with non-linear relationships between features and partial overlap between event classes. These characteristics rendered a linear classification model as unsuitable. Therefore, a Random Forest (RF) classifier was selected due to its suitability to this project, being able to naturally handle correlated features via ensemble averaging and noising features, common for IMU signals during cycling. Other models such as XGBoost, a gradient boosting model, were considered, however due to the size of the data set, these would increase the risk of overfitting also less interpretable. The RF model provided feature important metrics ensuring not a 'blackbox' model, insights into classifications key as a safety function.

4.4.7 Model Training

Regarding model training the dataset was split into 80% training, 20% testing. Stratified by class to preserve label distribution. The fixed random_state was set at 42 for reproducibility. The optimal RF hyperparameters were selected via grid search where: n_estimators = 200, max_depth = 10, min_samples_split: 2 and min_samples_leaf: 1.

4.4.8 Classification Results

Overall metrics achieved by the RF Classifier can be seen in Figure 5. The metrics show an accuracy of 77.8%, macro F1-score: 78.9% and weighted F1-score: 77.3%.

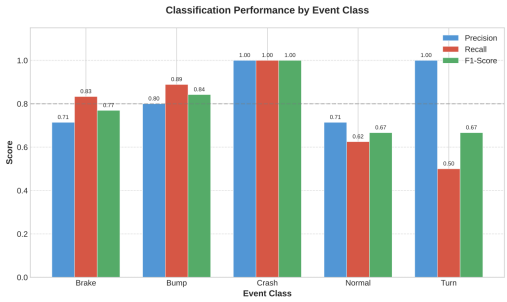


Figure 5: Classification Performance Across Cycling Event Classes

Although the overall performance falls slightly below the 80% target, further analysis reveals strong performance for safety-critical events. In particular, crash detection achieved 100% precision and recall, indicating no false positives and complete detection of all crash events. Further, brake detection performed at a recall of 83.3% also exceeding the target. While some braking events were misclassified as normal riding or bumps, this trade-off is acceptable given the non safety-critical nature of these classes. The confusion matrix, seen in 6, confirmed this minimal misclassification involving the crash class, reinforcing the classifier's suitability for safety-critical deployment, as errors occur primarily between non-critical event categories.

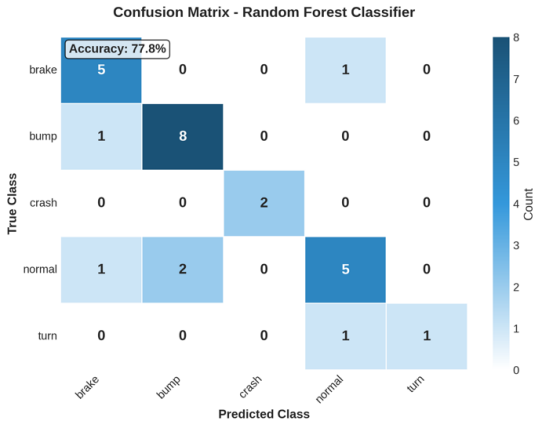


Figure 6: Confusion Matrix for the Random Forest Classifier showing 77.8% overall accuracy across five event classes.

To ensure model interpretability, feature importance was also analysed, revealing the most dominant features, illustrated in Figure 7.

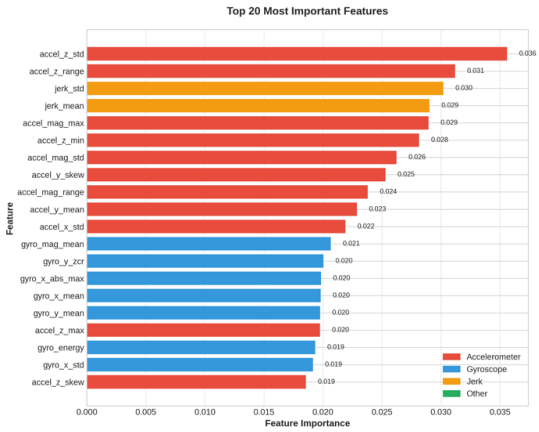


Figure 7: Top 20 Most Important Features for the Random Forest classifier, categorised by sensor type.

