POLITECNICO DI TORINO

Master degree course in Digital Skills for Sustainable Societal Transitions

Master Degree Thesis

Design and Validation of a Containerized IoT Architecture for Wearable Sensor Ingestion, Activity Recognition, and Multi-Source Data Visualization

Relatori
Prof. Gianvito Urgese
Dr. Giuseppe Fanuli
Dr. Andrea Pignata

Candidato
Arash Omidi Hosseinabadi

July 2026

Abstract

The widespread adoption of wearable sensors in sport science and health monitoring has exposed the need for data-processing infrastructures that are simultaneously reliable, extensible, and reproducible. Such infrastructures must support a coordinated sequence of operations: receiving data streams from heterogeneous sources; pre-processing and time-synchronising those streams; filtering and cleaning noisy or incomplete readings; and executing processing algorithms to extract meaningful information. These requirements motivate a flexible and modular infrastructure, where each component can be adapted, replaced, or extended without redesigning the entire system.

This thesis presents the design, implementation, and validation of *Wearable IoT Activity Pipeline*, a containerised Internet-of-Things (IoT) platform for wearable sensor acquisition, real-time data streaming, AI-based activity classification, time-series storage, and heterogeneous sensor integration. The platform is composed of twelve independent containers communicating through well-defined, lightweight message-passing interfaces, collectively enabling: (a) data sampling from physical sensors and pre-recorded datasets; (b) time-synchronised ingestion and persistent storage; (c) pre-processing and cleaning through configurable validation filters; (d) execution of machine-learning algorithms on the acquired streams; and (e) end-to-end monitoring through accessible user interfaces. The platform is built on Docker, MQTT, InfluxDB, ONNX, and Grafana, making it reproducible, extensible, and comparable with real-world IoT architectures.

Three complementary data flows are implemented and validated. In the first, the system is evaluated against a publicly available inertial measurement dataset to verify the correctness and reproducibility of the full ingestion, cleaning, storage, and inference pipeline under controlled conditions. In the second, the platform is tested with a live wearable device in real time, confirming that all components perform correctly in a realistic acquisition scenario. In the third, electroencephalography and electrocardiography signals from an open neuroimaging dataset are introduced through dedicated simulator and cleaning services, demonstrating that the architecture accommodates sensor modalities differing fundamentally in structure, sampling rate, and schema from inertial data, without modifying existing components. Across all three scenarios, the evaluation confirms that: (a) communication between containers is coherent and reliable; (b) every component sustains the required data rates without loss; (c) AI-based classification meets the expected accuracy for the targeted activity set; and (d) the architecture adapts to new sensor types with minimal engineering effort.

The principal contribution is a reusable IoT architecture that combines real-time stream processing with reproducible offline evaluation while preserving clear

ownership boundaries between acquisition, cleaning, storage, inference, and visualisation. The thesis also evaluates two execution modes for the activity-recognition service: a live mode, where cleaned data is consumed from the MQTT stream and classified continuously; and an evaluation mode, where data is stored in InfluxDB and then reprocessed, enabling repeatable experiments under different configurations. Runtime reliability is assessed through observable metrics including queue utilisation, retry behaviour, batch-write performance, and data-loss monitoring.

Acknowledgments

I express my sincere gratitude to my supervisor, Prof. Gianvito Urgese, for their invaluable guidance, constructive feedback, and continuous support throughout the development of this thesis.I am also grateful to Giuseppe Fanuli and Andrea Pignata for technical discussions and suggestions that contributed to the design and validation of the system.

I would like to thank my family, friends, and colleagues for their encouragement and patience during the development of this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Executive Summary

Growing use of wearable sensors requires infrastructure that can reliably collect, process, and store continuous data streams. Although Human Activity Recognition (HAR) research often emphasizes classification algorithms [1], reliable acquisition, replay, and extensibility also depend on the supporting data platform.

This thesis presents *Wearable IoT Activity Pipeline*, a containerized, microservice-based IoT platform [2, 3]. It focuses on architecture and validation by integrating MQTT [4], InfluxDB [5], ONNX inference [6], and Grafana [7] in a reproducible wearable HAR environment.

1.2 Context and Motivation

Wearable devices, including IMUs [8], smartwatches, and physiological sensors, support sports analytics, health monitoring, and activity recognition. Their value depends on infrastructure that reliably transports, validates, processes, stores, and visualizes sensor streams.

Monolithic systems make these responsibilities difficult to maintain and extend; research-oriented IoT systems therefore need modular services, containerization, message-based communication, and reproducible workflows.

1.3 Problem Statement

Wearable IoT platforms must balance low-latency processing with reproducible evaluation while separating raw data, processed measurements, and predictions for debugging, validation, and future extensions.

The central research question is:

How can a modular and reproducible IoT architecture be designed and validated to support both real-time wearable sensing and offline experimental evaluation while maintaining clear separation between acquisition, processing, storage, inference, and visualization components?

1.4 Research Gap and Thesis Positioning

IoT architecture and wearable HAR are mature research areas, but they are often addressed separately: IoT studies emphasize general architectures, HAR studies emphasize model performance, and technical documentation focuses on individual tools. Less attention is given to reproducible research infrastructure that integrates wearable acquisition, messaging, cleaning, time-series storage, inference, and observability.

Pipeline failures can also arise from inconsistent schemas, hidden coupling, failed writes, message backlogs, or incorrect preprocessing. This thesis therefore positions the IoT pipeline as its primary contribution: an architecture supporting live sensing, dataset replay, separated data ownership, observable runtime metrics, and extensible sensor integration.

1.5 Research Objectives

This thesis pursues the following objectives:

1. Design a modular microservice architecture for wearable sensing and HAR.
2. Implement a containerized ingestion pipeline using MQTT and time-series storage.
3. Support both live sensor acquisition and reproducible dataset replay.
4. Integrate an ONNX-based HAR service while separating sensor and prediction storage.
5. Provide operational monitoring through dashboards and runtime metrics.
6. Demonstrate extensibility by integrating additional sensor families.

1.6 Contributions

The main contributions of this thesis are:

1. A reusable microservice architecture separating acquisition, validation, storage, inference, and visualization.

2. A unified platform supporting both real-time streaming and offline experimental evaluation.

3. Quantitative validation using operational metrics including throughput, queue utilization, retry behavior, and data-loss monitoring.

4. An extensible architecture that supports integrating new sensor families without redesigning the processing pipeline.

1.7 System Scope

The platform combines MQTT ingestion, time-series storage, wearable-data processing, HAR inference, and Grafana, and is validated with live MetaWear data [9] and replayed datasets.

The thesis focuses on system architecture and validation rather than machine-learning innovation. Camera-based tracking, computer vision, EEG/ECG-specific models, edge deployment, and production-grade authentication are left for future work.

1.8 Architecture Overview

Figure 1.1 shows how the platform progressively transforms sensor data into a cleaner, reusable representation.

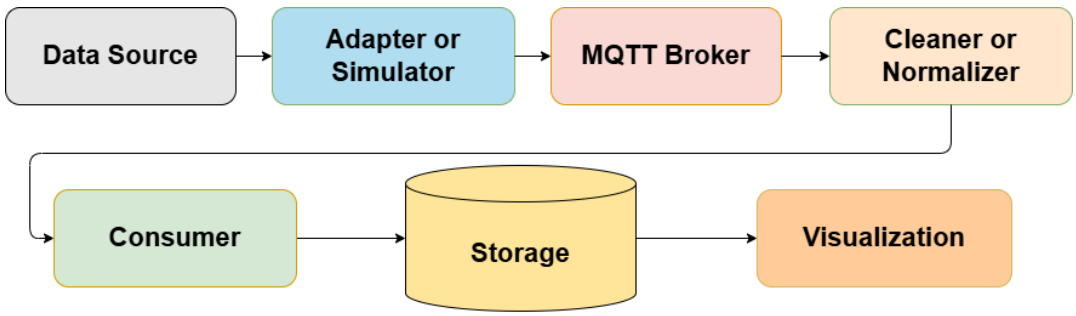

Figure 1.1: Processing pattern used by the Wearable IoT Activity Pipeline platform.

Figure 1.1 shows why the platform is not built as one continuous program. The adapter or simulator hides source-specific details, MQTT separates producers from consumers, the cleaner turns raw messages into stable records, and the final services can store, process, or visualize those records without reaching back into the acquisition code.

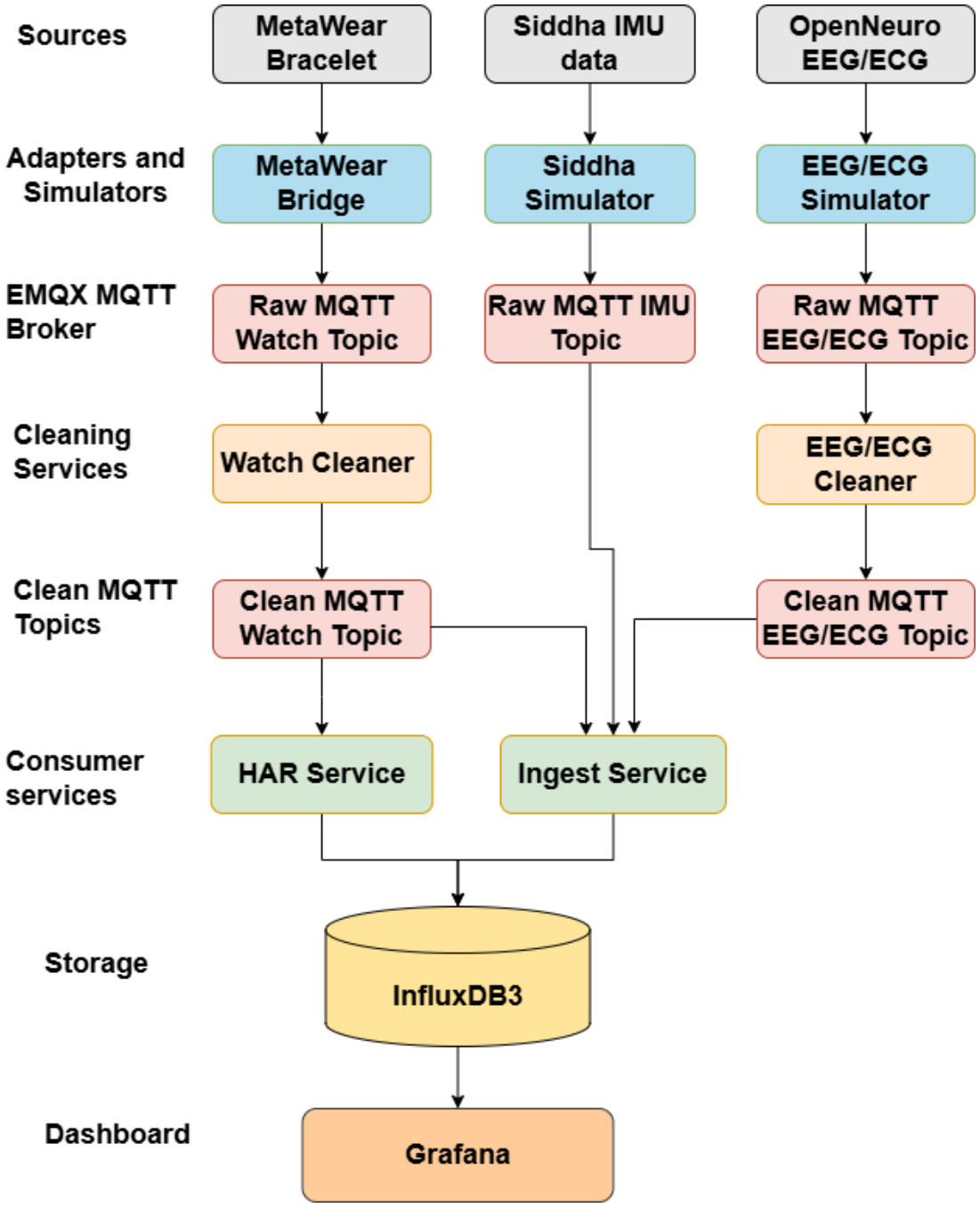

Figure 1.2: High-level architecture of the implemented Wearable IoT Activity Pipeline platform.

Figure 1.2 expands the general pattern into the concrete system implemented for the thesis. The upper path shows the live MetaWear workflow, the middle path shows reproducible SIDDHA replay, and the lower path shows the EEG/ECG

extension. All three paths share the same broker, ingest, storage, and dashboard foundation, which is the main reason new data sources can be added without redesigning the whole platform.

The remainder of this thesis is organized as follows.

Chapter 2 reviews the theoretical foundations of IoT architectures, MQTT messaging systems, wearable sensing, HAR, time-series databases, observability, and containerized deployment.

Chapter 3 presents the proposed architecture, system components, implementation choices, data flows, and validation methodology.

Chapter 4 reports the experimental results and evaluates system performance, reliability, and extensibility using both live and replay-based validation scenarios.

Finally, Chapter 5 summarizes the contributions of the work, discusses limitations, and outlines opportunities for future research and development.

Chapter 2

Literature Review

2.1 Chapter Overview

Chapter 1 introduced the motivation, problem statement, objectives, scope, and high-level architecture of *Wearable IoT Activity Pipeline*. This chapter reviews the scientific and technical background required to justify the system design.

The review focuses on IoT architectural models, microservice-based deployment, publish/subscribe communication, time-series data management, observability, wearable sensing, HAR, model portability, BLE gateway design, and heterogeneous sensor extension. Peer-reviewed literature and technical books are used for general architectural and methodological claims, while official documentation is cited when implementation-specific technologies are first discussed.

2.2 IoT Architectural Models

The Internet of Things is commonly described as a distributed computing paradigm in which physical devices, communication networks, data-processing services, and user-facing applications cooperate to provide higher-level functionality. Although different studies propose different numbers of layers, most IoT architectures distinguish between sensing, communication, processing, storage, and application responsibilities.

This layered perspective is important because IoT systems rarely consist of a single software component. A sensor produces measurements, a communication layer transports them, a processing layer validates or transforms them, a storage layer preserves them, and an application layer exposes them to users or other systems. Keeping these responsibilities separate improves maintainability, extensibility, and system interpretability.

