MIDDLE EAST TECHNICAL UNIVERSTIY

DEPARTMENT OF COMPUTER ENGINEERING

SUMMER PRACTICE REPORT CENG400

STUDENT NAME: Burak YILDIZ
ORGANIZATION NAME: ODTÜ-DTX (ODTÜ Dijital Dönüşüm ve İnovasyon Merkezi)
ADDRESS: BİLTİR Merkezi, Orta Doğu Teknik Üniversitesi, Dumlupınar Bulvarı 1, 06800 Çankaya/Ankara
START DATE: 01.09.2025
END DATE: 26.09.2025
TOTAL WORKING DAYS: 20

STUDENT'S SIGNATURE

ORGANIZATION APPROVAL

Contents

Chapter 1

Introduction

This report summarizes the summer practice I completed at ODTÜ-DTX (Dijital Dönüşüm ve İnovasyon Merkezi) between 01.09.2025 and 26.09.2025, under the supervision of Doç. Dr. Şeyda Ertekin. The practice was focused on improving my graduation project, a **Real-Time Location System (RTLS)** using BLE signals, ESP32 devices, and machine learning.

The main objective was to advance the accuracy and robustness of indoor positioning systems by combining embedded hardware, cloud data pipelines, and advanced ML models. During this practice, I:

- Programmed ESP32 boards to collect BLE RSSI data via ESP-NOW and uploaded readings to Firebase.
- Applied trilateration with path-loss models and compared its performance against ML-based approaches.
- Trained multiple ML models (kNN, Random Forest, SVM, XGBoost, Ensembles) on collected datasets.
- Performed feature engineering (differences, ratios, statistics) and applied Kalman filtering for noise reduction.
- Developed a frontend page for visualizing real-time ML-based predictions in the project's web application.
- Delivered periodic reports, research summaries, and evaluations to my academic supervisor.

The following chapters explain the project phases, organizational environment, and my personal reflections.

Chapter 2

Information about the Project

2.1 Analysis Phase

The project started with identifying the main challenges of indoor RTLS:

- **Noisy RSSI signals:** BLE RSSI suffers from multipath fading and environmental interference.
- **Accuracy requirement:** Achieving sub-1 meter accuracy is difficult using RSSI alone.
- **Scalability:** The system must support many tags and scanners in real-time.

We explored multiple approaches: classical trilateration with path-loss models, fingerprinting with ML classifiers, and data smoothing via filters (Kalman). A data collection plan was prepared for two testbeds:

- **Testbed A:** A 3×4 grid in a controlled lab environment, 1m $\times$ 1m cells, 60 samples per cell (total 720 samples).
- **Testbed B:** A 9×12 grid in a factory environment, 0.5m $\times$ 0.5m cells, 30 samples per cell (total 3240 samples).