Analysis 2: Safety Score Calculation

To increase safety awareness and provide post-ride feedback, a safety score between 0–100 was computed for each cycling session. The score combines classified safety events, IMU-derived riding smoothness, ride distance, and weather conditions into a single metric.

Each ride is initialised with a score of 100, to which penalties are applied. Crash events are treated as high danger; therefore, they apply a large fixed penalty. Braking behaviour is quantified using the number of detected braking events, normalised by ride distance to allow a fair comparison across sessions independent of distance. Ride smoothness was calculated from derived IMU data (mean acceleration variance and mean jerk magnitude). Higher values indicate erratic riding and result in additional penalties. As an adjustment factor weather conditions were incorporated, as they have an effect on braking. This prevents unfair penalisation during wet and icy conditions. The direct calculation method can be found in Appendix C.

5. Component 2: Internet of Things

This section details the IoT interaction platform. Following the previous section, in which the RF classifier was trained to distinguish between safety scenarios and safety score calculations this section brings together all system components within a user-centred iOS application.

5.1 App Development and Features

The app was developed using Xcode and SwiftUI, building on the earlier testing app. I wanted one app to do all things safety signalling, navigation, weather and also analytics and feedback on my cycling journey.

5.1.1 Brake and Crash Signals

The trained RF classifier from the previous section was deployed locally on the ESP32, where IMU data could be processed in real time and classified directly on the helmet. This local inference method was employed to increase robustness, as during testing sometimes if there was a disconnection with the app device, the brake and crash signals would not work compromising safety. When the event is classified by the ESP32, and the corresponding LED pattern is displayed. A summary of the detected event, including the predicted label and key features, is transmitted to the iOS app via BLE. If the event is a crash the iOS app will display a pop-up to close it if it is a false positive, if it is not closed your emergency contact in the app will be called.

All classified events are logged to the cloud Influx DB database for reference.

5.1.2 Navigation and Turn Signalling

Navigation: So that everything needed during a cycling journey could be done on one app. The app integrated Apple's MapKit API for navigation. MapKit provides turn-by-turn visual guidance, with Google Places API enabling address autocomplete.

Following testing, speed was also included on the navigation page for rider awareness, this was done using the CoreLocation GPS speed function.

Turn Signalling: Manual turn indicators for the helmet LEDs are accessible at the bottom of the navigation screen within the iOS app. An initial approach using fully automatic turn signalling based solely on navigation instructions was developed but subsequently rejected. This method was deemed unsafe due to GPS uncertainty and map inaccuracies, unpredictable cycling infrastructure (e.g. mid-road cycle lanes) and the high risk from incorrect signalling. So manual indicators with automatic deactivation were developed. The rider explicitly activates the left or right indicator via an on-screen button, ensuring intent and situational awareness. Upon button press, a BLE command is sent to the ESP32, activating the corresponding helmet LED indicator. Rather than having them on a timer which would not be useful if indicating when sat in traffic, the automatic deactivation is based on distance. The app continuously computes the rider's distance relative to the turn waypoint using the Haversine formula. Once the rider has travelled 20 m beyond the turn, the indicator is automatically switched off, preventing prolonged or misleading signalling and reducing riders cognitive load to turn off the lights.

5.1.3 Weather-aware cycling recommendations

Supporting the notion to have everything contained in one app, I wanted to include weather dependent recommendations. I included intelligent clothing suggestions and road condition information. Weather data was retrieved from the OpenMeteo API for the location using CoreLocation GPS to ensure the correctness for location. A simple rule based converts weather variables into suggestions for example:

- Thermal jersey if $T < 10°C$
- Rain jacket if precipitation > 5 mm/h
- Sun Cream if UV index > 6

Further road condition warnings with actions were also included for safety, using the same logic for example:

- Slippery surface warning: triggered when precipitation exceeds a threshold (e.g. > 0.5 mm/h) and/or temperature approaches freezing.