For wearable sensing applications, separation of concerns is particularly important. Wearable devices operate under constraints such as sampling frequency,

battery capacity, wireless communication, and sensor placement. Processing and storage layers instead address concerns such as timestamp handling, message consistency, persistence, queryability, and later analysis. A layered architecture therefore prevents device-specific logic from becoming unnecessarily coupled to downstream analytics or visualization.

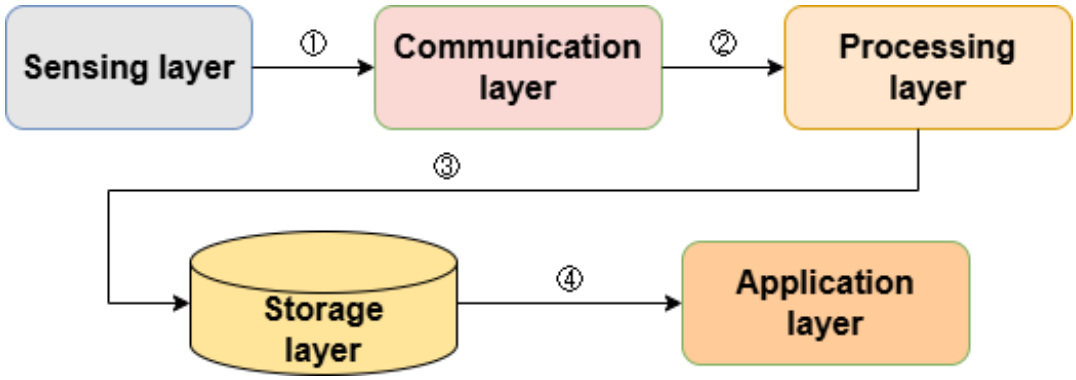

Figure 2.1: Generic layered IoT architecture separating sensing, communication, processing, storage, and application responsibilities.

Figure 2.1 should be read from left to right: measurements are first captured by sensors, then transported, processed, stored, and finally exposed through an application layer. The important point for this thesis is that each layer has a different responsibility, so a later change in storage, inference, or visualization should not require changes in the wearable acquisition logic.

Table 2.1 summarizes the responsibilities of the main IoT architectural layers and explains their relevance to wearable systems.

Table 2.1: General IoT architectural layers and their responsibilities.

Layer	**Typical responsibility**	**Relevance to wearable systems**
Sensing layer	Captures physical or physiological measurements.	Includes IMU, EEG, ECG, and other wearable sensors.
Communication layer	Transports data between producers and consumers.	Supports wireless acquisition and broker-based telemetry.
Processing layer	Cleans, validates, aggregates, or interprets data.	Converts noisy sensor streams into usable data products.
Storage layer	Persists historical measurements and derived results.	Enables replay, validation, and dashboard queries.
Application layer	Presents information or triggers higher-level actions.	Includes dashboards, analytics views, and monitoring interfaces.

2.3 Microservices and Containerized Deployment with Docker Compose

Microservice architecture decomposes a software system into small, independently deployable services. Each service owns a limited responsibility and communicates with other services through explicit interfaces [10]. This approach is especially relevant to IoT systems because sensing, communication, persistence, inference, and visualization often require different technologies and evolve at different speeds.

Compared with a monolithic architecture, a microservice-based system can improve modularity, maintainability, and fault isolation. For example, a wearable acquisition component can be modified without rewriting the storage layer, and an inference service can be updated without changing the message broker. This makes the approach suitable for research-scale experimentation, where live acquisition, dataset replay, and future sensor extensions must coexist within the same platform.

However, microservices also introduce operational complexity. Services require clear interfaces, explicit configuration, consistent deployment, and runtime monitoring. Without these controls, a distributed system can become difficult to debug and reproduce.

Containerization helps address this issue by packaging each service together with its runtime dependencies. This produces an isolated and repeatable unit that can be deployed without manually recreating the software environment on the host system [11].

Docker Compose extends this approach to applications that require multiple containers. A Compose configuration file declares the application services and can also define their images or build instructions, environment variables, exposed ports, networks, persistent volumes, and dependencies [12]. Each service definition is instantiated as a container, while shared networks allow the containers to communicate as one application. Persistent volumes can retain database or configuration data when containers are recreated.

The complete application can then be created and started through a single Compose command. Because the service topology and configuration are stored in a version-controlled file, the same set of containers can be stopped, rebuilt, and started again in a consistent way. In a thesis context, Docker Compose therefore supports reproducibility by making both the software dependencies and the deployment relationships explicit.

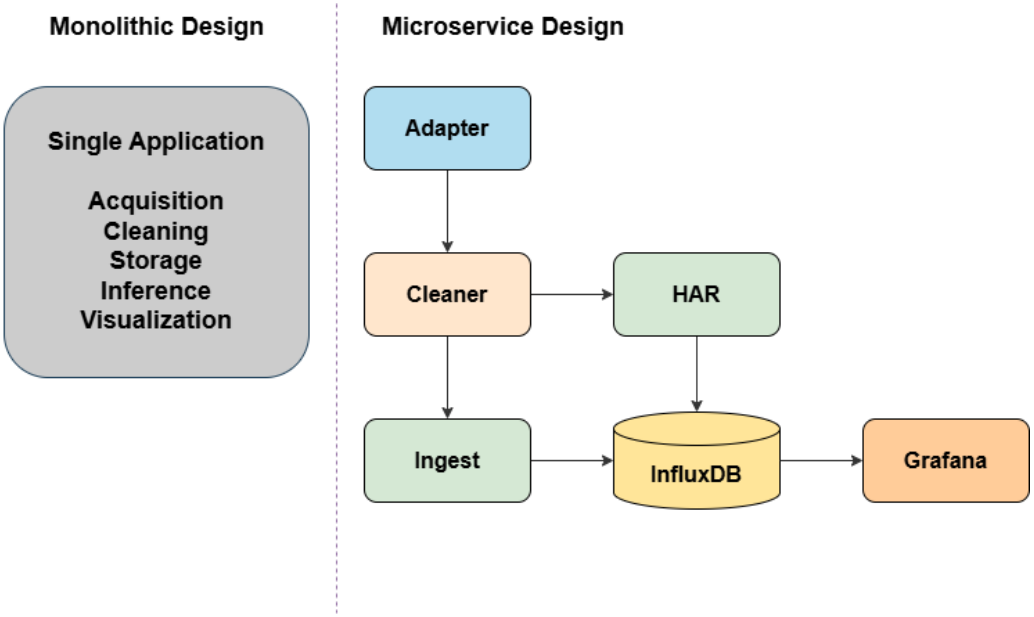

Figure 2.2: Conceptual comparison between a monolithic IoT application and a microservice-based IoT architecture.

Figure 2.2 contrasts two ways of organizing the same IoT responsibilities. In the monolithic case, acquisition, cleaning, storage, inference, and visualization are bundled together. In the microservice case, those responsibilities are split into smaller services connected by explicit interfaces, which makes individual parts easier to replace and validate.

Table 2.2 compares monolithic and microservice approaches across responsibility separation, maintainability, reproducibility, fault isolation, and operational complexity.

Table 2.2: Monolithic and microservice approaches in IoT experimentation.

Aspect	**Monolithic approach**	**Microservice approach**
Responsibility separation	Multiple functions are implemented together.	Each service owns a narrow function.
Maintainability	Changes may affect unrelated parts of the system.	Components can evolve independently.
Reproducibility	Runtime dependencies may be hidden.	Container definitions make deployment explicit.
Fault isolation	A failure can affect the whole application.	Failures can be localized to specific services.
Complexity	Simpler at small scale.	Requires configuration, monitoring, and clear interfaces.

2.3.1 FastAPI for Service APIs

An application programming interface (API) provides a defined way for one software component or user to request data or operations from another component. In a microservice system, HTTP APIs are commonly used for synchronous operations such as health checks, status inspection, configuration discovery, and database queries, while asynchronous message brokers handle continuous event streams.

FastAPI is a Python web framework for implementing such APIs. It uses standard Python type declarations to define request and response data, performs data validation, and generates an OpenAPI description with interactive API documentation [13]. These features are useful for operational interfaces because endpoint behavior and expected data structures remain explicit. In an IoT pipeline, FastAPI can therefore expose health and query endpoints without replacing MQTT as the transport mechanism for continuous sensor messages.

2.4 Publish/Subscribe Communication and MQTT Topics

Publish/subscribe communication is widely used in IoT systems because it decouples data producers from data consumers. Instead of sending messages directly to a specific receiver, producers publish messages to topics, and consumers subscribe to the topics relevant to their role. This model is well suited to sensor telemetry, where multiple services may need access to the same stream.

MQTT is a lightweight, topic-based publish/subscribe protocol standardized for telemetry and IoT scenarios. In wearable systems, MQTT can act as the communication backbone between acquisition components, cleaning services, ingestion services, and inference services [4].

In MQTT, a topic is a UTF-8 string that identifies the channel associated with a published message. Topic names can be organized into levels separated by forward slashes, such as `tennis/watch/raw`, so an application can represent a sensor family, source, or processing stage in the topic hierarchy. A subscriber registers a topic filter, which may select one exact topic or use the single-level `+` or multi-level `#` wildcard to match several related topics. The broker compares each published topic name with the active subscription filters and forwards the message to every matching subscriber [4]. The topic controls message routing, while the application-specific payload schema defines the content of the message.

The broker is central to this model. It receives published messages and forwards them to subscribed consumers. Broker-based communication reduces direct dependencies between components. A storage service and an inference service, for example, can consume the same cleaned data stream without the producer needing to know that either service exists.

EMQX is an MQTT broker platform designed for scalable publish/subscribe communication and IoT messaging scenarios [14]. In a containerized research pipeline, such a broker supports modularity by allowing services to communicate through topic contracts rather than direct function calls or tightly coupled database writes.

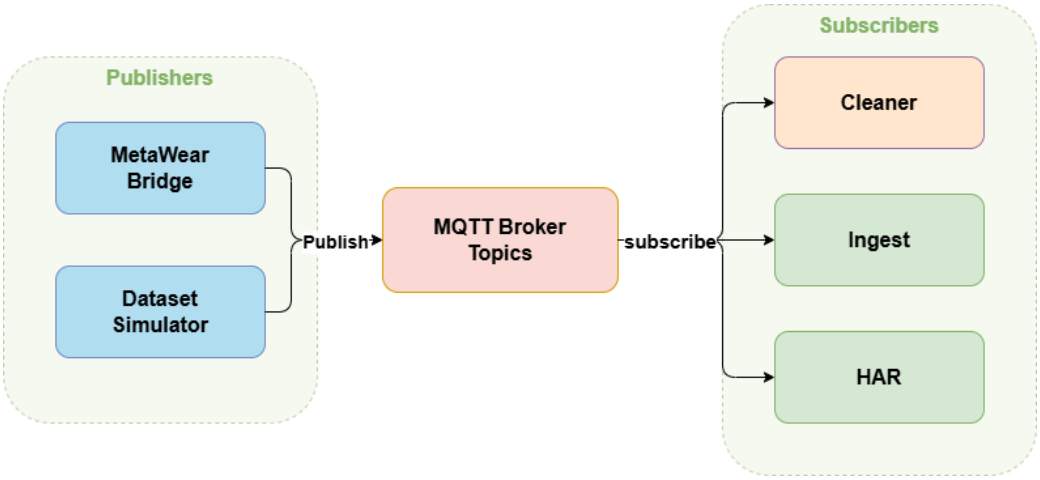

Figure 2.3: MQTT publish/subscribe communication model with publishers, broker topics, and independent subscribers.

Figure 2.3 shows the role of the broker as the only component that publishers and subscribers need to share. A data source publishes once to a topic, and different services can subscribe to that topic for cleaning, storage, inference, or monitoring. This is the communication pattern used throughout the implemented platform.

Table 2.3 summarizes the principal properties of the publish/subscribe model and their architectural benefits for IoT pipelines.

Table 2.3: Publish/subscribe properties relevant to IoT pipelines.

Property	**Explanation**	**Architectural benefit**
Producer-consumer decoupling	Producers publish without knowing the consumers.	Services can be added or removed with limited impact.
Topic-based routing	Messages are organized through named channels.	Data streams can be separated by sensor type or processing stage.
Asynchronous communication	Producers and consumers do not need to operate in lockstep.	Supports continuous telemetry and independent processing rates.
Multiple subscribers	Several services can consume the same stream.	Enables parallel storage, cleaning, inference, or monitoring.

2.5 Time-Series Data Management

Wearable and IoT systems generate continuous, timestamped data. These data are usually append-heavy: new measurements arrive over time, while historical records are queried for analysis, monitoring, replay, or validation. This makes time-series databases a natural fit for sensor-based systems.

Unlike general-purpose transactional databases, time-series databases are optimized around temporal queries, ordered measurements, time windows, and aggregation over intervals. These capabilities are important for sensor pipelines because sampling frequency, event ordering, and temporal windows are central to both activity recognition and experimental validation [15].

InfluxDB is a time-series database designed for telemetry and monitoring workloads. InfluxDB 3 organizes time-indexed records into tables whose rows contain a timestamp, tag columns for identifying metadata, and field columns for measured values [5]. Its SQL query support is suitable for validation and dashboard inspection. In a thesis context, this allows reported metrics to be checked against explicit queries over stored data.

Time-series persistence also supports reproducibility. Once sensor streams and prediction streams are stored, experiments can be reviewed after execution, record counts can be verified, and dashboard outputs can be compared with database queries.