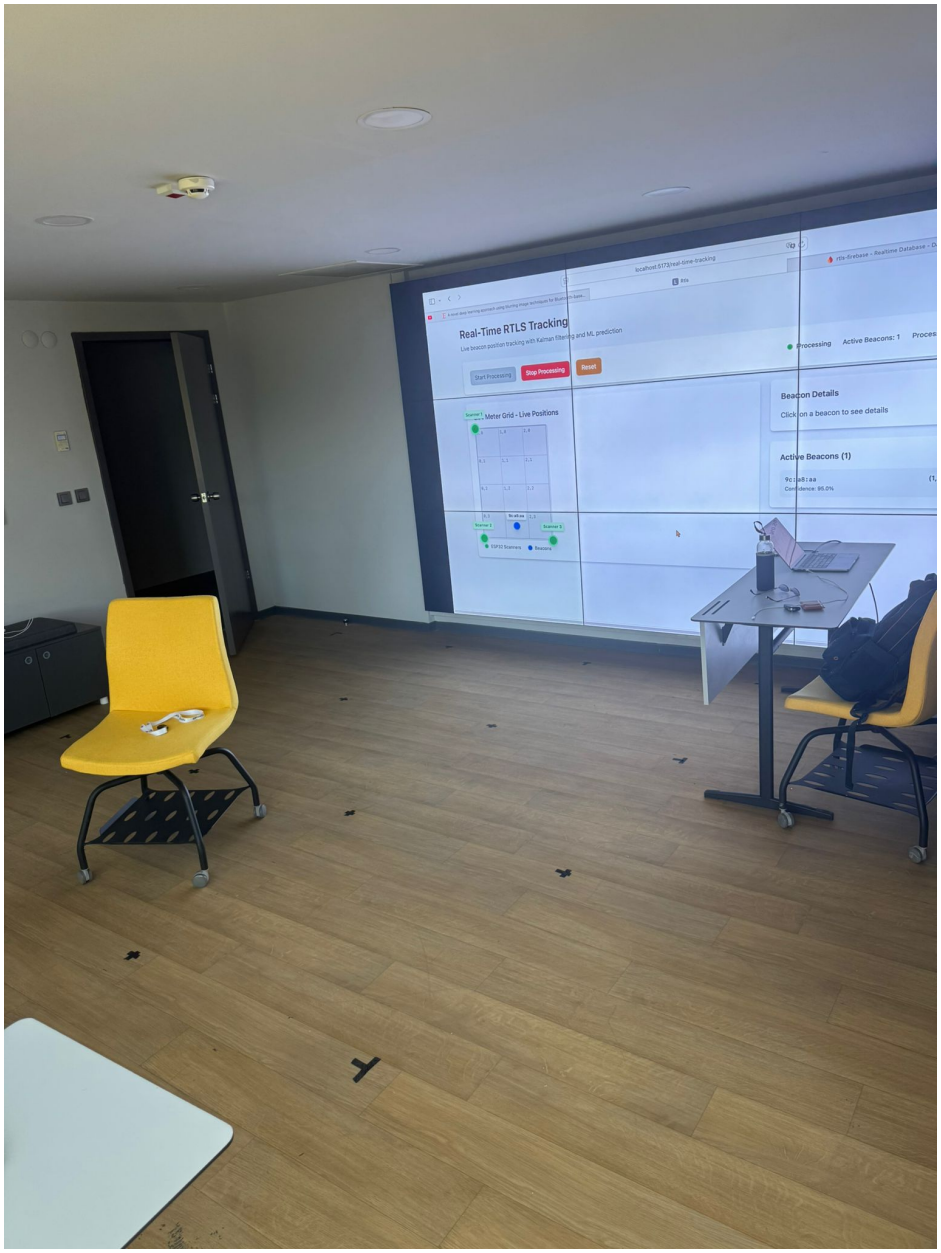

Figure 2.1: Testbed A layout and grid labeling (cell indices and anchor placement).

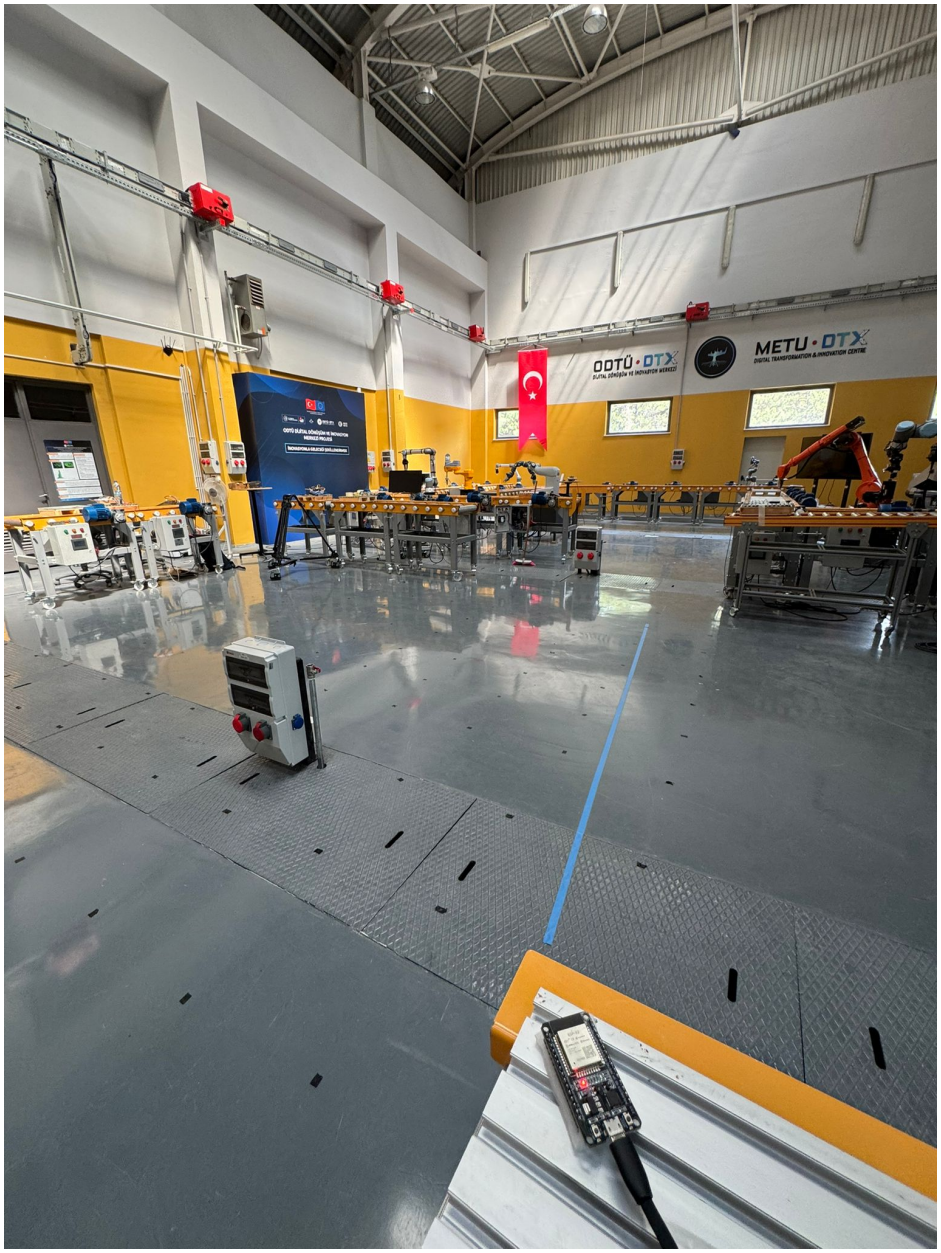

Figure 2.2: Testbed B layout (factory floor) with scanner (ESP32) anchor positions and walk path.