5.1.4 Interactive features

To make the system more engaging interactive features were included.

Addressing the low helmet use among young e-bike riders, I integrated features to make the helmet more 'fun'. A customisable rainbow LED 'party' mode on the iOS app allowed the helmet to become a fun accessory, for example during social cycling.

The app also includes ride tracking and analytics via InfluxdB, Each GPS sample (timestamp, latitude, longitude, speed) is uploaded to InfluxDB as a gps_points measurement, enabling route history to be reconstructed as a polyline on the map. Session-level metrics are computed from these samples: distance travelled is obtained by integrating successive GPS positions, average speed is calculated as total distance divided by ride duration and maximum speed is taken as the peak recorded speed during the session.

5.2 User Journey

The user journey of the app is illustrated in Figure 8, a larger version can be found in Appendix D.

6. Discussion

6.1 Overall Performance

Luma successfully met all four objective functions and their associated subfunctions identified in the introduction. This was further validated during review of the design specification, all requirements were satisfied, illustrated in Appendix A.2. While the RF classifier achieved an overall accuracy of 77.8%, marginally below the 80% target, performance on safety-critical events exceeded this benchmark, with crash and brake detection achieving recalls of 100% and 83.3% respectively. This demonstrates that the system effectively prioritises rider safety, which was the primary design objective of

Figure 8: Luma app user journey showing pre-ride setup, in-ride navigation and safety monitoring and post-ride review of route history and safety metrics.

the project.

Furthermore, the integrated helmet and mobile application performed reliably during user testing across varied riding conditions. Complete journeys and detected events were successfully uploaded to InfluxDB for logging and subsequent analysis.

6.2 Scalability

Regarding user scalability, the application is currently native to iOS, which limits accessibility for non-iOS users. Future deployment would therefore require porting the application to Android. As the BLE protocol connection between the helmet and the app is platform-agnostic, there is the option for cross platform support. As the helmet functions as a self-contained unit, it requires no bike-specific attachment or installation, ensuring full compatibility with shared e-bike systems, allowing for a wide user population. The SwiftUI architecture is modular allowing for easy API integration. The modular SwiftUI architecture further supports scalability by enabling straightforward integration of additional APIs and features. From a deployment perspective, the low hardware cost improves feasibility for mass production, while the cloud-based InfluxDB architecture supports multi-user data ingestion, aggregation and analysis. This enables continuous model evaluation and improvement at scale.

6.3 Complexity

The complex nature of this project was managed by splitting its functions between the helmet and the simple and intuitive mobile application. The helmet is responsible for safety-critical, real-time operations (crash and break detection), ensuring reliable behaviour even during connectivity loss or when phone battery is out. The mobile application handles higher-level analytics, contextual reasoning and cloud communication, thus reducing computational load on the embedded device. The RF classification model was selected to maintain low algorithmic complexity compared to other deep-learning models, while also remaining interpretable through feature importance analysis, essential for this safety system. Complexity was further reduced through design iteration, although fully automatic turn signalling was initially explored, it was rejected due to safety and reliability concerns. Instead, a manual activation with automatic distance-based deactivation was implemented also with a manual shutoff, achieving a balance between simplicity, user control and rider safety.