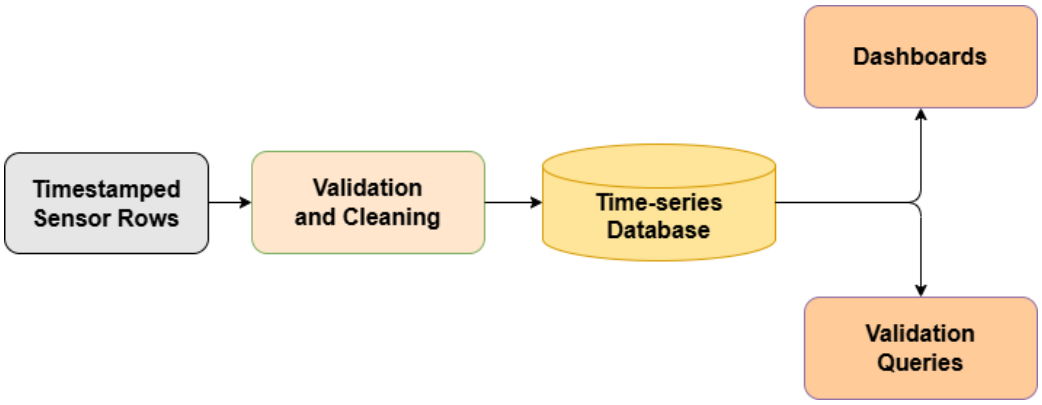

Figure 2.4: Time-series storage workflow for continuous sensor streams and validation queries.

Figure 2.4 highlights why persistence is part of the validation method, not only an implementation detail. Once sensor rows are cleaned and stored with timestamps, the same data can feed dashboards for live inspection and explicit queries for later verification.

Table 2.4 compares the strengths and limitations of the main storage approaches available for timestamped sensor data.

Table 2.4: Storage approaches for timestamped sensor data.

Storage approach	**Strength**	**Limitation**
Time-series database	Optimized for timestamped telemetry and monitoring queries.	Adds a specialized infrastructure component.
Relational database	Mature SQL support and transactional guarantees.	Less naturally optimized for high-frequency telemetry streams.
File-based storage	Simple for offline exports and archival datasets.	Less suitable for live querying and dashboards.
In-memory storage	Useful for temporary buffering.	Not appropriate as the primary persistence layer.

2.6 Observability and Dashboard-Based Validation

Observability refers to the ability to understand a system's internal behavior from the signals it exposes. In distributed IoT systems, observability is important because data may pass through several services before reaching storage, inference, or visualization. If failures occur, it must be possible to identify where they happened [16].

For thesis validation, observability has a methodological role. A system should not only be described through diagrams; it should expose measurable evidence of runtime behavior. Relevant indicators include ingestion rate, queue depth, failed writes, retry count, dropped messages, prediction frequency, and dashboard refresh behavior.

Grafana is widely used to build dashboards from connected data sources, including time-series databases. Its value in this thesis is not only visual presentation. It also provides a practical way to inspect whether stored signals and prediction outputs are arriving as expected during live or replay-based experiments.

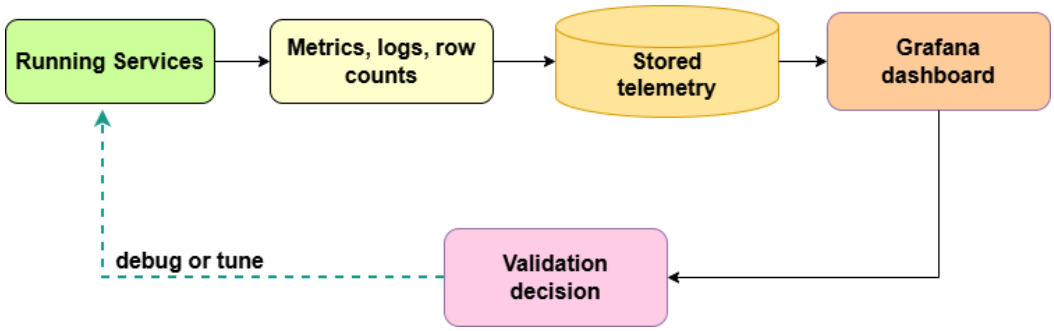

Figure 2.5: Observability feedback loop linking runtime signals, storage, dashboards, and validation decisions.

Figure 2.5 presents observability as a feedback loop. Services expose signals such as row counts, queues, logs, and write errors; these signals are stored or displayed; and the resulting evidence guides debugging or tuning decisions. In this thesis, that loop is used to support claims about reliability with measured behavior.

Table 2.5 summarizes the operational metrics used to observe the IoT pipeline and explains the role of each metric in system validation.

Table 2.5: Operational metrics for observable IoT validation.

Metric	**Meaning**	**Validation role**
Throughput	Number of rows or messages processed per second.	Shows whether the pipeline sustains the expected data rate.
Queue depth	Number of waiting records in an internal buffer.	Indicates whether downstream processing keeps up.
Failed writes	Number of storage operations that did not succeed.	Reveals persistence reliability.
Retry count	Number of repeated write attempts after failure.	Indicates temporary storage or network instability.
Dropped records	Number of discarded or invalid records.	Supports data-loss analysis.
Prediction interval	Time between consecutive inference outputs.	Shows the effective cadence of the HAR pipeline.

The concrete values of these metrics are reported later in Chapter 4, where they are connected to the implemented validation runs.

2.7 Wearable Sensors and IMU-Based Activity Recognition

Wearable sensing is widely used in activity recognition because body-mounted devices can continuously capture movement and physiological signals in naturalistic environments. IMU-based sensing is especially common because accelerometers and gyroscopes provide compact representations of motion, orientation, and movement intensity.

HAR transforms such sensor streams into semantic activity labels. Surveys of wearable HAR show that performance is influenced by several factors beyond the classifier itself, including sensor placement, sampling rate, segmentation window, preprocessing, feature representation, label quality, and subject variability [17].

This has an important implication for system design. A model that performs well offline may behave differently in a live pipeline if the runtime data format, channel order, sampling rate, or preprocessing assumptions differ from those used during evaluation. For this reason, HAR pipelines require disciplined data contracts and explicit preprocessing boundaries.

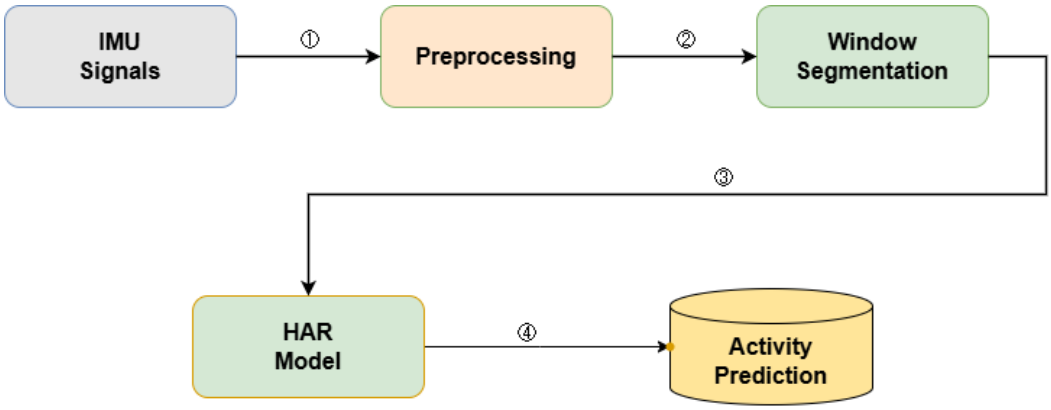

Figure 2.6: Typical wearable HAR workflow from IMU acquisition to activity prediction.

Figure 2.6 breaks HAR into the stages that must agree with one another for inference to be meaningful. The model is only one part of the workflow: preprocessing, channel order, and window construction must also match the assumptions used when the model was evaluated [18].

Table 2.6 summarizes the principal factors that affect wearable HAR and links each factor to its system-level implication. Together, sensor placement, sampling rate, preprocessing, segmentation, and subject variability determine the characteristics of the model input, the prediction cadence, and the extent to which performance generalizes across users and deployment conditions.

Table 2.6: Factors affecting wearable HAR systems.

Factor	Description	System implication
Sensor placement	Position of the wearable on the body.	Affects signal patterns and model transferability.
Sampling rate	Frequency at which measurements are captured.	Influences window construction and temporal resolution.
Preprocessing	Filtering, normalization, or channel ordering.	Must match model assumptions.
Segmentation	Division of streams into inference windows.	Determines prediction frequency and context length.
Subject variability	Differences between users and movement styles.	Affects generalization and real-world accuracy.

This thesis therefore treats HAR as a system-level workflow rather than only a classification task.

2.8 Model Portability and ONNX-Based Inference

Deploying machine-learning models inside software systems requires a stable interface between training artifacts and runtime inference services. Without such an interface, the inference layer may become tightly coupled to the training framework, making deployment and maintenance more difficult.

ONNX provides an open format for representing machine-learning models across frameworks and runtimes [6]. This is useful for service-oriented architectures because the model can be exported once and then consumed by an independent inference service.

In the context of HAR, ONNX-based deployment supports separation between the model artifact and the surrounding IoT platform. The inference service can focus on receiving correctly formatted windows, executing the model, and storing predictions, while data acquisition, cleaning, storage, and visualization remain separate responsibilities.

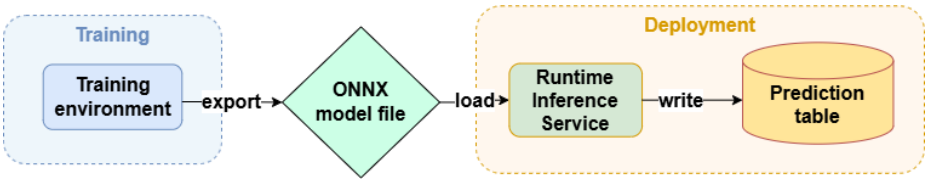

Figure 2.7: ONNX-based model deployment as a boundary between model training and runtime inference.

Figure 2.7 shows ONNX as the hand-off point between model development and system deployment. The training environment exports a model file, while the runtime service loads that file and writes prediction rows. This separation keeps the IoT platform from depending on the training framework itself.

2.9 BLE Acquisition and Wearable Gateway Design

Many wearable devices communicate through Bluetooth Low Energy (BLE), which is designed for low-power wireless communication [19]. BLE is suitable for wearable acquisition, but it is not by itself a complete distributed-system interface. Device-specific callbacks and communication details must usually be translated before the data can enter a broader IoT pipeline.

A gateway or bridge component performs this translation. It separates hardware-specific acquisition from system-level messaging. This pattern is useful because downstream services should not need to know the internal BLE communication details of the wearable device.

The wearable device used in this thesis is the MbientLab MetaMotionS (MMS), a compact BLE sensor node measuring 27 mm × 27 mm × 4 mm and weighing approximately 0.2 oz. It integrates an accelerometer, gyroscope, magnetometer, and sensor-fusion functionality, together with barometric pressure, temperature, and ambient-light sensors. According to the vendor, its rechargeable 70–100 mAh lithium-polymer battery supports approximately 24–48 hours in streaming mode or 24–72 hours in recording mode. Suggested applications include patient monitoring, motion capture, balance and gait analysis, clinical and scientific studies, indoor navigation, fall prevention, and robotics [20].

Within the proposed IoT architecture, a BLE-to-message-broker bridge allows the MetaMotionS to participate in a publish/subscribe pipeline while preserving separation between hardware acquisition and software processing.

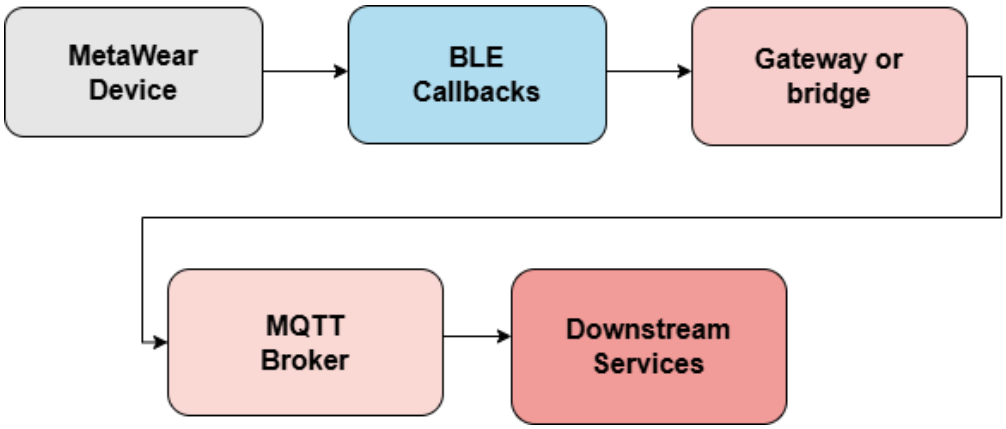

Figure 2.8: Wearable gateway pattern translating BLE device communication into broker-based IoT messaging.

Figure 2.8 explains why the wearable path needs a bridge service. BLE callbacks are device-specific, while downstream services expect system-level messages. The gateway absorbs that hardware-specific logic and publishes data into MQTT so the rest of the platform can remain device-agnostic.

2.10 Heterogeneous Sensor Extension

Wearable IoT platforms may include more than one type of signal. IMU, EEG, and ECG streams differ in sampling assumptions, units, preprocessing needs, and interpretation. Therefore, heterogeneous sensing cannot be handled correctly by forcing all sources into a single generic schema.

A more maintainable approach is to preserve sensor-specific cleaning and schema design while reusing shared infrastructure such as message brokers, storage systems, container orchestration, and dashboards. This allows the platform to grow without making existing pipelines responsible for data they were not designed to process.

This principle is particularly important for extensible research systems. New sensor families should be added through new adapters, cleaners, and storage contracts rather than by modifying unrelated components. Such a pattern supports modularity and reduces schema coupling.