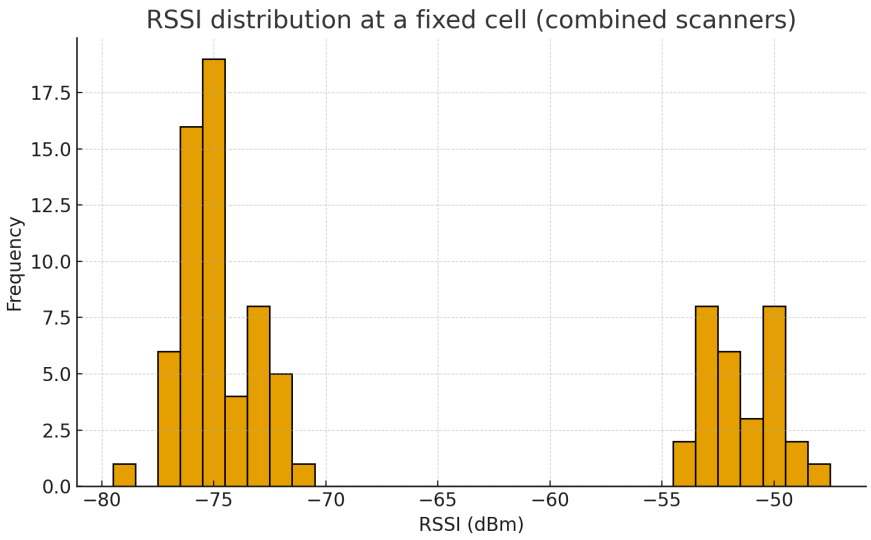

Figure 2.3: Typical RSSI distribution at a fixed cell showing variance and multipath-induced tails (motivation for filtering).

2.2 Design Phase

The design phase focused on the end-to-end pipeline:

- **Hardware:** 3 ESP32 scanners placed at key anchor points, configured with ESP-NOW for low-latency data collection.
- **Database:** Firebase Realtime Database as the central storage layer for RSSI readings.
- **Algorithms:** Both trilateration and ML models designed to process RSSI data.
- **Frontend:** Integration of a dashboard into the existing React-based webapp to visualize predictions and errors in real time.

Architectural diagrams were created to ensure modularity: hardware (data collection), software (data processing), ML (training + inference), and UI (visualization).

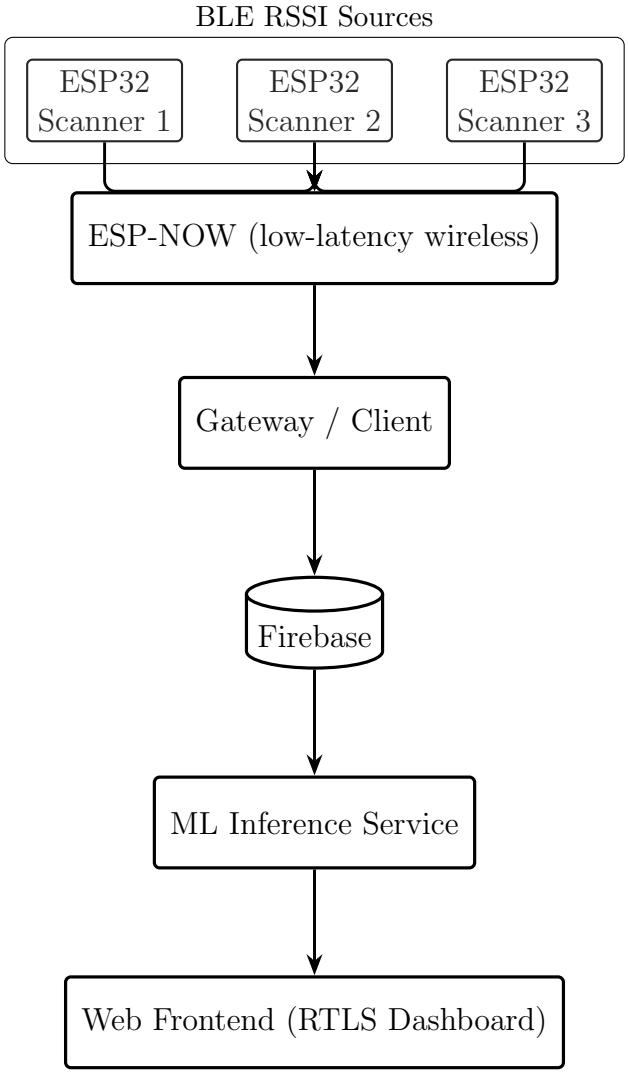

Figure 2.4: System architecture: ESP32 scanners → ESP-NOW → Gateway/Client → Firebase → ML inference service → Web frontend.

