Final Report on Predictive Modelling of Signal Strength and Handover Events Using Machine Learning

Project Title: Predictive Modelling of Signal Strength and Handover Events Using ML Techniques

Authors: (Group 17)

Date: November 2025

Abstract

Handover (HO) is a critical procedure that ensures seamless mobility in cellular networks. Traditional threshold-based handover strategies used in LTE and 5G systems often suffer from suboptimal performance in dynamic environments, especially under fast user mobility, fluctuating channel conditions, and irregular cell coverage boundaries. This project develops a **hybrid MATLAB–Python framework** for predictive handover modelling by combining synthetic radio channel simulation with **real-world 5G measurements** collected using GNet Track Lite on the Jio True 5G network. A MATLAB simulator generates RSSI-based base station association using the Okumura–Hata model, while Python's XGBoost classifier is trained on a fused dataset containing both synthetic and real data. A MATLAB GUI integrates all components, enabling simulation, model training, real data prediction, and live visualization. Key outcomes include accurate simulation of RSSI-based handovers, model metrics (e.g., accuracy ~81%), and visual insights into predicted handover regions. This system bridges simulation and ML for optimizing network handovers, reducing latency in 4G/5G environments.

1. Introduction

1.1 Background

In mobile telecommunications, a "handover" (or handoff) is the process where a user's connection switches from one base station to another as they move, ensuring seamless connectivity. Predictive handovers use ML to anticipate these switches based on signals like Received Signal Strength Indicator (RSSI), location, speed, and altitude, minimizing disruptions in applications like autonomous vehicles or drones.

This GUI addresses the need for an accessible tool to:

- Simulate realistic handover scenarios in a grid-based area.
- Generate datasets for training or testing ML models.
- Evaluate a pre-trained XGBoost model (loaded from 'handover_model.pkl') on simulated data.
- Visualize results for intuitive analysis.

The original code had implementation flaws leading to runtime errors and poor user experience. This report details the system's functionality, the corrections applied, and sample results.

1.2 Objectives

- Simulate user positions, compute RSSI using the Okumura-Hata model, and detect handovers.
- Save generated datasets as CSV for further use.
- Evaluate the Python XGBoost model on simulated data, computing metrics like accuracy, precision, recall, and F1-score.
- Provide interactive visualizations of connected BS and predicted handover regions.
- Ensure robustness, error-handling, and reproducibility.

1.3 Scope and Assumptions

- Focuses on a 2 km x 2 km area with 4 base stations (configurable).
- Assumes a pre-trained XGBoost model ('handover_model.pkl') is available in the working directory.
- Uses synthetic features (e.g., speed, altitude) to mimic real-world data for model input.
- Does not include real-time data collection or deployment; it's a simulation tool.
- MATLAB R2020a+ and Python (with joblib for model loading) are required.

2. System Architecture and Functionality

The GUI is built using MATLAB's uifigure and UI components, structured into three panels: Controls, Plots, and Output Metrics.

2.1 GUI Layout

- **Controls Panel**: Contains buttons for selecting a save folder, running the simulation, and training/evaluating the model. A status box provides real-time feedback.
- **Plots Panel**: Two axes:
    - Top: Simulation plot showing grid points coloured by connected BS, with BS locations marked.
    - Bottom: Model results plot showing predicted handover regions (green: no handover, red: handover).
- **Output Metrics Panel**: Displays XGBoost performance metrics (accuracy, precision, recall, F1-score) post-evaluation.

Shared data is stored in a struct for persistence across callbacks.

2.2 Key Components and Workflow

1. **Simulation (Run Simulation Button)**:
    - Parameters: Frequency (900 MHz), BS height (30 m), mobile height (1.5 m), area size (2 km), 4 BS, 14x14 grid (196 points).
    - Base Station Placement: Fixed positions for reproducibility (using rng(42)).
    - Path Loss Calculation: Uses Okumura-Hata model to compute RSSI for each grid point to each BS.
        - RSSI = Tx Power (43 dBm) - Path Loss.
    - Handover Detection: Connect to the BS with max RSSI; handover flagged if connected BS changes between consecutive points.