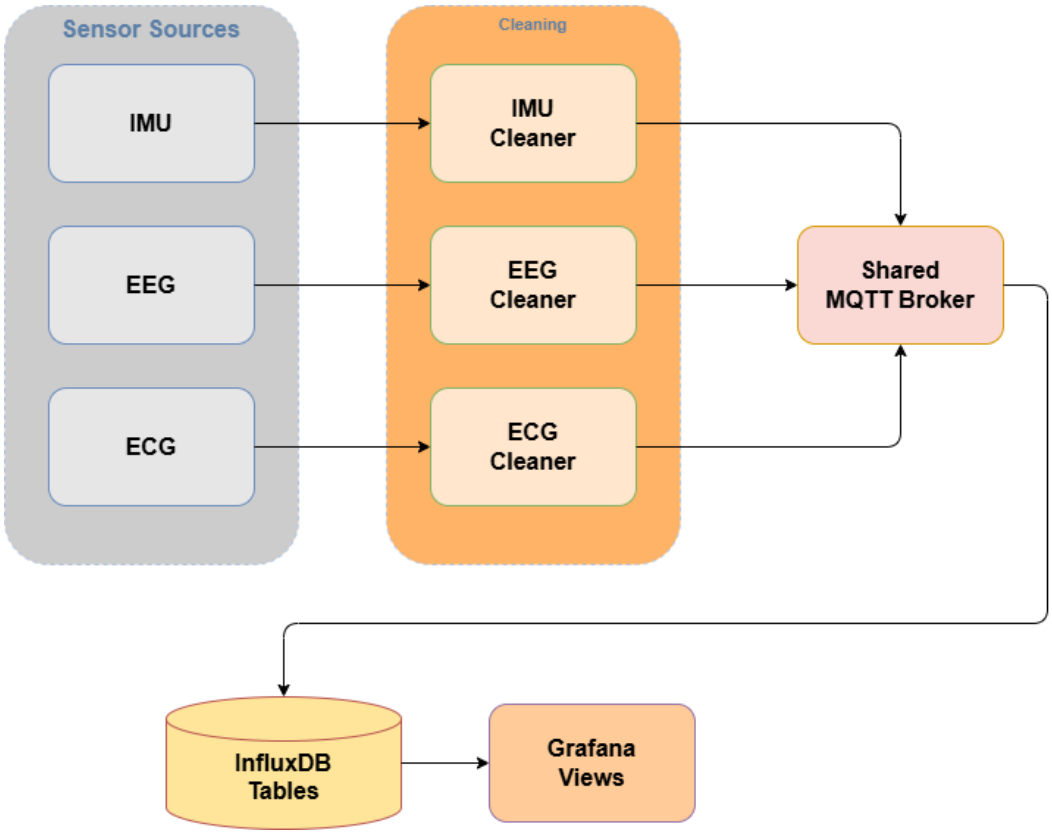

Figure 2.9: Extensible sensor-integration pattern in which heterogeneous signals reuse shared infrastructure while preserving sensor-specific schemas.

Figure 2.9 shows the extension principle used later for EEG and ECG. Each signal family keeps its own cleaner and schema, but the cleaned outputs still use the shared broker, storage, and dashboard infrastructure. This avoids forcing physiological signals into the same structure as IMU data.

2.11 Chapter Summary

This chapter reviewed the literature required to justify the design direction introduced in Chapter 1. IoT architectural models support the separation of sensing, communication, processing, storage, and application layers. Microservice and containerization principles explain why independent services and reproducible deployment are suitable for research-scale IoT systems. MQTT provides a publish/subscribe communication model for decoupled telemetry. Time-series databases support persistent, queryable sensor streams. Dashboard-based observability provides measurable evidence of runtime behavior. Finally, wearable HAR literature shows that activity recognition depends not only on the model, but also on acquisition,

preprocessing, segmentation, and deployment assumptions.

The next chapter translates these principles into the concrete methodology of the *Wearable IoT Activity Pipeline* system, including its implemented services, data flows, topic contracts, storage design, inference modes, and validation strategy.

Chapter 3

Methodology and System Design

3.1 Chapter Overview

This chapter presents the methodology used to design and implement the *Wearable IoT Activity Pipeline*. The previous chapter established the theoretical basis for using microservices, publish/subscribe communication, time-series storage, observability, and wearable HAR. This chapter translates those principles into the concrete architecture, service responsibilities, data contracts, storage model, inference modes, and validation strategy used in the implemented system.

The resulting system is designed as a research-scale IoT platform rather than a production deployment. The methodological objective is therefore not to optimize every service for industrial-scale operation, but to implement a reproducible, observable, and extensible pipeline and validate its behavior using measurable evidence.

3.2 Methodological Approach

The work follows an iterative design-and-validation methodology. Each phase adds one architectural capability and validates it before the next capability is introduced. This reduces the risk of hidden coupling because the broker, ingestion layer, dataset replay, HAR service, real wearable path, visualization layer, and heterogeneous sensor extension can each be checked separately.

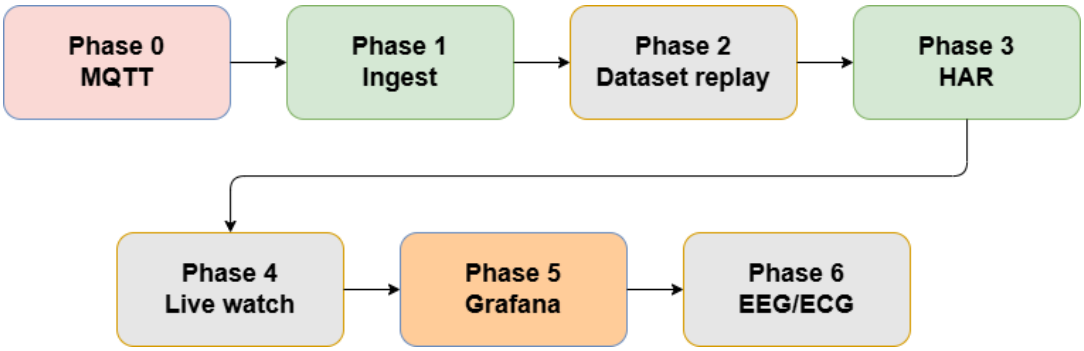

Figure 3.1: Incremental design-and-validation methodology used in the project.

Figure 3.1 shows the phase sequence. The methodology is cumulative: later phases reuse the validated services from earlier phases rather than replacing them. This allows the final platform to support both live sensor acquisition and reproducible dataset replay.

3.3 Design Requirements

The design requirements are derived from the research problem introduced in Chapter 1. They are grouped into functional, architectural, and validation requirements.

Table 3.1 translates these requirements into the concrete mechanisms implemented in the platform.

Table 3.1: Design requirements for the Wearable IoT Activity Pipeline platform.

Requirement	**Meaning**	**Implemented mechanism**
Live acquisition	The platform must ingest data from a real wearable device.	MetaWear bridge and watch cleaner.
Dataset replay	Experiments must be repeatable without hardware.	SIDDHA and OpenNeuro simulators.
Decoupled transport	Producers and consumers should not call each other directly.	EMQX MQTT broker.
Clean storage	Persisted rows must follow stable schemas.	Dedicated cleaner services and InfluxDB tables.
HAR inference	The system must execute an exported activity-recognition model.	ONNX-based HAR service.
Observability	Runtime behavior must be measurable.	Health endpoints, counters, and Grafana dashboards.
Extensibility	New sensor families should not modify existing IMU or HAR paths.	Separate EEG/ECG services and tables.

These requirements guide both the architecture and the validation strategy. The system is considered successful only if the implemented data paths can be measured through stored rows, prediction outputs, queue metrics, and dashboard views.

3.4 Implementation Environment

Following the Docker Compose deployment model described in Section 2.3, all software services in the platform are packaged as containers and orchestrated as one application. The twelve service containers include the broker, database, ingest and HAR services, Grafana, the live-watch bridge and cleaner, and the dataset simulators and cleaners. Only the physical MetaMotionS device and the host Bluetooth hardware remain outside the containerized software environment.

The principal tools used in this work are:

- **Docker Compose**, which defines, connects, configures, and starts the complete set of service containers.
- **EMQX**, which runs as the MQTT broker container and provides topic-based transport between producers and consumers [14].
- **InfluxDB 3**, which runs as the time-series database container for clean sensor rows and HAR predictions [5].

- **FastAPI**, which provides the API framework inside the containerized ingest service and exposes its operational endpoints [13].
- **ONNX**, which provides the portable model format executed inside the containerized HAR service [6].
- **Grafana**, which runs as the visualization container and reads persisted data from InfluxDB [7].
- **MNE-Python**, which is installed in the EEG and ECG dataset containers to read electrophysiological source files [21].
- **MetaWear integration tools**, which run in the bridge and watch-cleaner containers and convert BLE sensor callbacks into canonical MQTT records [9].

Table 3.2 summarizes the role of each main technology and the reason it was selected for the implementation.

Table 3.2: Main technologies used in the implementation.

Technology	**Role**	**Reason for use**
Docker Compose	Container orchestration	Reproducible local deployment of all software services.
EMQX	Containerized MQTT broker	Decoupled topic-based telemetry.
FastAPI	Framework in the ingest container	Health, statistics, schema, and sensor endpoints.
InfluxDB 3	Containerized time-series database	Persistent storage for sensor and prediction tables.
ONNX	Model format used in the HAR container	Portable HAR model artifact.
Grafana	Containerized visualization service	Dashboards backed by persisted InfluxDB data.
MNE-Python	Library in the EEG/ECG containers	Access to electrophysiological file formats [21].

The containerized deployment is separated into a core set of services and optional profiles. The core containers include the broker, database, ingest service, HAR service, watch cleaner, and Grafana. Dataset replay and EEG/ECG extension containers can be started through Compose profiles, allowing experiments to run only the services required for a specific validation scenario. The MetaWear bridge container is used when the live wearable path is enabled.

3.5 Service Architecture

The system follows the processing pattern introduced in Chapter 1: data source, adapter or simulator, MQTT broker, cleaner or normalizer, storage or processing, and visualization. Each microservice owns one responsibility. This avoids placing acquisition, validation, storage, inference, and visualization logic inside one large application.

Table 3.3 identifies the implemented services and defines the responsibility assigned to each service boundary.

Table 3.3: Implemented services and responsibilities.

Service	**Responsibility**
`emqx`	MQTT broker for raw and clean sensor messages.
`ingest-service`	Subscribes to clean sensor topics and writes sensor rows to InfluxDB.
`siddha-sensor-sim`	Replays SIDDHA IMU dataset rows through MQTT.
`metawear_bridge`	Converts MetaWear BLE callbacks into raw MQTT messages.
`watch-cleaner-service`	Pairs accelerometer and gyroscope samples into canonical IMU rows.
`har-service`	Executes ONNX inference and writes prediction rows.
`eeg-dataset-sim`	Replays EEG samples from OpenNeuro ds006848.
`eeg-cleaner-service`	Validates and normalizes EEG rows.
`ecg-dataset-sim`	Replays ECG samples from OpenNeuro ds006848.
`ecg-cleaner-service`	Validates and normalizes ECG rows.
`influxdb3`	Stores clean sensor data and prediction data.
`grafana`	Visualizes sensor signals, predictions, and counters.

The central architectural rule is that the ingest service owns clean sensor storage, while the HAR service owns prediction storage. This prevents the ingest service from becoming a generic writer for every output type and keeps model outputs separate from model inputs.

3.6 Implemented Data Flows

Three data flows are implemented and validated. Each flow uses the same broker-and-storage backbone but has a different source and processing path.

3.6.1 Live MetaWear Watch Flow

The live wearable path begins with a MetaWear bracelet. The device sends accelerometer and gyroscope readings through BLE. A local bridge adapts those callbacks into MQTT raw messages. The watch cleaner then validates, pairs, and normalizes the raw accelerometer and gyroscope samples into complete IMU rows.

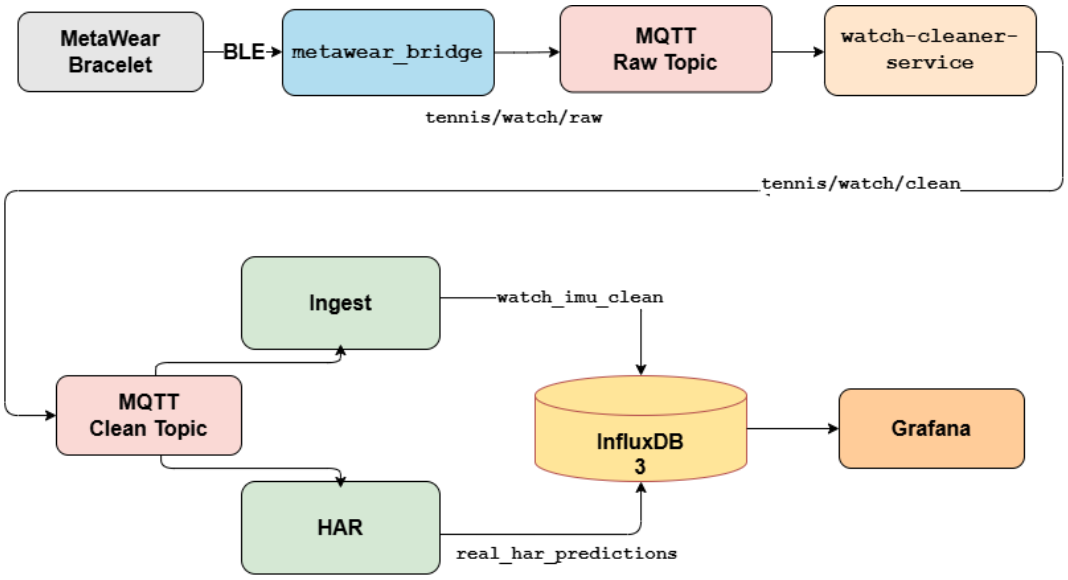

Figure 3.2: Live MetaWear watch data flow.

Figure 3.2 shows the primary real-time workflow. The clean watch topic has two consumers: the ingest service stores clean IMU rows, and the HAR service consumes the same stream for live inference. This demonstrates the benefit of broker-based fan-out: storage and inference can run in parallel without either service calling the other.

Using the service names and responsibilities defined in Table 3.3, the containerized services involved in the live-watch use case are:

- **Acquisition and cleaning:** `metawear_bridge` receives BLE callbacks, and `watch-cleaner-service` creates canonical IMU rows.
- **Message transport:** `emqx` carries the raw and clean watch topics.
- **Storage:** `ingest-service` writes clean rows to `influxdb3`.
- **Inference and visualization:** `har-service` produces live predictions, and `grafana` visualizes the stored signals and predictions.

3.6.2 SIDDHA Dataset Replay Flow

The SIDDHA dataset replay path supports reproducible evaluation using smartwatch and smartphone IMU records from the SIDDHA dataset [22]. Dataset rows are read from Parquet files [23], published through MQTT by `siddha-sensor-sim`, and stored in the `imu_raw_full_rows` table. The HAR service can then read stored rows in database polling mode, build windows, execute the ONNX model, and write predictions to `har_predictions_7_activity`.