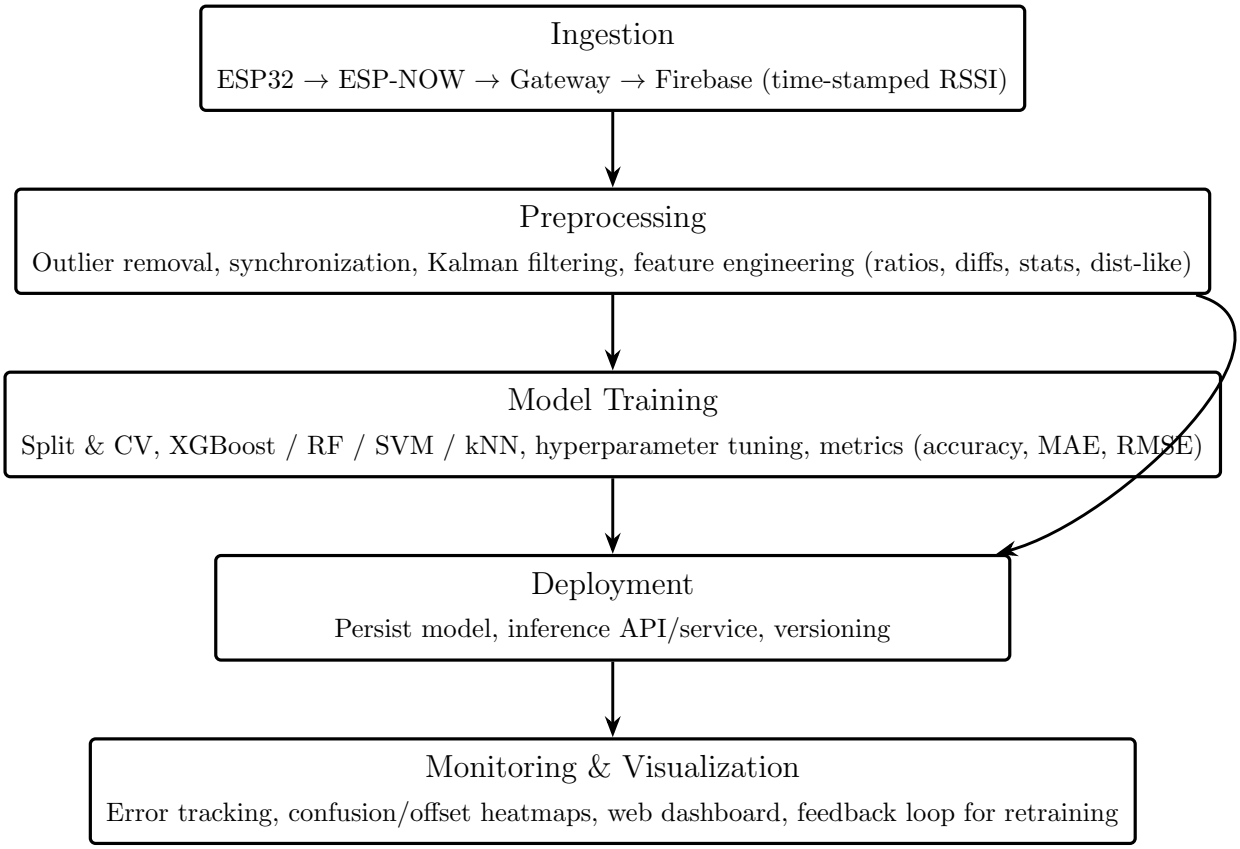

Figure 2.5: Data pipeline and processing stages: ingestion, preprocessing (filtering & feature engineering), model training, deployment, and monitoring.

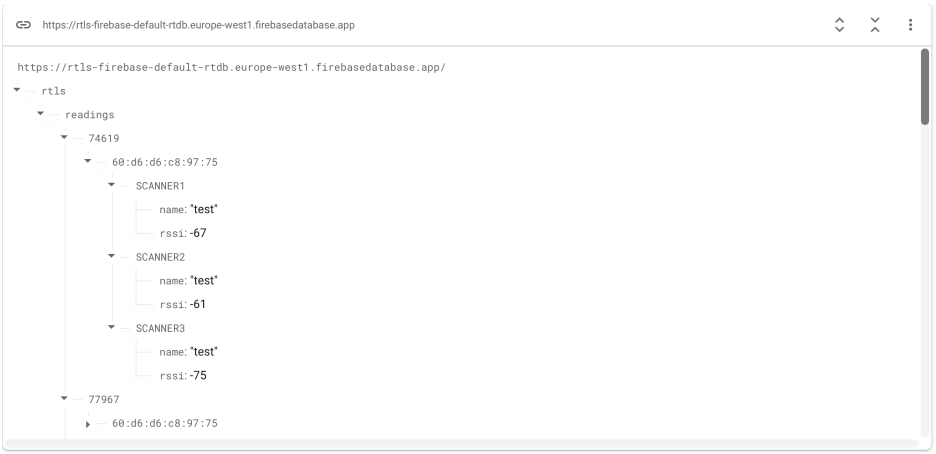

Figure 2.6: Firebase Realtime Database structure for RSSI samples, device metadata, and predictions.