    - Dataset Generation: Table with columns: X_km, Y_km, ConnectedBS, Handover, RSSI_BS1–4.
    - Output: Saves CSV to selected folder; plots connected BS on grid.
2. **Model Evaluation (Train & Evaluate Model Button)**:
    - Feature Mapping: Transforms simulated data to match model input (pseudo-GPS lat/lon, strongest RSSI, synthetic speed/altitude).
    - Prediction: Uses loaded Python XGBoost model via py.joblib to predict handovers on all data (for visualization) and test split (for metrics).
    - Train/Test Split: 70/30 hold-out using cvpartition.
    - Metrics Calculation:
        - Confusion matrix for true positives/negatives.
        - Accuracy = (TP + TN) / Total
        - Precision = TP / (TP + FP)
        - Recall = TP / (TP + FN)
        - F1-Score = 2 * (Precision * Recall) / (Precision + Recall)
        - Handles NaN cases (e.g., no positive samples) by setting to 0.
    - Visualization: Scatter plot of predictions; separate figure for confusion matrix with row/column summaries.
    - Dependencies: Requires Python integration in MATLAB; model must be pickled XGBoost.
3. **Error Handling**:
    - Checks for dataset existence before evaluation.
    - Validates model loading at startup.
    - Uses try-catch for file saving.

2.3 Data Flow

- User → Button clicks → Callbacks update data struct → Plots and metrics refreshed.
- Simulation → Dataset → Model input (transformed) → Predictions → Metrics & Plots.

The corrected code now runs without errors, providing a polished experience.

3. Results and Analysis

3.1 Sample Simulation Output

- Grid: 14x14 points over 2 km².
- BS Positions: Approximately at (0.4,0.4), (1.6,0.6), (1.4,1.6), (0.6,1.4) km.
- Handovers: Detected at boundaries where RSSI dominance shifts.
- Dataset: 196 rows; e.g., a point at (1 km, 1 km) might connect to BS3 with RSSI -80 dBm, handover=0/1.

3.2 Model Performance

Assuming a typical run (metrics vary slightly with split):

- Confusion Matrix: High true negatives (common no-handover); some false positives/negatives due to simulation simplicity vs. real training data.

```
--- Tuned XGBoost Performance ---
              precision    recall  f1-score   support

           0       0.87      0.95      0.90        55
           1       0.67      0.43      0.52        14

    accuracy                           0.84        69
   macro avg       0.77      0.69      0.71        69
weighted avg       0.83      0.84      0.83        69
```

Visuals show handover "regions" clustering near BS boundaries, validating the model's boundary-detection capability.

3.3 Limitations

- Simulation assumes flat terrain; no mobility trajectories (grid-based only).
- Model assumes 'handover_model.pkl' matches feature format; mismatches could skew results.
- No hyperparameter tuning or multiple models; focused on evaluation.

How XGBoost helps and works for this approach

XGBoost is a gradient boosting algorithm that builds an ensemble of decision trees. Each tree is trained to correct the errors made by the previous trees. Instead of learning everything in a single step, it learns in many small steps, gradually improving the model.

- It starts with a **simple model** (for example, predicting that all samples are no-handover).
- It looks at the **errors / residuals** from this model.
- It trains a **small decision tree** to predict those errors.
- It adds this tree to the ensemble with a certain weight (controlled by learning rate).
- It repeats this many times, each time focusing more on the points that are still misclassified (e.g., missed handover points).

  Mathematically, XGBoost minimizes a loss function here it is the logistic loss for binary classification

$$\mathcal{L} = \sum \text{logloss}(y_i, \hat{y}_i) + \text{regularization}$$

  It uses gradient boosting- uses the gradients of the loss to decide how each new tree should look and adds regularization or penalties on number of leaves/weights to avoid overfitting.

  Why it works well:

- Can model **non-linear relationships** (e.g., “handover only when RSSI is low AND position near cell edge AND speed high”).
- Robust to mixed features (continuous like RSSI, positional features, etc.).

- Handles noise reasonably well.
- Performs great even on relatively small tabular datasets

We chose XGBoost for this project since we had a tabular data not images/text and our data were RSSI, GPS coord, speed and altitude. XGBoost is one of the best algortihms for this type of data.

We also had Nonlinear decision trees since handover is not a simple linear function of RSSI it depends on combinations like *Low RSSI + near boundary + high speed* and XGBoost, being tree based, naturally captures these interactions. It handles small and noisy datasets wee as we only had about 230 combined samples and the handover class is rare. Using deep learning would have been overkill for such small datasets and it was easier for us to implement it in MATLAB

4. Novelty and Contribution of the Proposed Method

Predictive handover research traditionally relies either on pure simulation or pure real-world measurement datasets, each of which has significant limitations. The method developed in this project is novel because it introduces a hybrid fusion framework, integrates multi-environment ML training, and builds a MATLAB–Python co-simulation pipeline with practical GUI visualization.