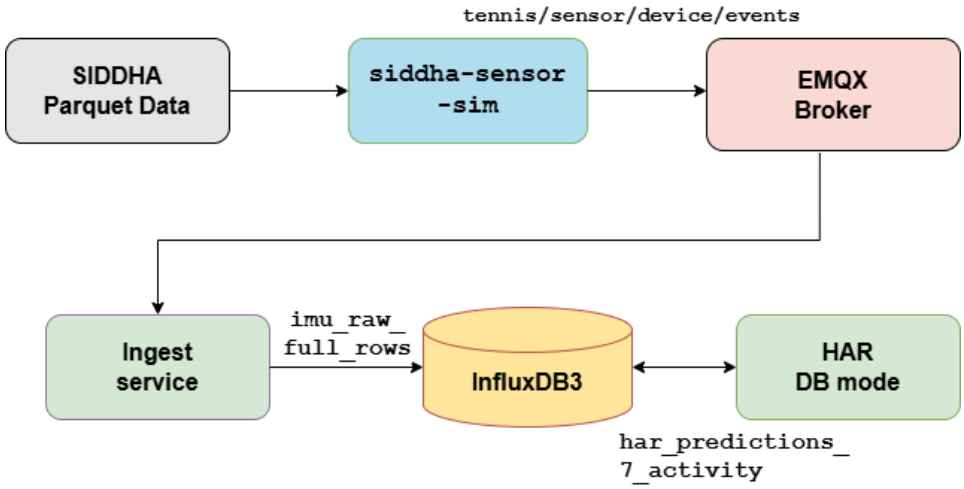

Figure 3.3: SIDDHA replay and database-driven HAR evaluation flow.

Figure 3.3 shows the replay workflow used for reproducible evaluation. This path is intentionally database-driven: it allows the same stored rows to be queried repeatedly, supports deterministic ordering by time and sample index, and separates model evaluation from live hardware timing.

For the SIDDHA replay use case, the relevant services from Table 3.3 are:

- **Dataset replay:** `siddha-sensor-sim` reads the Parquet data and publishes IMU rows.
- **Message transport and storage:** `emqx` routes the replay messages, `ingest-service` writes the replay rows, and `influxdb3` persists them.
- **Reproducible inference:** `har-service` reads the stored rows in database-polling mode and writes the resulting predictions back to `influxdb3`.

3.6.3 EEG and ECG Extension Flow

The EEG and ECG extension path uses the OpenNeuro ds006848 dataset [24] as a dataset-based mock sensor source. OpenNeuro supports open neuroscience

data sharing [25], and BIDS-based organization is a widely used standard for neuroimaging and electrophysiology datasets [26, 27]. In this project, EEG and ECG are used only to validate extensibility, storage, and visualization. No EEG or ECG machine-learning model is implemented.

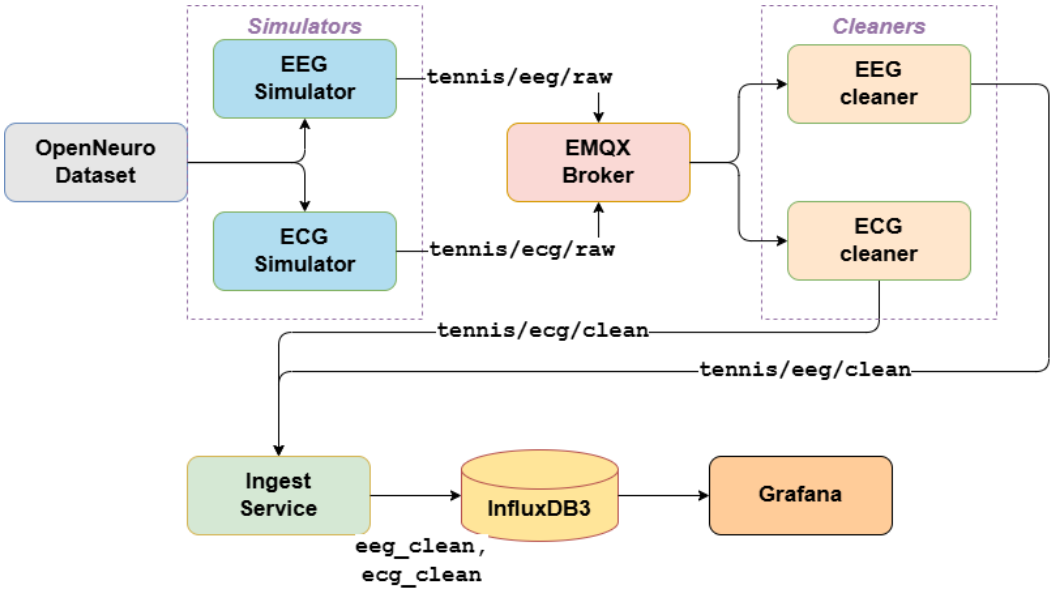

Figure 3.4: EEG and ECG extension path for heterogeneous sensor validation.

Figure 3.4 demonstrates the main extensibility principle. New sensor families are added through new simulators, cleaners, topics, and tables, while the live watch pipeline and the HAR ownership model remain unchanged.

For the heterogeneous-sensor use case, the services identified in Table 3.3 are:

- **Dataset simulation:** `eeg-dataset-sim` and `ecg-dataset-sim` replay the two physiological signal streams.
- **Sensor-specific cleaning:** `eeg-cleaner-service` and `ecg-cleaner-service` validate and normalize their respective schemas.
- **Message transport and storage:** `emqx` routes the raw and clean topics, `ingest-service` writes the canonical rows, and `influxdb3` stores them in separate tables.
- **Visualization:** `grafana` displays the persisted EEG and ECG signals and row counts.

3.7 MQTT Topics and Data Contracts

Following the MQTT topic-routing model introduced in Section 2.4, this implementation uses topic names as explicit contracts between services. Raw topics carry source-specific data, while clean topics carry canonical rows that are safe to store or process. This distinction prevents downstream services from needing to understand hardware-specific or dataset-specific raw formats.

Table 3.4 lists the main MQTT topics together with their producers and their role in the corresponding data flow.

Table 3.4: Main MQTT topics and their roles.

Topic	Producer	Consumer or role
`tennis/watch/raw`	`metawear_bridge`	Raw watch input for the watch cleaner.
`tennis/watch/clean`	`watch-cleaner -service`	Stored by ingest service and consumed by HAR stream mode.
`tennis/sensor/device /events`	`siddha-sensor-sim`	Dataset replay input for ingest service; `device` is replaced by the source device name.
`tennis/eeg/raw`	`eeg-dataset-sim`	Raw EEG input for EEG cleaner.
`tennis/eeg/clean`	`eeg-cleaner-service`	Clean EEG rows for storage.
`tennis/ecg/raw`	`ecg-dataset-sim`	Raw ECG input for ECG cleaner.
`tennis/ecg/clean`	`ecg-cleaner-service`	Clean ECG rows for storage.

The data-contract strategy follows a consistent pattern:

> source format → raw MQTT payload → cleaner or normalizer → canonical clean payload → InfluxDB table.

For the real watch path, the cleaner pairs accelerometer and gyroscope samples into one complete IMU row. For EEG and ECG, separate cleaners preserve sensor-specific schemas because physiological signals should not be forced into an IMU data model.

3.8 Storage and Ownership Model

Following the InfluxDB data model introduced in Section 2.5, the implementation creates six tables for sensor inputs and prediction outputs. These tables form three logical data groups:

- **Dataset IMU and HAR:** `imu_raw_full_rows` stores replayed SIDDHA IMU samples, and `har_predictions_7_activity` stores predictions derived from windows of those samples.
- **Live watch and HAR:** `watch_imu_clean` stores canonical MetaWear IMU rows, and `real_har_predictions` stores predictions derived from the live watch windows.
- **Heterogeneous sensor extension:** `eeg_clean` and `ecg_clean` store independent physiological streams. They are not connected to prediction tables because no EEG or ECG model is implemented.

Table 3.5 summarizes each table's data category and its relationship to the other stored data.

Table 3.5: InfluxDB tables and logical relationships in the final system.

Table	Data category	Relationship or purpose
`imu_raw_full_rows`	Sensor input	SIDDHA IMU rows used for dataset-based HAR.
`watch_imu_clean`	Sensor input	MetaWear IMU rows used for live HAR.
`eeg_clean`	Sensor input	Independent EEG extension stream.
`ecg_clean`	Sensor input	Independent ECG extension stream.
`har_predictions_7_activity`	Prediction output	Derived from windows of SIDDHA IMU rows.
`real_har_predictions`	Prediction output	Derived from windows of live watch IMU rows.

The table relationships are logical rather than foreign-key relationships. Prediction rows preserve metadata such as device, recording identifier, timestamp, model name, and input layout so that they can be associated with the sensor windows from which they were produced. Stable identifiers such as device, subject, task, and recording identifier are stored as tags, while frequently changing values such as sensor axes, sample indices, predicted labels, and confidence scores are stored as fields.

After the table structure is defined, write ownership is divided between two services. The ingest service owns the four sensor-data tables and writes the rows received from the clean or replay topics. The HAR service reads the relevant IMU inputs and exclusively writes the two prediction tables. This separation prevents

the ingest service from becoming responsible for model output schemas and makes the source of every stored row explicit.

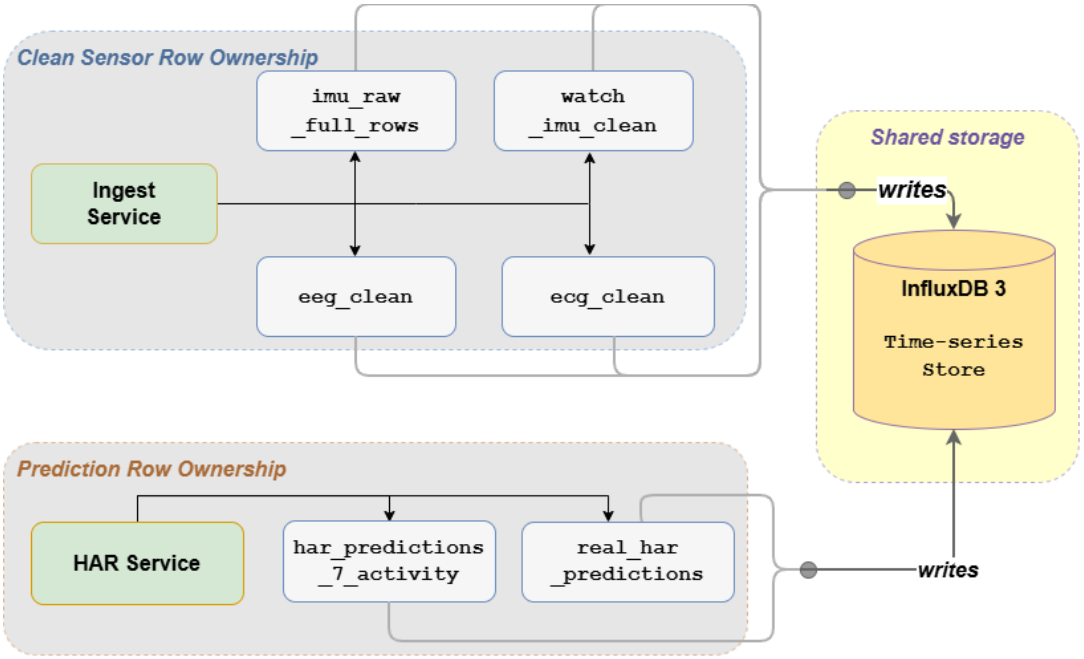

Figure 3.5: Storage ownership model separating sensor rows from prediction rows.

Figure 3.5 summarizes this ownership boundary. InfluxDB is shared infrastructure, but the ingest and HAR services do not share responsibility for writing the same tables, which simplifies debugging and schema changes.

3.9 Ingest-Service Design

Following the FastAPI introduction in Section 2.3.1, the ingest service uses the framework to provide a synchronous HTTP interface for operational inspection and stored-data queries. Its API exposes endpoints for health, statistics, table discovery, schema inspection, devices, and sensor queries. In parallel, a background MQTT subscriber receives sensor messages and passes them to the InfluxDB writer. The service uses a bounded writer queue, batch writes, retry counters, drop counters, and an explicit writer-thread health signal.

The ingest service subscribes to configured MQTT topics and writes clean sensor rows to the correct InfluxDB table. It does not own prediction storage, because predictions have a different schema and are produced by the HAR service. This design preserves service ownership and keeps the ingestion layer focused on sensor data.

Table 3.6 summarizes the operational endpoints and the validation or observability information exposed by each one.

Table 3.6: Operational endpoints used for validation and observability.

Endpoint	Purpose
`/health`	Reports service and writer health.
`/stats`	Reports table counts and writer metrics such as queue depth and failed writes.
`/tables`	Lists configured InfluxDB table names.
`/schema`	Reports table schema information.
`/devices`	Lists known devices across stored sensor tables.
`/sensors/imu`	Queries SIDDHA IMU rows.
`/sensors/watch`	Queries real watch clean IMU rows.
`/sensors/eeg`	Queries EEG mock-sensor rows.
`/sensors/ecg`	Queries ECG mock-sensor rows.

3.10 HAR Service Design

The HAR service loads an ONNX model and supports two execution modes: database polling and MQTT streaming. Database polling is used for reproducible evaluation on stored SIDDHA rows. MQTT streaming is used for live MetaWear watch inference.