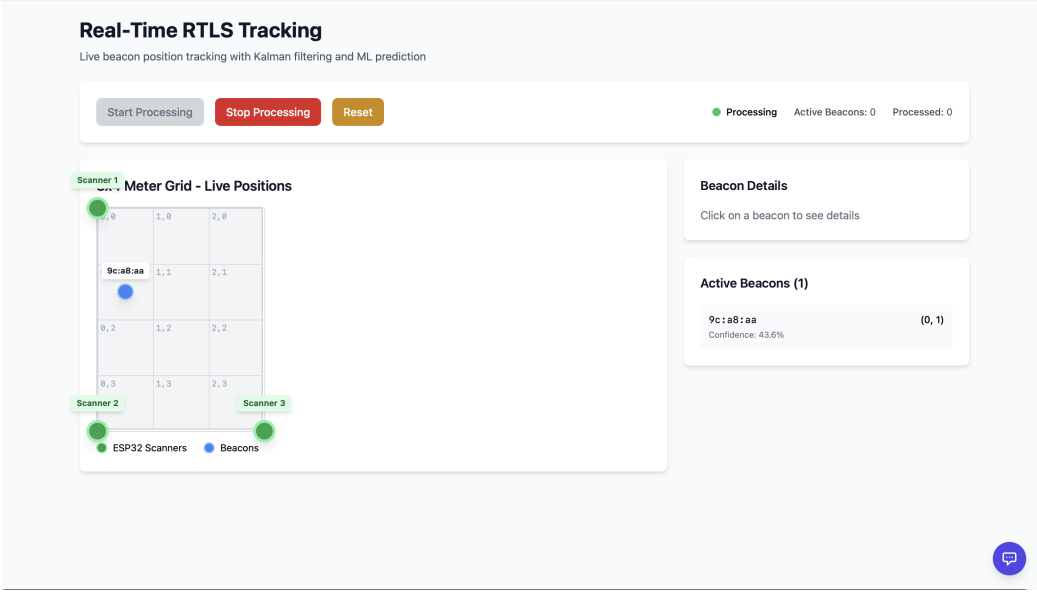

Figure 2.7: Frontend dashboard wireframe for real-time position visualization, confidence metrics, and error overlays.

2.3 Implementation Phase

The main implementation tasks were:

- **ESP32 Programming:** Configured ESP32 WROOM-32 boards to scan BLE beacons, filter known MAC addresses, and transmit results via ESP-NOW.
- **Firebase Integration:** Implemented code to stream RSSI readings into structured JSON format.
- **ML Modeling:** Created pipelines for kNN, RF, SVM, and XGB, with feature engineering (ratios, differences, statistics, dist-like transforms).
- **Kalman Filtering:** Implemented an in-pipeline Kalman filter to reduce variance while preserving signal trends.
- **Frontend Extension:** Added an RTLS visualization panel to the web dashboard, showing real-time position predictions and error heatmaps.

```
38  String targetName = "test";
39
40  /*************** WIRE PROTOCOL ***************/
41  // 'C' (gateway->remote): MsgCycle { type, cycleId }
42  // 'R' (remote->gateway): MsgReading { type, mac6, int16_t rssi, cycleId, uint8_t scannerId }  // MAC targets
43  // 'T' (remote->gateway): MsgReading { type='T', same layout }                                  // NEW name target
44  // 'D' (remote->gateway): MsgDone { type, cycleId }
45
46  #pragma pack(push, 1)
47  struct MsgCycle  { uint8_t type; uint32_t cycleId; };
48  struct MsgReading{
49    uint8_t type;      // 'R' or 'T'
50    uint8_t mac[6];
51    int16_t rssi;
52    uint32_t cycleId;
53    uint8_t scannerId; // 2/3 (remote id); may be 0 if older firmware
54  };
55  struct MsgDone   { uint8_t type; uint32_t cycleId; };
56  #pragma pack(pop)
57
58  /*************** DATA MODEL ***************/
59  struct RemoteInfo {
60    uint8_t mac[6];
61    const char* label;
62    uint8_t scannerId; // 2 or 3
63  };
64  RemoteInfo remotes[2] = {
65    { {0}, LAB_REM1, 2 },
66    { {0}, LAB_REM2, 3 }
67  };
68
```