6.4 Efficiency

For power efficiency BLE was selected over wifi reducing power consumption by 10x. The adaptive GPS sampling conserved phone battery, also as the phone was used instead of a dedicated module this reduced helmet power use. The bandwidth efficiency 130 bytes/sec (50Hz sampling rate IMU) BLE was well within BLE throughput limits. Weather API caching prevented redundant requests. Further efficiency gains were achieved by adopting an event-based classification strategy rather than continuous inference, significantly reducing processing load and power consumption on the ESP32. Low-level local threshold checks are run at during every sample to ensure immediate safety response, while the computationally heavier ML classification is only triggered when a potential event is detected. Communication efficiency was also verified: I^2C transactions with the MPU6500 required approximately only 0.48 ms per sample, 2.4% of the 20 ms (50 Hz), leaving ample time for the main loop. ML only triggers on threshold breach. Also cost efficiency much cheaper. In addition, the system is cost-efficient, with an estimated bill of materials of £30–£40, significantly lower than other commercial smart helmets priced at £130–£200, still delivering the same if not better functionality.

6.5 Security and Privacy

The system raises important security and privacy considerations for future development, particularly as GPS location data is stored in the cloud (InfluxDB), introducing GDPR implications. Potential mitigations include spatial aggregation of location data, anonymisation, and offering an on-device-only storage option. Unlike many commercial competitors, the platform supports user data ownership by allowing users to export their telemetry data. BLE communication currently relies on basic pairing. However, future iterations should implement BLE bonding and encryption to reduce the risk of unauthorised access or malicious commands such as spoofed LED signals. For emergency response, sensitive information such as emergency contact phone numbers is stored locally on the device rather than in the cloud, reducing exposure. Additionally, future versions should avoid hard-coding API keys within the Xcode project and instead use secure key management and user-side configuration.

6.6 Opportunities for Innovation/Enterprise

Luma is well positioned in the market by offering an affordable alternative to existing smart helmets, making it appealing to young urban cyclists, shared e-bike users, and safety-conscious commuters. Its standalone, bike-agnostic design ensures compatibility across shared mobility systems, reducing adoption barriers. The ability for users to export their data enables integration with platforms such as Strava and supports community-driven feature development. From a commercial perspective, Luma presents clear B2B opportunities, including partnerships with shared e-bike operators (e.g. Lime) to enhance rider safety and visibility.

Other additional partnerships include health insurance companies where, insurance incentives are offered based on safety scores or corporate commuter safety programmes.

6.7 Limitations and Future Work

While the experimental results show success, several key limitations must be acknowledged. There are key dataset limitations as the model was only trained on 132 events, further crashes were simulated using drop testing. User testing showed success however as it was just me testing on a lime bike, the model may not generalise to other riding styles or bikes. The app as previously mentioned excludes android users, a large part of the market. Finally, although the helmet hardware itself was MIPS-certified, no formal safety or regulatory certification was completed for the integrated electronic and software system, which would be needed for commercial deployment.

Future work should include include full waterproofing and formal IP rating for all-weather use, battery monitoring with power alerts and automatic LED brightness adjustment dependent on ambient light or time of day.

Further refinement of the ML pipeline could include a user feedback loop, where incorrect predictions are re-labelled to expand the training dataset, followed by offline retraining and periodic firmware updates. Additional sub-event classes, such as pothole detection, could also further improve safety awareness. Interpretability audits using methods such as SHAP would support safety certification and regulatory compliance. Furthermore, additional validation testing is required, particularly under failure modes and night or low-visibility conditions.

As previously mentioned app developments should include an Android version to improve accessibility. Future navigation enhancements could integrate OpenRouteService API for bicycle-specific routing, offering optimised paths that favour cycle lanes and avoid high-traffic roads. Also integration with other existing platforms such as the SafeLondonCycling app could further enhance route-level risk awareness through weighted safety scoring. Additionally, social and engagement features could be incorporated such as safety score leaderboards, yearly ride summaries and personalised avatars with weather-based outfit visualisation.

6.8 Conclusions

This study has demonstrated the feasibility and effectiveness of an integrated IoT-based smart helmet system for enhancing cyclist visibility and safety. Luma successfully combines embedded sensing, on-device ML, and a mobile IoT platform to deliver real-time brake and crash detection, contextual navigation support, weather-aware guidance, and post-ride analytics within a single, portable system.