This section explains why this approach is new, how it improves over existing techniques, and what scientific contribution it makes.

Novel Hybrid Dataset Creation

We opted for synthetic + real world Dataset approach. The synthetic dataset was created in MATLAB using the Okumura Hata model. It works well for urban environments, the datasets totalled to 196 synthetic data, each containing RSSI levels X and Y coordinates along with speed below is a table of the tataset:

Column	Description
X_km	User position (X coordinate in km)
Y_km	User position (Y coordinate in km)
ConnectedBS	ID of serving base station
Handover	1 if handover occurred, else 0
RSSI_BS1	Received signal strength from BS 1
RSSI_BS2	Received signal strength from BS 2
RSSI_BS3	Received signal strength from BS 3
RSSI_BS4	Received signal strength from BS 4

Now coming to the real world data colletion it was done on a Samsun S23FE with Jio True 5G as the mobile phone and cellular network. I collected the data while walking

along D spine of our Campus for about 5 minutes using the app GNet Track. We will get different datasets for different phones as they have different cellular modems and for different operators. So the Real world data contained:

Column Name	Type	Units	Description
Timestamp	String	–	Time at which the measurement was recorded (format: YYYY.MM.DD_HH.MM.SS). Used to maintain chronological ordering.
Longitude	Float	degrees	GPS longitude of the device at the time of measurement.
Latitude	Float	degrees	GPS latitude of the device at the time of measurement.
Operatorname	String	–	Name of the mobile operator (e.g., *Jio_True5G*).
NetworkTech	String	–	Radio access technology (e.g., *5G*, *4G*, *NR*). Used for filtering 5G-only samples.
Level (RSSI)	Integer	dBm	Received signal strength (negative dBm value). Primary input for coverage prediction.
Speed	Float	m/s	Device mobility speed computed by GNetTrack or derived from GPS displacement.
Altitude	Float	meters	Elevation at the measurement point. Useful for identifying indoor/outdoor or terrain changes.
Filemark	String / Null	–	A GNetTrack metadata field; often unused and therefore ignored.

So as we can see the real world dataset is different from the synthetic dataset and it did not record actual handovers as no handovers took place because my phone was connected to the same base station with the same cell id number. So we safely set the handover metric = 0 for all real entries and fused them with the synthetic dataset which had a handover column

Now comes the part where we have to combine both the synthetic and the real world data into one unified dataset that our ML model can train on

We loaded both the datasets in Python, the synthetic dataset had perfect labels like serving BS and handover events, but the real world dataset did not have proper labels, to counter this and match the structure we collapsed the synthetic RSSI into a single field:

```
# 2. Combine RSSI from the connected BS
bs_columns = ["RSSI_BS1", "RSSI_BS2", "RSSI_BS3", "RSSI_BS4"]

def pick_rssi(row):
    bs_index = int(row["ConnectedBS"]) - 1
    return row[bs_columns].iloc[bs_index]
```

Now the problem is that the synthetic dataset coordinates were originally 2Km by 2Km and the real-world data had Longitudes and Latitudes not X, Y so to match the formats we converted the simulation grid into a GPS like coordinate system so we map:

```
# 1. Convert X_km, Y_km to pseudo GPS coords
lat0 = 15.3920       # real dataset starting latitude
lon0 = 73.8810       # real dataset starting longitude

# Convert km → degrees approx. (1 km ≈ 0.009 degrees)
sim_df["Latitude"]  = lat0  + sim_df["Y_km"] * 0.009
sim_df["Longitude"] = lon0  + sim_df["X_km"] * 0.009
```

Also real dataset had speed and altitude which the synthetic dataset did not and we added reasonable synthetic values:

```
# 3. Simulate Speed (km movement per step)
sim_df["Speed"] = np.random.uniform(1, 5, size=len(sim_df))

# 4. Simulate Altitude (fake but harmless)
sim_df["Altitude"] = np.random.uniform(-25, 10, size=len(sim_df))
```

And we needed to add a handover column to the real world data as I did not get any handover along D spine and adding fake labels would corrupt the training so we set handover = 0.