Figure 2.8: ESP32 firmware (ESP-NOW + BLE scan) code excerpt showing packet format and filtering logic.

```python
# Buffers
# readings_buffer: { ts: { mac: { scanner_id: {"name": str, "rssi": int} } } }
readings_buffer = {}
first_seen = {}              # when ts was first seen
last_debug = {}              # last time we printed waiting status for ts
completed_timestamps = set() # timestamps we've uploaded

def ts_completion_status(ts_dict):
    """
    Return (is_complete, complete_macs, missing_info)
      - is_complete: True if at least one MAC named 'test' has all REQUIRED_SCANNERS
      - complete_macs: list of macs that are complete
      - missing_info: {mac: [missing_scanners]} for those not complete
    """
    complete_macs = []
    missing = {}
    for mac, per_scanner in ts_dict.items():
        present = set(per_scanner.keys())
        needed = REQUIRED_SCANNERS - present
        if not needed:
            complete_macs.append(mac)
        else:
            missing[mac] = sorted(list(needed))
    return (len(complete_macs) > 0, complete_macs, missing)

def upload_timestamp(ts_key, ts_dict, macs_to_upload):
    """
    Upload only the MACs that are complete (name=='test' by construction).
    Schema:
      /rtls/readings/<ts>/<mac>/<SCANNERX> -> { rssi, name }
    """
```

Figure 2.9: Data ingestion snippet: pushing timestamped RSSI tuples to Firebase under session and cell keys.

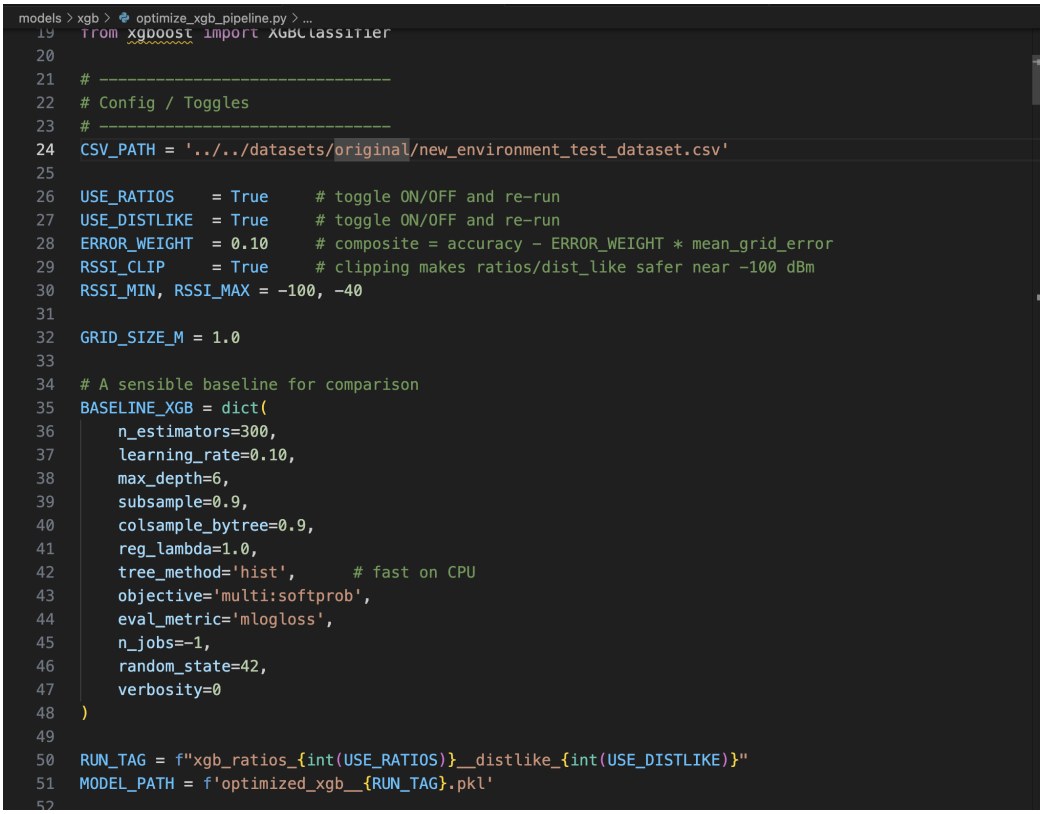

Figure 2.10: ML training pipeline script: feature generation (ratios/diffs/stats), train/validation split, model registry.