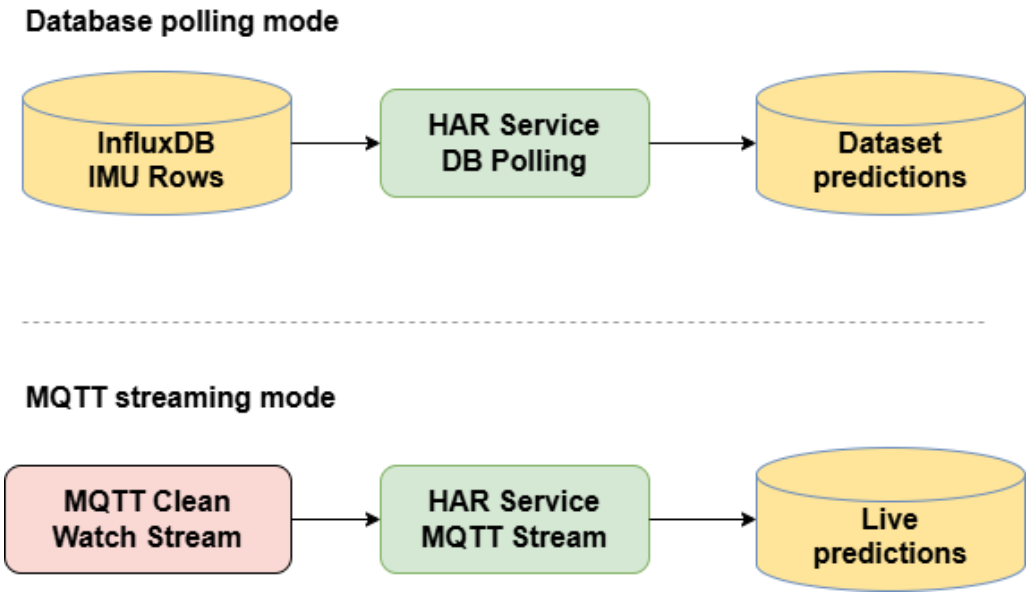

Figure 3.6: Two HAR execution modes used for reproducible evaluation and live inference.

Figure 3.6 separates the two ways the HAR service is used. Database polling mode is for repeatable evaluation on stored rows, while MQTT streaming mode is for live inference from the clean watch stream. Both modes build sliding windows from IMU rows before inference. The validated runtime configuration uses a window size of 40 samples, a stride of 20 samples, `gyro_then_accel` input layout, no temporal preprocessing, and summed score aggregation. In database mode, rows are filtered to supported activity labels and grouped by device and recording. In streaming mode, the service consumes clean watch rows and writes live predictions as new windows become available.

3.11 Visualization Methodology

Grafana uses InfluxDB as the source of truth. This choice intentionally avoids making the dashboard depend only on transient MQTT messages. If a panel shows data, the corresponding rows have already been stored and can be verified by SQL queries. This strengthens the link between visualization and validation.

The primary dashboard visualizes live watch accelerometer and gyroscope signals, the latest predicted activity, prediction confidence, prediction history, and stored row counters. The EEG/ECG dashboard validates heterogeneous sensor ingestion by showing row counts and signal previews for the two additional sensor families.

3.12 Validation Methodology

Validation is performed through phase-specific criteria. Each phase has an expected data path, expected storage outcome, and observable runtime metrics. The validation does not rely only on service startup; it checks whether messages are transported, rows are stored, predictions are written, queues drain, and dashboards read persisted data.

Table 3.7 defines the metrics applied across the validation phases and the behavior expected from a healthy run.

Table 3.7: Validation metrics used across the platform.

Metric	**Meaning**	**Expected healthy behavior**
Stored row count	Number of rows persisted in the target table.	Matches expected run size or is greater than zero for exploratory replay.
Prediction count	Number of HAR rows written by the HAR service.	Consistent with window size and stride.
Queue depth	Number of rows waiting in the ingest writer.	Returns to zero after replay or sustained live run.
Failed batches	Number of failed database batch writes.	Zero during validation.
Retried lines	Number of rows retried after write instability.	Zero during validated runs.
Dropped lines	Number of records discarded by the writer.	Zero during validated runs.
Dashboard visibility	Whether Grafana displays stored rows.	Panels display current signals or counters.

The next chapter reports the results obtained from this validation methodology.

Chapter 4

Validation and Results

4.1 Chapter Overview

This chapter reports the validation results of the Wearable IoT Activity Pipeline platform. The evaluation follows the methodology defined in Chapter 3: each architectural capability is validated through observable runtime behavior, stored database rows, prediction outputs, and dashboard visibility. The reported values are taken from the validation outputs produced during the implementation and testing of the thesis platform.

The chapter is organized around the validated capabilities rather than around source-code modules. This makes the results easier to connect to the research objectives: reliable ingestion, reproducible replay, live wearable processing, HAR inference, dashboard observability, and extensibility to heterogeneous sensor families.

4.2 Validation Strategy

The validation strategy combines functional checks and quantitative runtime metrics. A phase is considered valid only when its data path works end to end. For example, a service that starts successfully is not sufficient evidence by itself; the expected messages must also arrive, the expected tables must be populated, and writer-health metrics must remain stable.

Table 4.1 summarizes the capability validated in each phase and the evidence used to determine its outcome.

Table 4.1: Summary of validation phases and evidence.

Phase	Validated capability	Evidence type
Phase 0	MQTT broker transport.	Publisher/subscriber delivery.
Phase 1	Ingest service and InfluxDB storage.	Health endpoints, stored rows, queue metrics.
Phase 2	SIDDHA IMU dataset replay.	Structured rows in `imu_raw_full_rows`.
Phase 3	ONNX HAR service in DB polling mode.	Prediction rows and classification accuracy.
Phase 4	Real MetaWear watch pipeline.	Live clean IMU rows, live predictions, writer health.
Phase 5	Grafana visualization.	InfluxDB-backed panels and 1-second refresh.
Phase 6	EEG/ECG heterogeneous extension.	Separate clean tables and dashboard query set.

Validation was performed as phase-based integration testing rather than by a dedicated validation microservice or an autonomous watchdog. For each phase, the relevant Docker Compose services were started, the corresponding live or replay input was triggered, and the resulting evidence was reviewed against the expected criteria. Test orchestration and final inspection were therefore manual, while evidence generation during each run was automatic.

The automatic evidence came from instrumentation built into the implemented services. MQTT publisher/subscriber checks confirmed message delivery; the ingest service's `/health` and `/stats` endpoints reported writer status, queue depth, failed batches, retries, and dropped lines, as described in Table 3.6; InfluxDB queries provided sensor-row and prediction counts; and Grafana panels confirmed that persisted signals could be visualized. No additional watchdog service was introduced. The final system is therefore evaluated as a thesis-scale architecture through manually supervised tests supported by automatically generated time-series and operational evidence, rather than as a production-ready deployment with continuous autonomous monitoring.

4.3 MQTT Transport and Ingestion Validation

The first validation step confirmed that EMQX could operate as the MQTT transport backbone. A test publisher sent messages to the broker and a test subscriber received them. This established the communication base used by dataset simulators, the MetaWear bridge, cleaner services, the ingest service, and the HAR service.

The second validation step added the ingest service and InfluxDB. The ingest service starts a FastAPI application, launches an MQTT subscriber in the background, and writes received clean sensor rows to InfluxDB using a bounded writer queue. Validation used the service health and statistics endpoints to check that the writer remained alive and that the queue drained after replay.

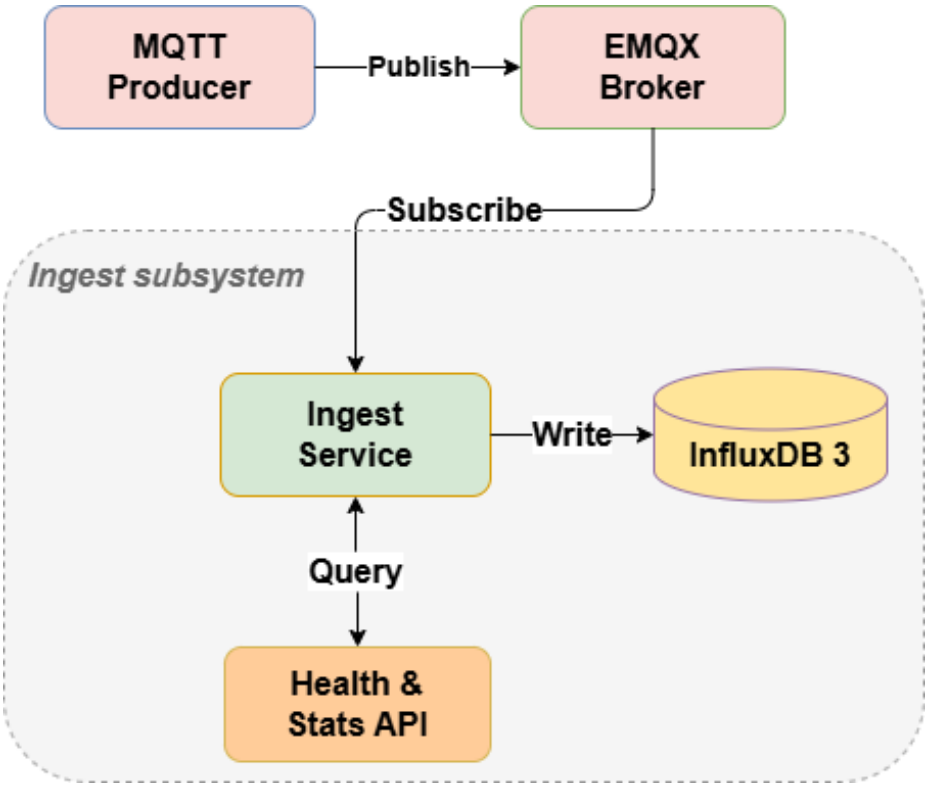

Figure 4.1: Validated MQTT-to-InfluxDB ingestion path.

Figure 4.1 shows the first validated runtime path. The producer and broker confirm MQTT delivery, the ingest service confirms subscription and database writing, and the health/statistics API provides the operational evidence used to check that the writer remains healthy.

Table 4.2 reports the acceptance criteria for the ingest service and the status observed for each check.

Table 4.2: Ingest-service validation criteria.

Check	Expected result	Status
FastAPI service starts	Application running.	Pass
MQTT connection established	Broker connection active.	Pass
Configured topics subscribed	Messages can be received.	Pass
InfluxDB writes succeed	Rows appear in target table.	Pass
`/health` responds	Service status returned.	Pass
`/stats` responds	Counts and writer metrics returned.	Pass
Writer queue drains	`queue_depth = 0` after replay.	Pass

These results validate the storage backbone used by all later phases.

4.4 SIDDHA Replay Validation

The SIDDHA replay phase validates reproducible dataset ingestion. The `siddha-sensor-sim` service reads IMU rows from a Parquet dataset, publishes them to MQTT, and the ingest service stores them in `imu_raw_full_rows`. The stored schema includes device, recording identifier, sample index, dataset timestamp, accelerometer axes, gyroscope axes, and ground-truth activity label.

Table 4.3 presents the end-to-end checks used to validate dataset loading, MQTT replay, structured storage, and writer health.

Table 4.3: SIDDHA replay validation criteria.

Check	Expected result	Status
Dataset loads	Dataset rows available.	Pass
Simulator publishes messages	Messages reach EMQX.	Pass
Ingest service receives rows	MQTT subscription works.	Pass
Rows stored in InfluxDB	`imu_raw_full_rows` populated.	Pass
Structured IMU fields exist	ACC/GYRO fields present.	Pass
Writer queue drains	`queue_depth = 0`.	Pass
Failed batches	Zero.	Pass
Dropped lines	Zero.	Pass

This validation is important because it separates reproducible evaluation from live hardware availability. Once rows are stored, the HAR service can query the same data repeatedly and build windows deterministically.

4.5 HAR Dataset Evaluation

The HAR model was not trained as part of this thesis. The `L2MU_plain_leaky.onnx` artifact originates from the HAR work of Fra et al. [18], which investigated recurrent Legendre Memory Unit (LMU) models for activity recognition from windowed accelerometer and gyroscope data. That study selected seven general hand-oriented activities and used segmentation as the only preprocessing step. This thesis reuses the trained artifact as a fixed inference component so that the evaluation focuses on model integration and data-pipeline behavior. The original artifact name is retained: `L2MU` identifies the supplied LMU-family model, `plain_leaky` identifies the exported network variant, and `.onnx` identifies the portable deployment format.

The runtime configuration terms used in this evaluation mean the following:

- **Window size and stride:** each inference uses 40 consecutive IMU rows, after which the window advances by 20 rows. Consecutive windows therefore overlap by 20 samples, or 50%.

- **`gyro_then_accel` input layout:** the six channels supplied for each sample are ordered as the three gyroscope axes followed by the three accelerometer axes. This order must match the model's expected input tensor.

- **No temporal preprocessing:** the service preserves the sample order and performs window segmentation without applying an additional temporal filter or transformation before inference.

- **Summed score aggregation:** class scores produced across the model output are added for each activity, and the activity with the largest accumulated score is selected as the window prediction.

- **Seven-activity subset:** evaluation is restricted to the seven activities supported by the reused model; their dataset codes are defined in Table 4.5.

Table 4.4 records the model, data tables, filters, and runtime parameters used for the reproducible HAR evaluation.

Table 4.4: HAR dataset evaluation setup.

Parameter	Value
Model	`L2MU_plain_leaky.onnx`
Input table	`imu_raw_full_rows`
Output table	`har_predictions_7_activity`
Device filter	`watch`
Supported labels	`F, G, O, P, Q, R, S`
Window size	40 samples
Stride	20 samples
Input layout	`gyro_then_accel`
Temporal preprocessing	`none`
Score aggregation	`sum`

Table 4.5 maps the seven supported label codes to the activities included in the validated evaluation subset.

Table 4.5: Supported HAR activity subset.

Code	Activity
F	Typing
G	Brushing teeth
O	Playing catch / tennis-related catch
P	Basketball dribbling
Q	Writing
R	Clapping
S	Folding clothes

The validated result for the supported watch-device subset is:

$$\text{Accuracy} = \frac{119}{140} = 85.0\%. \tag{4.1}$$

The repository also records an unfiltered overall result of 21 correct classifications out of 140 windows, equal to 15.0%. The difference is expected because the validated configuration filters to the watch device and to supported activity labels. Without this filter, the evaluation includes unsupported activities and phone-device rows that are outside the validated model configuration.

Table 4.6 compares the supported watch-device result with the unfiltered run, showing the effect of applying the validated scope.

Table 4.6: HAR evaluation result.