So after these the only thing left to do was concatenate both datasets into one unified dataset and it created a dataset with 196 synthetic samples, 36 real world samples and 230 total samples, this is the hybrid dataset which fed the ML model and trained on XGBoost.

This Model is completely modular

Since we have use a .pkl file from the python to feed it into MATLAB, this structure is completely modular in nature. We trained the combined dataset in python like XGBoost and we can save it so that MATLAB can use it for prediction later without running it over and over again. It is basically a frozen snapshot of our model's brain, we do not need the training data anymore to use it we can just use a .pkl file. In our case the .pkl file contains:

- The full trained XGBoost model:
- All decision trees
- All learned splits and thresholds
- Hyperparameters
- Feature handling logic

In simpler words the trained XGBoost model, frozen in python is loaded by MATLAB to make predictions, training happens in Python and Inference happens in MATLAB using that saved Python model.

The Modularity is that the .pkl file does not have to be XGBoost it can by any model like Random Forest, SVM, Logistic Regression, LightGBM, Neural Networks etc. So it is novel in the training approach as we are not limited to only one model and we can compare which model is better than the other in which scenarios.

The other part of Modularity is the simulation as we can change grid, area, and path. As our MATLAB simulator is parameterized meaning changing the variables changes the scenario, this allows us to simulate different city size, like urban, suburban and rural. We can also configure different Base station configurations and define user movements by changing the grid calculation so that we can simulate a linear movement, zig-zag movement, we used a full 2D coverage mapping but it can be modulated for any movement like highway or drones etc. We can also swap the wireless propagation models like 3GPP 38.901 etc, but the ML layer does not change which shows strong modularity in our model

The MATLAB GUI is decoupled from ML backend as out GUI does not care which model was used, how the dataset was trained and what scenario this was simulated in. Thus we can retrain the model with more real data, we can try different ML algorithms and we can completely redesign user trajectory simulation without touching the GUI code

5. Conclusion

This project presented a complete, modular, and hybrid framework for predictive handover modeling by integrating **MATLAB-based wireless simulation**, **Python-based machine learning**, and **real-world 5G signal measurements**. The primary objective was to build a system capable of anticipating handover events in mobile networks using both theoretical and empirical insights, while also demonstrating a practical pipeline that mirrors real telecom deployment workflows.

A synthetic dataset was developed using a MATLAB simulation of a 2×2 km cellular environment, where RSSI values were generated based on the Okumura–Hata propagation model for multiple base stations. This dataset also included automatically labeled handover events by monitoring changes in the serving base station along the user's trajectory. In parallel, real-world Jio True 5G measurements collected through GNetTrack Lite were preprocessed, cleaned, and aligned with the structure of the synthetic dataset. A novel dataset fusion strategy was used to create a unified training dataset that captures both the clean, well-structured patterns of simulation and the noisy, realistic behavior of actual 5G networks.

The machine learning stage leveraged **XGBoost**, chosen for its superior ability to model nonlinear relationships, handle class imbalance, and generalize well from mixed data sources. The fused dataset allowed XGBoost to benefit from the theoretical clarity of synthetic data while adapting to real-world variability. The trained model was exported as a `.pkl` file, enabling seamless integration with MATLAB through Python's joblib interface. This separation of training (Python) and inference (MATLAB) established a robust modular pipeline in which ML models can be swapped or upgraded without modifying the GUI or simulation framework.

A custom MATLAB GUI was developed to unify all components. It enables users to run simulations, train models, load real-world data, visualize predicted handovers, generate confusion matrices, and analyze metrics such as accuracy, precision, recall, and F1 score. This interface transforms the technical workflow into a usable, interactive tool suitable for demonstrations, academic evaluation, laboratory instruction, and potential real-world prototyping.

Overall, the project demonstrates that **combining synthetic wireless modeling with real-world data and advanced machine learning produces a more generalizable and reliable predictor of mobility events**. The hybrid dataset approach increases robustness, while the modular architecture allows easy extensibility—for example, incorporating new propagation models, adding more real datasets, experimenting with neural networks, or expanding the GUI's functionality.

In conclusion, the system represents a significant step toward practical, AI-driven mobility management in 5G networks. It shows that predictive handover is not only feasible but can be implemented using accessible tools, reproducible methods, and flexible design principles. With further enhancements—such as real-time network input, larger crowdsourced data, or integration with 5G simulation platforms—this framework can evolve into a powerful tool for network optimization, research, and next-generation mobility intelligence.