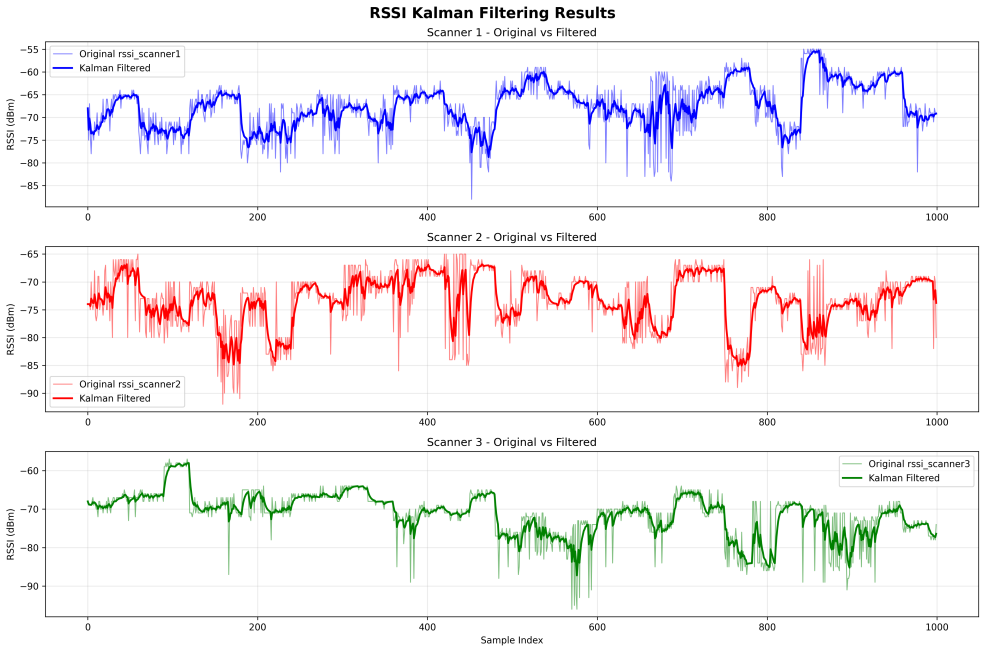

Figure 2.11: Kalman filtering concept applied to RSSI streams: measurement vs. filtered estimate across time.

2.4 Testing Phase

Testing included:

- **Unit Testing:** Verified ESP32 firmware modules (BLE scanning, ESP-NOW transmission).
- **Dataset Testing:** Split datasets into 80/20 train-test for fair evaluation.
- **Cross-validation:** Used 5-fold CV for hyperparameter tuning of ML models.
- **Performance Metrics:** Accuracy, mean grid error, mean absolute error (MAE), and RMSE were tracked.
- **Visualization:** Confusion matrices and offset heatmaps used to diagnose spatial error distributions.

Results showed that ML-based fingerprinting with XGB and Random Forest achieved much higher accuracy than raw trilateration, especially when combined with feature engineering and Kalman filtering.

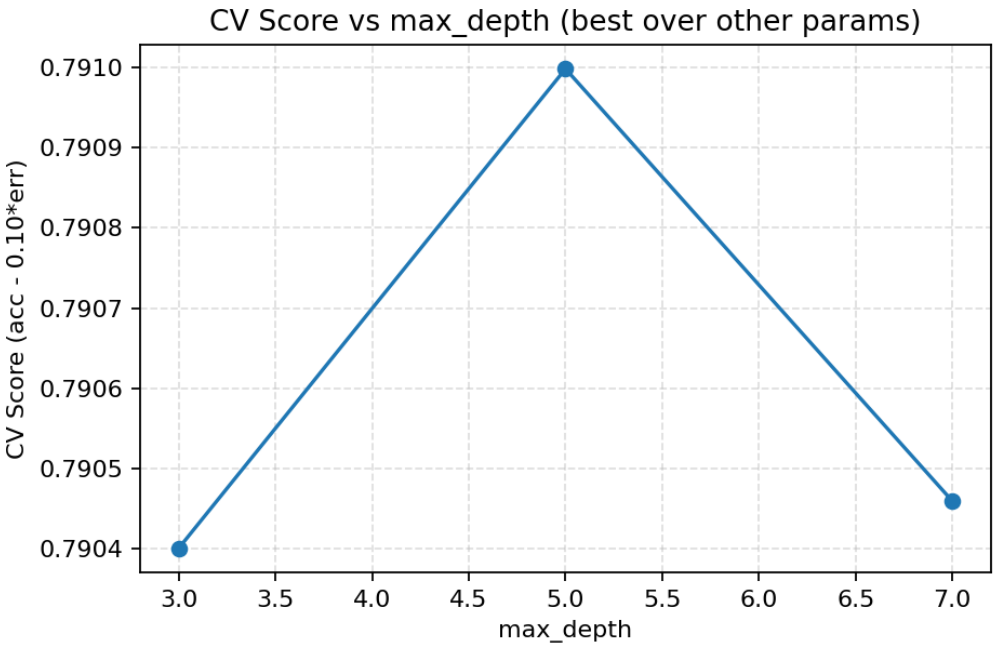

Figure 2.12: Cross-validation curve for XGBoost (max_depth) and validation accuracy trends.