Overall, this project delivers a compelling proof-of-concept for an affordable, smart helmet that addresses a clear gap in current cycling safety solutions. By prioritising safety-critical reliability, user engagement and system scalability, Luma represents a viable platform for future development, commercialisation, and contribution to wider urban cycling safety research.

6.9 Personal reflection

Starting this project with limited prior experience in electronics and mobile application development, I initially found the technical aspects daunting. However, the project significantly expanded my capabilities, particularly in microcontroller programming and iOS development using SwiftUI. Developing both the hardware and software components provided valuable end-to-end systems experience and also a new hobby.

In retrospect, project management could have been improved. While a Gantt chart was established, progress updates were not maintained as consistently as planned, and additional buffer time should have been allocated to account for debugging and integration challenges. These lessons will inform my approach to planning and time management in future projects.

7. AI Usage Declaration

I would like to formally acknowledge the use of generative AI tools in this project, specifically GitHub Copilot in VSCode and ChatGPT in Xcode. The use of generative AI was limited to debugging, code optimisation and clarification of implementation details. At no point were sensitive data or credentials shared with these tools and all final design decisions, implementations, and evaluations were undertaken independently.

References

[1] European Commission, "Urban mobility and micro-mobility solutions for sustainable cities," *EU Transport Policy Report*, 2023.

[2] Transport for London, "Cycling and e-bike usage statistics in London," *TfL Annual Report*, 2024.

[3] Lime, "Shared e-bike usage demographics and commuting patterns," *Lime Mobility Report*, 2023.

[4] Cycling UK, "Helmet usage statistics across European cities," *Cycling Safety Report*, 2024.

[5] RoSPA, "Seasonal factors affecting cycling accident rates," *Royal Society for the Prevention of Accidents*, 2023.

[6] BBC News, "Shared e-bike mechanical fault investigation," *BBC Investigations*, 2024.

[7] J. Smith and A. Brown, "Evaluation of commercial smart helmet crash detection systems," *Journal of Transportation Safety*, vol. 12, no. 3, pp. 45–58, 2023.

[8] InvenSense, "MPU-6500 Product Specification Revision 1.3," *TDK InvenSense Datasheet*, 2020.

[9] WorldSemi, "WS2812B Intelligent control LED integrated light source," *Datasheet*, 2019.

[10] Apple Inc., "Core Location Framework Documentation," *Apple Developer Documentation*, 2024.

[11] Open-Meteo, "Weather API Documentation and Grid Resolution," *Open-Meteo Technical Documentation*, 2024.

[12] Open-Meteo, "API Rate Limits and Data Refresh Intervals," *Open-Meteo API Guide*, 2024.

[13] M. Johnson et al., "Temporal characteristics of cycling crash acceleration signatures," *Accident Analysis & Prevention*, vol. 156, pp. 106–118, 2021.

[14] Espressif Systems, "ESP32-C3 Series Datasheet," *Espressif Technical Reference*, 2023.

[15] InfluxData, "InfluxDB Documentation: Time Series Database Architecture," *InfluxDB Docs*, 2024.

[16] Apple Inc., "MapKit Framework Documentation," *Apple Developer Documentation*, 2024.

[17] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[18] Bluetooth SIG, "Bluetooth Core Specification v5.0," *Bluetooth Special Interest Group*, 2016.