Evaluation scope	**Correct / total**	**Accuracy**
Watch-device supported subset	119 / 140	85.0%
Unfiltered overall run	21 / 140	15.0%

This result validates the inference pipeline: stored IMU rows can be fetched in order, sliding windows can be built, the ONNX model can execute inside the HAR service, and prediction rows can be written back to InfluxDB. It does not claim full 18-activity classification, tennis-specific stroke recognition, or guaranteed real-world MetaWear accuracy.

4.6 Live MetaWear Watch Validation

The live watch validation tested the complete real-time path from MetaWear BLE acquisition to MQTT, cleaning, InfluxDB storage, HAR inference, and prediction storage. The validation run used the recording identifier `phase4_live_validation_001`.

Figure 4.2 shows the MetaWear wearable used for the live validation, providing a visual reference for the physical sensing hardware discussed in this section.

Figure 4.2: MetaWear wearable device used for live IMU acquisition and validation.

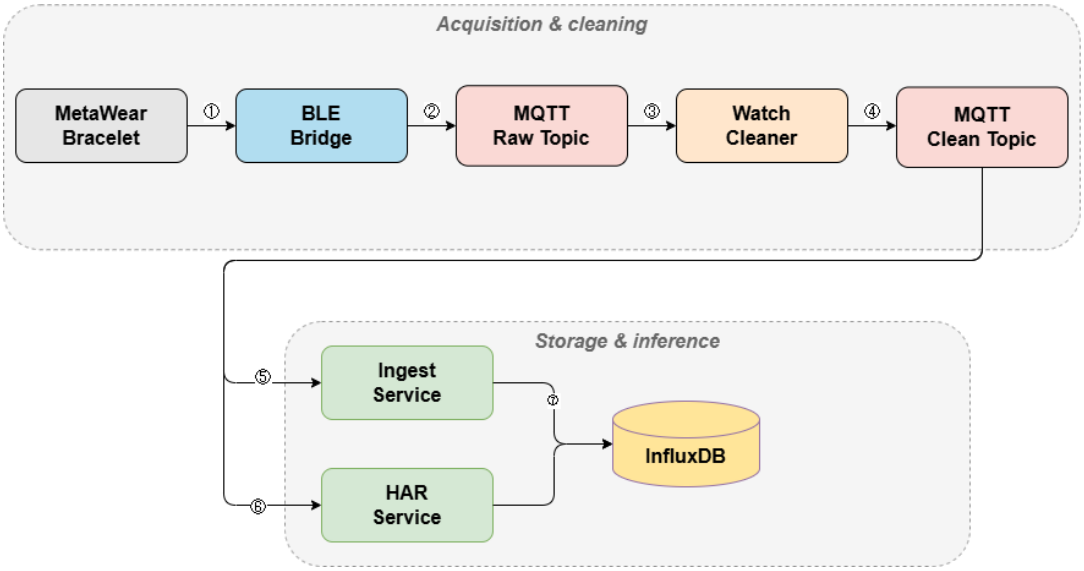

Figure 4.3: Validated live watch path from BLE acquisition to storage and inference.

Figure 4.3 shows the complete live validation path. Data leave the MetaWear device through the BLE bridge, pass through raw and clean MQTT stages, and then split toward storage and HAR inference. The numbered arrows indicate the sequence checked during the validation run.

Table 4.7 records the device, sampling rate, duration, date, and recording identifier that define the live validation run.

Table 4.7: Live MetaWear validation context.

Item	**Value**
Recording ID	`phase4_live_validation_001`
Validation date	2026-05-14
Sensor	MetaWear MMR
Configured sampling rate	25 Hz
Streaming duration	Approximately 385 seconds

Table 4.8 compares the observed live row and prediction counts with their reference values and reports the ingest writer's health indicators.

Table 4.8: Live MetaWear validation metrics.

Metric	Observed value	Status
Clean IMU rows stored	9 599	Pass
Expected clean IMU rows	Approximately 9 625	Reference
Row-count difference	Approximately 0.27%	Acceptable
HAR predictions stored	463	Pass
Expected approximate predictions	Approximately 481	Reference
Approximate clean row rate	24.9 rows/s	Pass
Approximate prediction interval	0.83 s	Pass
Ingest queue depth	0	Pass
Failed batches	0	Pass
Retried lines	0	Pass
Dropped lines	0	Pass
Writer thread alive	true	Pass

The live validation confirms that the platform can process a real wearable stream in near real time. The small row difference from the theoretical count is attributed to startup and pairing behavior during initial streaming. The prediction count is also lower than the theoretical maximum because the HAR service must accumulate a complete first window and operates with runtime timing overhead.

The important result is that no failed batches, retries, or dropped lines were reported during the validation run. This supports the reliability claim for thesis-scale live testing.

4.7 Grafana Visualization Validation

Grafana validation confirmed that the dashboard can visualize persisted sensor data from InfluxDB. The same dashboard environment is used for both the EEG/ECG extension and live MetaWear monitoring, with sensor-specific panels and queries for each data source. Reading persisted data rather than transient MQTT messages keeps visualization consistent with the database-backed validation method.

Figure 4.4 shows the implemented Grafana dashboard populated from the EEG and ECG database tables. The displayed view combines stored-row counters, the latest clean records, and signal previews. The same dashboard structure is also used for live MetaWear watch monitoring, where the corresponding panels display IMU signals and HAR predictions.

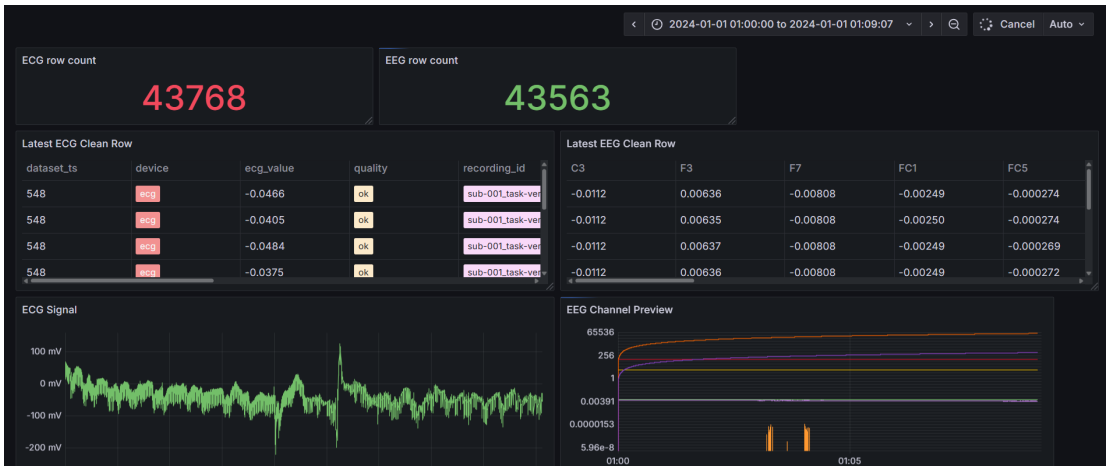

Figure 4.4: Grafana dashboard showing persisted EEG and ECG data. The same dashboard structure is also used for live MetaWear monitoring.

Table 4.9 identifies the corresponding live-watch panels available in the shared dashboard, their source tables, and the validation purpose they serve.

Table 4.9: Grafana dashboard panels for live watch validation.

Panel	**Source table**	**Purpose**
Current predicted activity	`real_har_predictions`	Show latest HAR output.
Current confidence	`real_har_predictions`	Show latest confidence.
Prediction timeline	`real_har_predictions`	Inspect prediction changes.
Live accelerometer	`watch_imu_clean`	Show watch acceleration.
Live gyroscope	`watch_imu_clean`	Show watch rotation.
Stored clean IMU rows	`watch_imu_clean`	Validate ingestion.
Stored predictions	`real_har_predictions`	Validate HAR output.

The dashboard refresh interval was configured to 1 second, which was sufficient for thesis-scale monitoring. MQTT/Grafana Live was considered but not required because InfluxDB-backed panels were already fast enough and more reproducible.

4.8 EEG and ECG Extension Validation

Phase 6 tested whether the architecture could support two additional heterogeneous data families without modifying the real watch pipeline or the HAR ownership model. The extension used OpenNeuro ds006848 as the dataset source and replayed EEG and ECG as mock sensors.

Table 4.10 specifies the dataset selection, duration, sampling rate, channel limit, and target tables used for this extension test.

Table 4.10: EEG/ECG extension configuration.

Configuration item	**Value**
Dataset	OpenNeuro `ds006848`
Task used	`verbalwm`
Example subject	`sub-001`
EEG maximum duration	600 seconds
ECG maximum duration	600 seconds
EEG downsample rate	100 Hz
ECG downsample rate	100 Hz
EEG channel limit	8 channels
EEG table	`eeg_clean`
ECG table	`ecg_clean`

With 600 seconds at 100 Hz, each sensor stream is expected to produce about 60 000 rows:

$$600 \text{ seconds} \times 100 \text{ samples/second} = 60\,000 \text{ samples}. \tag{4.2}$$

Table 4.11 applies this calculation to the EEG and ECG tables and states the expected row count for each stream.

Table 4.11: EEG/ECG expected validation size.

Table	**Expected rows**
`eeg_clean`	Approximately 60 000
`ecg_clean`	Approximately 60 000

Table 4.12 reports the checks used to confirm that both sensor paths reached their raw topics, clean topics, storage tables, and Grafana panels.

Table 4.12: EEG/ECG extension validation criteria.

Check	**Expected result**	**Status**
EEG simulator starts	Running.	Pass
ECG simulator starts	Running.	Pass
EEG cleaner starts	Running.	Pass
ECG cleaner starts	Running.	Pass
EEG raw topic receives data	`tennis/eeg/raw` active.	Pass
ECG raw topic receives data	`tennis/ecg/raw` active.	Pass
EEG clean topic receives data	`tennis/eeg/clean` active.	Pass
ECG clean topic receives data	`tennis/ecg/clean` active.	Pass
EEG rows stored	`eeg_clean` populated.	Pass
ECG rows stored	`ecg_clean` populated.	Pass
Grafana EEG panel works	Signal visible.	Pass
Grafana ECG panel works	Signal visible.	Pass

This result validates the extensibility claim. EEG and ECG are integrated through new simulators, cleaner services, MQTT topics, storage tables, and dashboard queries. The watch pipeline, IMU schema, and HAR prediction tables do not need to be changed.

4.9 Integrated Result Summary

Table 4.13 consolidates the validated outcome for each architectural capability and its associated reliability evidence.

Table 4.13: Final validated outcomes.

Area	Validated outcome
MQTT transport	EMQX broker delivered messages between producers and consumers.
Ingestion	FastAPI ingest service stored clean sensor rows in InfluxDB.
Dataset replay	SIDDHA IMU rows were replayed and stored for reproducible evaluation.
HAR DB mode	ONNX inference produced dataset prediction rows.
Live wearable path	MetaWear data were captured, cleaned, stored, and processed by HAR.
Grafana	InfluxDB-backed dashboards visualized signals, predictions, and counters.
EEG/ECG extension	Heterogeneous dataset-based sensors were integrated with separate schemas.
Reliability metrics	Queue depth, failed batches, retries, and dropped lines remained healthy in validation.

The strongest result is the validated architecture rather than the HAR model alone. The system demonstrates that real wearable acquisition, reproducible dataset replay, time-series persistence, activity-recognition inference, dashboard observability, and heterogeneous sensor extension can coexist within one containerized IoT platform.

4.10 Limitations of the Results

The results must be interpreted within the declared scope of the thesis. The 85.0% HAR accuracy applies to the validated seven-activity watch-device subset, not to all SIDDHA activities and not to tennis-specific stroke recognition. The live MetaWear validation demonstrates pipeline reliability, not supervised real-world activity accuracy, because live watch data did not include ground-truth labels.

The EEG and ECG extension validates ingestion, cleaning, storage, and visualization only. It does not claim EEG classification, ECG classification, clinical interpretation, or real physiological hardware integration. The platform also does not claim production-grade authentication, authorization, edge deployment, or camera-based tracking.

These limitations are intentional. They keep the thesis focused on a measurable infrastructure contribution: an extensible and observable IoT pipeline for wearable and dataset-based sensor processing.

Chapter 5

Conclusions and Future Work

5.1 Chapter Overview

This final chapter summarizes the thesis contributions, answers the research question, discusses the limitations of the implemented platform, and outlines future work. The thesis set out to design and validate a modular IoT architecture for wearable sensing, reproducible dataset replay, HAR inference, and heterogeneous sensor integration. The final system demonstrates that these requirements can be addressed within a single containerized platform.

5.2 Answer to the Research Question

The research question introduced in Chapter 1 was:

> How can a modular and reproducible IoT architecture be designed and validated to support both real-time wearable sensing and offline experimental evaluation while maintaining clear separation between acquisition, processing, storage, inference, and visualization components?

This thesis answers the question by proposing and validating a service-separated architecture based on five principles:

1. Use MQTT as the communication backbone so that producers and consumers are decoupled through topic contracts.

2. Separate adapters, cleaners, storage, inference, and visualization into independently deployable services.

3. Store clean sensor data and prediction data in separate InfluxDB tables with clear service ownership.

4. Support both live streaming inference and database-driven reproducible evaluation.

5. Validate the platform through stored rows, prediction counts, writer-health metrics, and dashboard visibility.

The resulting architecture supports real MetaWear wearable acquisition, SIDDHA dataset replay, ONNX-based HAR inference, Grafana monitoring, and OpenNeuro EEG/ECG extension paths. The design therefore satisfies the original requirement of supporting both real-time operation and offline experimental evaluation without merging all responsibilities into a single application.

5.3 Main Contributions

The thesis contributes a practical IoT architecture and a validation approach for research-scale wearable analytics. The main contributions are summarized in Table 5.1.

Table 5.1: Summary of thesis contributions.

Contribution	**Description**
Modular architecture	A Docker-based microservice platform separating acquisition, cleaning, storage, inference, and visualization.
Dual evaluation mode	Support for both MQTT streaming inference and database-driven reproducible HAR evaluation.
Storage ownership model	A clear distinction between clean sensor tables owned by the ingest service and prediction tables owned by the HAR service.
Live wearable validation	End-to-end validation of MetaWear BLE acquisition, MQTT transport, cleaning, storage, and live HAR prediction.
Observable runtime behavior	Validation using queue depth, failed batches, retries, dropped lines, stored row counts, prediction counts, and dashboards.
Heterogeneous extension pattern	Integration of EEG and ECG dataset-based mock sensors without changing the watch or HAR pipelines.