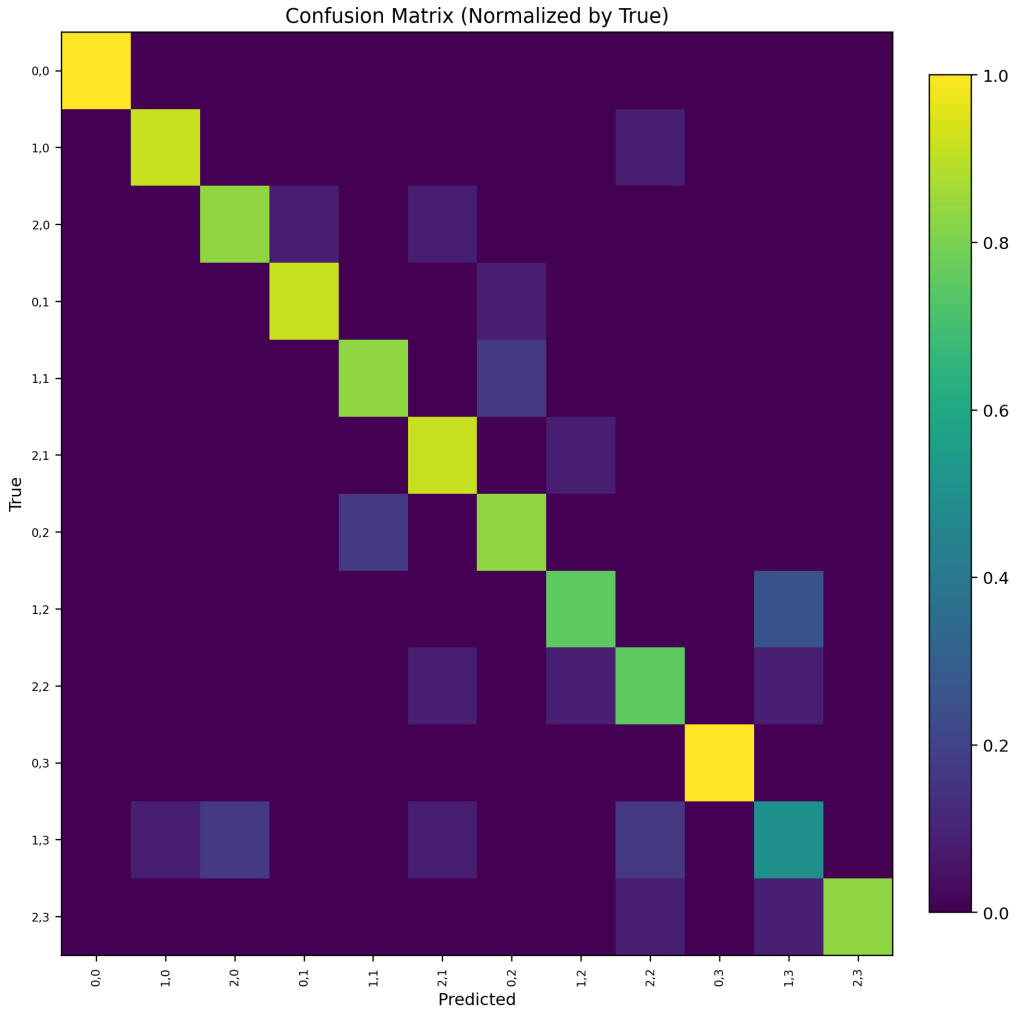

Figure 2.13: Confusion matrix over grid cells for the best model; darker diagonal indicates correct localizations.

Chapter 3

Organization

3.1 Organization and Structure

ODTÜ-DTX (Dijital Dönüşüm ve İnovasyon Merkezi) is a research and innovation center at Middle East Technical University, focused on accelerating digital transformation in various domains. It operates in collaboration with faculty members and industry partners, providing an environment where academic research meets practical applications.

The RTLS project was part of the center's IoT and Industry 4.0 initiatives, supervised by Doç. Dr. Şeyda Ertekin and technically supported by DTX staff. The team structure included:

- **Supervisor:** Guidance on ML methodology, evaluation metrics, and research direction.
- **Other researchers:** Support on embedded systems and data pipelines.
- **Student Researcher (myself):** Responsible for implementation, experimentation, and reporting.

3.2 Methodologies and Strategies Used in the Organization

The organization follows a research-oriented but structured methodology:

- **Agile-inspired workflow:** Weekly meetings to review progress and adapt research direction.
- **Experiment-driven development:** Hypothesize, implement, test, and compare ML and filtering approaches.

- **Continuous reporting:** Delivered reports and experimental results to the supervisor for feedback.
- **Toolset:** Python (scikit-learn, XGBoost, Firebase SDK), ESP-IDF/Arduino IDE for embedded programming, Firebase for cloud storage, React + TypeScript for frontend.

This methodology balanced research freedom with structured deliverables.

Chapter 4

Conclusion

The summer practice at ODTÜ-DTX allowed me to deepen my expertise in embedded systems, IoT data pipelines, and ML for real-time localization. Key achievements include:

- Successfully built a complete RTLS pipeline: ESP32 scanners → Firebase → ML pipeline → frontend visualization.
- Demonstrated that ML fingerprinting outperforms trilateration in noisy indoor environments.
- Validated that Kalman filtering can reduce RSSI noise and improve prediction stability.
- Delivered reproducible datasets, trained models, and visualization dashboards.

This practice provided me with hands-on experience in both hardware and ML, preparing me for future research and industrial projects in IoT and smart factory domains.