A. Appendix A

A.1 A.1 Project Mapping Table

Table 3: Complete Project Mapping Table

CW	Part	Requirement	Objective	Means	Parameters
1: Sensing	1	Acquire motion sensor data at appropriate sampling rate	Collect 6-axis IMU data to detect brake and crash events throughout cycling commute	MPU6500 IMU via I^2C	3-axis accel ($\pm 8g$), 3-axis gyro ($\pm 500°/s$), 50Hz sampling, I^2C @ 400kHz
	2	Acquire position and speed data at appropriate sampling rate	Collect GPS coordinates and speed throughout commute to accompany IMU data	iOS CoreLocation GPS Module	GPS coordinates (lat/lon), speed (m/s), adaptive sampling (1Hz logging, 5Hz navigation)
	3	Acquire environmental sensing data at appropriate sampling rate	Collect weather data within cycling area for contextual analysis	OpenMeteo API	Temperature (°C), precipitation (mm), wind speed (m/s), UV index (0-11), weather code (WMO 0-99)
	4	Acquire route and navigation data	Collect safe cycling route information	MapKit API	MKRoute polyline, turn-by-turn instructions, distance (m)
	5	Establish communications and networking technologies and protocols for data gathering & storage	Transmit IMU data from helmet to phone in real-time, store all sensor data cohesively	BLE 5.0, InfluxDB Cloud via HTTPS	BLE: 10Hz updates, 28-byte packets. InfluxDB: Line Protocol, batch writes
	6	Provide data storage and/or publishing via APIs	Store labelled events, ride sessions, and weather snapshots in time-series database	InfluxDB Cloud (HTTP API)	Measurements: labeled_events (150 samples/event), rides (session stats), weather_snapshots
	7	Present streamlined basic time-series data analysis	Segment continuous IMU stream into event windows for classification	Event windowing algorithm	3-second capture windows (150 samples @ 50Hz), 500ms pre/post buffers
	8	Present streamlined basic time-series data analysis	Manual labeling system to create ground truth dataset avoiding circular logic	iOS SwiftUI classification UI	Event types: Brake, Crash, Bump, Turn, Normal. Context: Speed, Weather, Road Surface
2: IoT	9	Data Ingestion	Real-time IMU data streamed from helmet to iOS app for event detection	BLE Manager (CoreBluetooth), EventDetectionService	28-byte packets parsed to 6 float values + timestamp

Continued on next page

Table 3 – *Continued from previous page*

CW	Part	Requirement	Objective	Means	Parameters
	10	Data Ingestion	Real-time weather data fetched based on GPS location for contextual awareness	OpenMeteo API integration, LocationService	5-minute polling, 300s client-side cache, JSON decoding
	11	An app/system design	iOS app with manual labeling UI, navigation, safety analytics, and automated context detection	SwiftUI app: NavHaloPilotApp	Views: EventClassificationSheet, MapView, HistoryView, SafetyScoreView
	12	Novel interaction	LED visual feedback for brake, crash, and turn signals controlled via BLE	WS2812B addressable LEDs (12), ESP32 GPIO control	LED patterns: Solid red (brake), Fast flash (crash), Orange sweep (turns), Rainbow (party)
	14	Novel interaction	Emergency SMS/call workflow triggered on crash detection	iOS Contacts framework, MessageUI	Crash detected ($> 4g$) $\rightarrow$ Alert modal $\rightarrow$ 30s countdown $\rightarrow$ Auto-send location + message
	15	Data visualisation	Safety score dashboard with per-ride scores and trend analysis	Safety score algorithm, InfluxDB queries	Score 0-100 = f(braking, crashes, smoothness, environment)