These contributions are primarily architectural and methodological. The thesis does not claim to introduce a new HAR model. Instead, it shows how an existing ONNX model can be integrated into a reproducible and observable IoT system.

5.4 Validation Summary

The final platform was validated across seven project phases. MQTT transport was validated through broker-based message delivery. The ingest service was validated by storing rows in InfluxDB and exposing writer-health metrics. The SIDDHA replay path validated reproducible IMU ingestion. The HAR service validated ONNX inference on stored watch-device data and achieved 85.0% accuracy on the supported seven-activity subset. The live MetaWear validation stored 9 599 clean IMU rows and 463 HAR predictions during a 385-second run, with zero failed batches, zero retries, and zero dropped lines. Grafana validated dashboard observability from persisted InfluxDB data. Finally, the EEG/ECG extension validated that additional heterogeneous sensor families can be added through dedicated simulators, cleaners, topics, and tables.

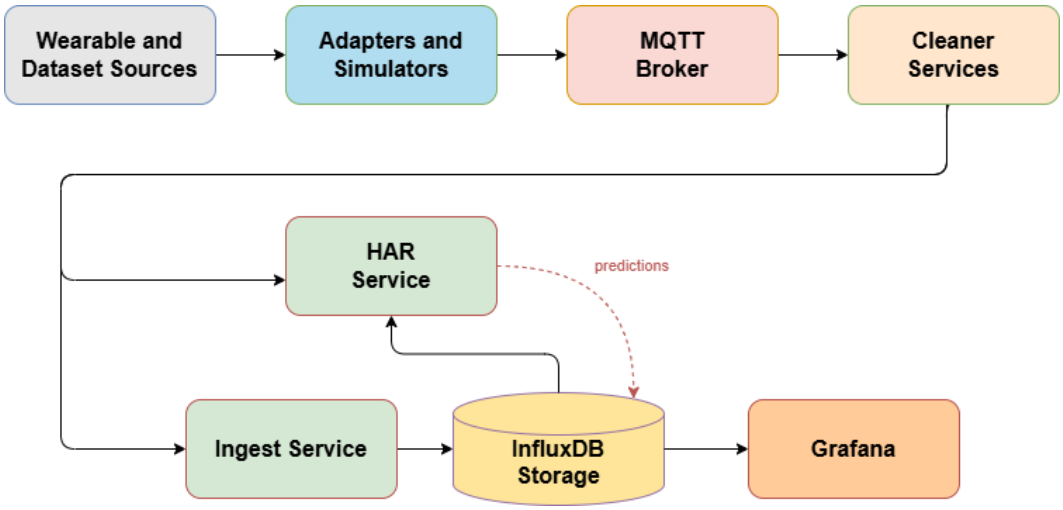

Figure 5.1: Final validated architecture pattern.

Figure 5.1 summarizes the validated platform as implemented in the repository. Wearable devices and replayed datasets first enter the system through adapters or simulators, which publish raw messages to the MQTT broker. Cleaner services then validate and normalize those messages. Clean sensor rows follow the main storage path through the ingest service, which is the component responsible for writing sensor tables to InfluxDB. Grafana reads from InfluxDB, so dashboard panels are based on persisted evidence rather than transient MQTT traffic.

The HAR service is separate from the ingest service. In live mode it consumes the clean watch stream produced by the cleaner path. In database-polling mode it reads stored IMU rows from InfluxDB for reproducible evaluation. In both modes, the HAR service owns prediction output and writes prediction rows back to InfluxDB. This is why the diagram separates sensor ingestion, prediction generation, storage, and visualization while still showing how they are connected.

5.5 Practical Significance

The practical value of the system lies in its explicit separation of concerns. A wearable adapter can be changed without rewriting the storage layer. A cleaner can be updated without changing Grafana. A new sensor family can be added without modifying the HAR service. Prediction rows can be inspected separately from input sensor rows. These boundaries make the system easier to debug, extend, and explain.

The second practical benefit is reproducibility. Dataset replay allows experiments to be repeated without requiring the MetaWear bracelet. Database polling mode allows the HAR service to evaluate stored rows deterministically. InfluxDB-backed Grafana panels keep visualization tied to persisted evidence rather than transient messages.

The third benefit is observability. The system exposes operational evidence: row counts, prediction counts, queue depth, failed writes, retry counts, dropped records, and dashboard panels. These signals make reliability claims more defensible than architecture diagrams alone.

5.6 Limitations

The limitations of the work are intentional and define the boundary of the contribution.

Table 5.2 summarizes the principal limitations and clarifies which claims are outside the validated scope of the platform.

Table 5.2: Limitations of the implemented platform.

Limitation	Explanation
No tennis-specific HAR model	The validated model supports a subset of SIDDHA activities and does not prove tennis stroke recognition.
No live ground-truth labeling	The real MetaWear validation checks pipeline reliability, not supervised live activity accuracy.
No EEG/ECG machine learning	EEG and ECG validate extensibility, storage, and visualization only.
No clinical interpretation	Physiological signals are not analyzed for medical conclusions.
No camera pipeline	Camera-based tracking is outside the implemented scope.
No production security	Authentication, authorization, secrets management, and hardening are not production-grade.
No edge deployment	The implementation is validated on a local Docker-based environment.

These limitations do not invalidate the architecture result. They clarify that the thesis contribution is the validated IoT data infrastructure, not a complete commercial sports-analytics product.

5.7 Future Work

Several directions can extend the platform.

5.7.1 Tennis-Specific HAR Model

The most direct future improvement is to train or fine-tune a model on tennis-specific movements such as forehand, backhand, serve, volley, and recovery steps. This would require a labeled dataset collected from the target wearable placement and sampling configuration. The existing architecture can support this work because the storage and replay paths already preserve ordered IMU rows and prediction outputs.

5.7.2 Live Ground-Truth Annotation

The live MetaWear pipeline currently records sensor data without ground-truth activity labels. A future annotation tool could allow a user or coach to mark activities during live sessions. Those labels could then be stored alongside sensor windows and used for supervised evaluation of real-world performance.

5.7.3 Physiological Signal Analytics

The EEG and ECG paths currently validate extensibility only. Future work could add signal-processing and machine-learning services for physiological features, provided the use case remains non-clinical unless appropriate medical validation is performed. Any such extension should keep physiological tables separate from IMU and HAR prediction tables.

5.7.4 Dashboard Export and Reproducibility

The live watch dashboard is provisioned in the repository. The EEG/ECG dashboard query set is documented, but a fully exported dashboard JSON would make that visualization more reproducible in a fresh clone. Future work should commit the exported EEG/ECG dashboard configuration and include screenshot or artifact evidence for validation runs.

5.7.5 Security and Deployment Hardening

The current platform is suitable for local thesis-scale validation. Production deployment would require secure MQTT configuration, service authentication, authorization, secrets management, network isolation, encrypted transport, backup policies, and monitoring alerts. These concerns were excluded from the scope of this thesis but are necessary for deployment outside a controlled research environment.

5.7.6 Edge and Mobile Deployment

Another direction is to move selected services closer to the sensing device. For example, cleaning or inference could run on an edge computer near the court. This would reduce network dependency and latency, but would also introduce constraints on compute resources, battery consumption, model size, and update management.

5.8 Final Statement

This thesis demonstrated that a modular IoT architecture can support real wearable acquisition, reproducible dataset replay, HAR inference, time-series storage, dashboard observability, and heterogeneous sensor extension within a single containerized platform. The strongest outcome is the validated infrastructure pattern shown in Figure 5.2.

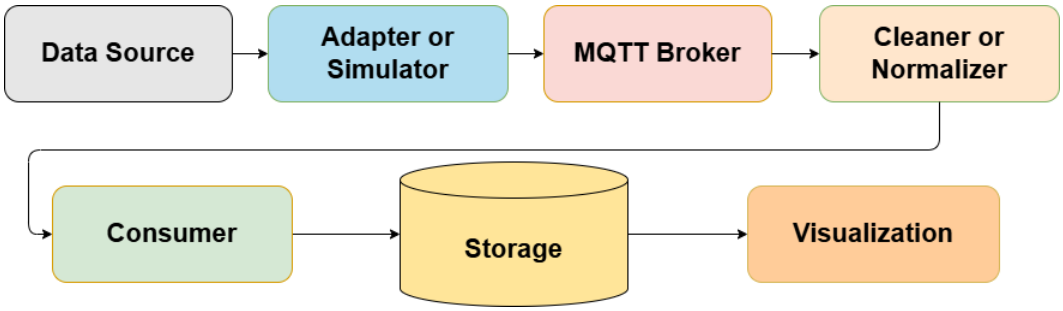

Figure 5.2: Core validated data-flow pattern of the thesis platform.

Figure 5.2 expresses the thesis contribution in its most compact form. Source-specific details are isolated at the adapter or simulator stage, MQTT decouples producers from consumers, the cleaner creates canonical records, and the final services either persist the data, run inference, or visualize the result. By keeping these responsibilities explicit and validating behavior through stored data and runtime metrics, the Wearable IoT Activity Pipeline platform provides a practical foundation for future sports and wearable-computing research.

Bibliography

[1] Oscar D. Lara and Miguel A. Labrador. «A Survey on Human Activity Recognition Using Wearable Sensors». In: *IEEE Communications Surveys & Tutorials* 15.3 (2013), pp. 1192–1209. DOI: 10.1109/SURV.2012.110112.00192.

[2] Luigi Atzori, Antonio Iera, and Giacomo Morabito. «The Internet of Things: A Survey». In: *Computer Networks* 54.15 (2010), pp. 2787–2805. DOI: 10.1016/j.comnet.2010.05.010.

[3] Jayavardhana Gubbi et al. «Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions». In: *Future Generation Computer Systems* 29.7 (2013), pp. 1645–1660. DOI: 10.1016/j.future.2013.01.010.

[4] OASIS. *MQTT Version 5.0*. 2019. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (visited on June 12, 2026).

[5] InfluxData. *InfluxDB 3 Documentation*. 2026. URL: https://docs.influxdata.com/influxdb3/ (visited on June 12, 2026).

[6] ONNX Community. *ONNX: Open Neural Network Exchange*. 2026. URL: https://onnx.ai/ (visited on June 12, 2026).

[7] Grafana Labs. *Grafana Documentation*. 2026. URL: https://grafana.com/docs/grafana/latest/ (visited on June 12, 2026).

[8] Andreas Bulling, Ulf Blanke, and Bernt Schiele. «A Tutorial on Human Activity Recognition Using Body-Worn Inertial Sensors». In: *ACM Computing Surveys* 46.3 (2014), 33:1–33:33. DOI: 10.1145/2499621.

[9] MbientLab. *MbientLab Documentation*. 2026. URL: https://docs.mbientlab.com/ (visited on June 12, 2026).

[10] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. 2nd ed. O'Reilly Media, 2021.

[11] Claus Pahl. «Containerization and the PaaS Cloud». In: *IEEE Cloud Computing* 2.3 (2015), pp. 24–31. DOI: 10.1109/MCC.2015.51.

[12] Docker Inc. *Docker Compose Documentation*. 2026. URL: https://docs.docker.com/compose/ (visited on June 12, 2026).

[13] FastAPI. *FastAPI Documentation*. 2026. URL: https://fastapi.tiangolo.com/ (visited on June 12, 2026).

[14] EMQ Technologies. *EMQX Documentation*. 2026. URL: https://docs.emqx.com/en/emqx/latest/ (visited on June 12, 2026).

[15] Martin Kleppmann. *Designing Data-Intensive Applications*. O'Reilly Media, 2017.

[16] Cindy Sridharan. *Distributed Systems Observability: A Guide to Building Robust Systems*. O'Reilly Media, 2018.

[17] Florenc Demrozi et al. «Human Activity Recognition Using Inertial, Physiological and Environmental Sensors: A Comprehensive Survey». In: *IEEE Access* 8 (2020), pp. 210816–210836. DOI: 10.1109/ACCESS.2020.3037715.

[18] Vittorio Fra et al. «Human Activity Recognition: Suitability of a Neuromorphic Approach for On-Edge AIoT Applications». In: *Neuromorphic Computing and Engineering* 2.1 (2022), p. 014006. DOI: 10.1088/2634-4386/ac4c38.

[19] Bluetooth SIG. *Bluetooth Specifications and Documents*. 2026. URL: https://www.bluetooth.com/specifications/specs/ (visited on June 12, 2026).

[20] MbientLab. *MetaMotionS Wearable Sensor*. 2026. URL: https://mbientlab.com/metamotions/ (visited on July 4, 2026).

[21] Alexandre Gramfort et al. «MEG and EEG Data Analysis with MNE-Python». In: *Frontiers in Neuroscience* 7 (2013), p. 267. DOI: 10.3389/fnins.2013.00267.

[22] Riccardo Pignari et al. «The Smart Inertial Device Data from Human Activities Dataset». In: *Scientific Data* 13 (2026), p. 864. DOI: 10.1038/s41597-026-06860-w.

[23] Apache Software Foundation. *Apache Parquet Documentation*. 2026. URL: https://parquet.apache.org/docs/ (visited on June 12, 2026).

[24] OpenNeuro. *OpenNeuro Dataset ds006848*. 2026. URL: https://openneuro.org/datasets/ds006848 (visited on June 12, 2026).

[25] Christopher J. Markiewicz et al. «The OpenNeuro Resource for Sharing of Neuroscience Data». In: *eLife* 10 (2021), e71774. DOI: 10.7554/eLife.71774.

[26] Krzysztof J. Gorgolewski et al. «The Brain Imaging Data Structure, a Format for Organizing and Describing Outputs of Neuroimaging Experiments». In: *Scientific Data* 3 (2016), p. 160044. DOI: 10.1038/sdata.2016.44.

[27] Cyril R. Pernet et al. «EEG-BIDS, an Extension to the Brain Imaging Data Structure for Electroencephalography». In: *Scientific Data* 6.1 (2019), p. 103. DOI: 10.1038/s41597-019-0104-8.