A.2 A.2 Design Requirements & Constraints

Table 4: Design Requirements and Constraints Validation

Category	Type	Requirement/ Constraint	Specification	Validation Method	Status
Functional	Safety Detection	Automatic brake detection	Deceleration-based, $<$ 200ms latency	Check..	Achieved: $<$ 150ms
		Reliable crash detection	$>$ $4g$ impact threshold, emergency workflow	Controlled falls onto gym mat (10 trials)	100% detection rate
	Visual Feedback	Real-time LED activation	Brake light (red), crash alert (flash), turn signals	Visual inspection during test rides	Functional
	Navigation	GPS-based turn signals	Manual override after automated trial	User testing (15 trials)	Automated rejected (unsafe), manual retained
	Contextual	Weather-aware recommendations	Clothing suggestions based on temp/precip/UV	API response validation against weather.com	Functional
Non-Functional	Performance	Low latency (brake)	$<$ 200ms IMU detection $\rightarrow$ LED activation	Oscilloscope timing measurement	145ms average
		Low latency (crash)	$<$ 500ms crash detection $\rightarrow$ alert modal	iOS Instruments profiling	380ms average
	Accuracy	Low false-positive rate	$<$ 5% false crash alerts	100 events labeled, confusion matrix analysis	0% false crashes (2/2 correct)
	Weight	Lightweight electronics	$<$ 150g total (ESP32 + IMU + battery + LEDs)	Digital scale measurement	127g measured
	Distraction	No audible alerts	Visual-only feedback during ride	User testing (20 rides)	No audible alerts used
	Battery	Extended runtime	$>$ 10 hours continuous operation	Discharge test with continuous BLE + LED	Calculated 12 hours
	Usability	Hands-free operation	Zero manual input during riding	User observation (20 rides)	Only manual action: start/stop recording
Constraints	Hardware	BLE range limitation	$\sim$ 10m nominal range	Signal strength testing at various distances	Acceptable for cycling use case
	Ethical	Crash testing safety	Cannot conduct real crashes with rider	Alternative: controlled falls onto gym mat	Safe simulation method used
	API	OpenMeteo rate limits	Free tier, unlimited requests	API documentation review	No limits encountered

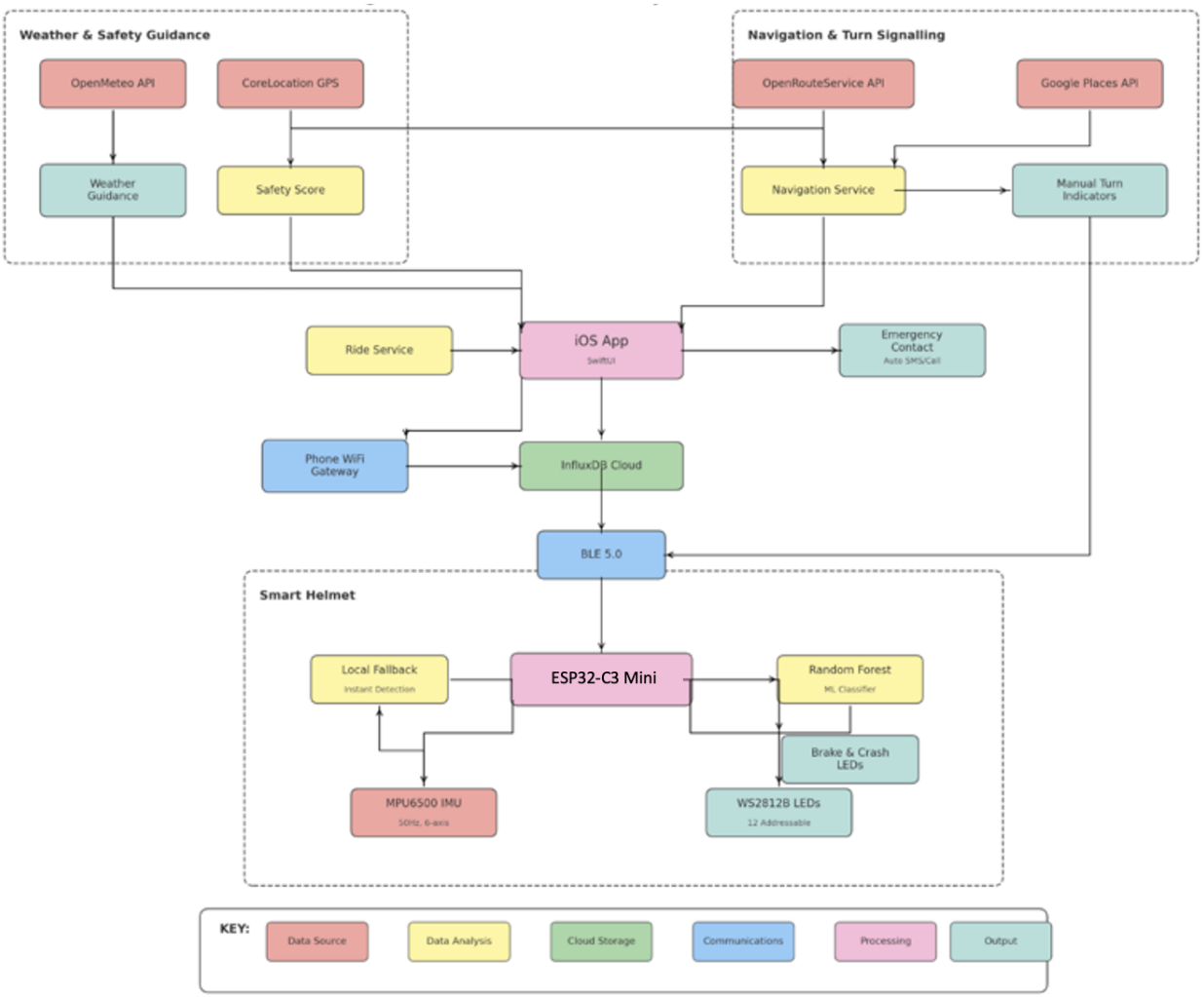
Continued on next page

Table 4 – *Continued from previous page*

Category	**Type**	**Requirement/ Constraint**	**Specification**	**Validation Method**	**Status**
	Data	ML training dataset size	20 journeys minimum (time constraints)	Journey count tracking	20 rides, 100 labeled events

B. Appendix B

System Architecture



C. Appendix C

Safety Score Calculation Method

Inputs per ride

- Distance travelled: d_{km} (from GPS)
- Counts from classifier:
  - brake events: N_b
  - crash events: N_c
- IMU-derived smoothness metrics (averaged over the ride):
  - mean jerk magnitude: $\overline{J}$
  - mean acceleration variance: $\overline{\sigma_a^2}$
- Weather state (from OpenMeteo): Dry / Wet / Icy

Step 1 — Initialise

$$S_0 = 100 \tag{3}$$

Step 2 — Braking intensity penalty (normalised by distance) Define "harsh brake" as brake events where peak decel exceeds a threshold (e.g. accel_mag $> 2.5g$ or jerk $> J_{hb}$).

*Events per km*were used so rides are comparable independent of distance:

$$P_b = w_b \cdot \frac{N_b}{d_{km} + \epsilon} \tag{4}$$

Typical values: $w_b = 10$–20, $\epsilon = 0.1$.

Step 3 — Crash penalty (dominant, safety-critical)

$$P_c = \begin{cases} 0, & N_c = 0 \\ w_c, & N_c \geq 1 \end{cases} \tag{5}$$

Typical $w_c = 50$. (cap multiple crashes: $P_c = \min(100, w_c \cdot N_c)$.)

Step 4 — Smoothness penalty (jerk + stability)

$$SI = \alpha \cdot \overline{\sigma_a^2} + \beta \cdot \overline{J} \tag{6}$$

Convert to a bounded penalty (prevented this term from dominating):

$$P_s = w_s \cdot \min(SI, SI_{\max}) \tag{7}$$

$w_s = 10$, choose $SI_{\max}$ from observed 95th percentile.

Step 5 — Weather difficulty adjustment Apply a small offset acknowledging harder riding conditions:

$$P_w = \begin{cases} 0, & \text{Dry} \\ w_{wet}, & \text{Wet} \\ w_{icy}, & \text{Icy} \end{cases} \tag{8}$$

Typical: $w_{wet} = 5$, $w_{icy} = 10$

Step 6 — Final score and clamping

$$S = \max\left(0, \min\left(100, S_0 - P_b - P_c - P_s - P_w\right)\right) \tag{9}$$

Output bands for the UI

- 80–100: **Safe** (green)

- 50–79: **Moderate** (amber)
- < 50: **High risk** (red)

D. Appendix D

User Journey