CHAPTER FOUR

System Implementation and Result Analysis

4.1 System Implementation Overview

The full implementation of the proposed energy forecasting framework was carried out in a modular simulation environment, which integrated multiple software tools and components to emulate real-world IoT-enabled energy monitoring conditions. The system was designed to simulate a realistic pipeline—from data generation through transmission, aggregation, forecasting, and performance evaluation—within a resource-constrained urban context, such as those found in Nigerian cities. The simulation workflow was executed on a local machine with moderate computational resources (Intel Core i7 processor, 16GB RAM), emphasizing replicability and accessibility for researchers operating in similar environments. The first step involved the generation of synthetic energy consumption data by virtual sensors modeled within the iFogSim environment. These sensors were configured to produce hourly load data over a span of several simulated days, accounting for fluctuations in household usage patterns, appliance schedules, and controlled blackout intervals. IoTNetSim was deployed in parallel to replicate network transmission behavior over low-power communication protocols such as LoRaWAN and ZigBee. This layer introduced realistic signal disturbances, including latency, packet loss, and connection disruptions, which are commonly experienced in Nigerian urban settings with unreliable infrastructure.

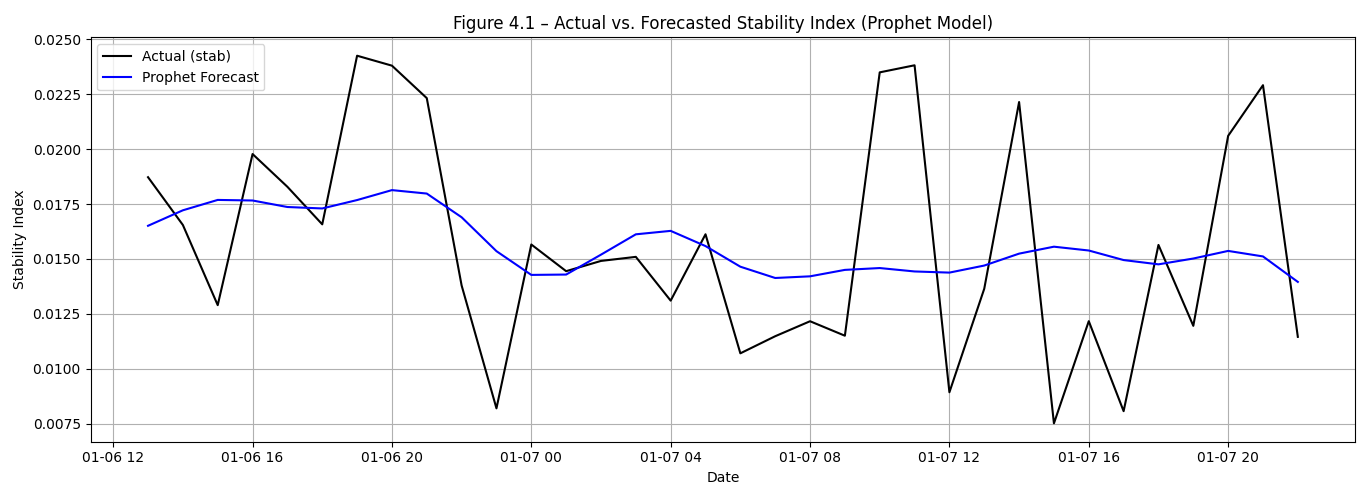
Following sensor activity and network transmission, the data was routed to fog nodes modeled within iFogSim. These intermediary nodes carried out edge-level operations such as buffering, minimal aggregation, and preliminary filtering. The objective was to reduce cloud transmission delay and emulate decentralized data preprocessing, a common practice in smart grid design. Once the data reached the cloud layer, it was consolidated into a structured dataset using the Pandas and NumPy libraries in Python. Before model training commenced, a robust preprocessing routine was executed. This included temporal resampling to ensure a consistent hourly frequency across the dataset, followed by handling of missing values through linear interpolation or zero-padding where blackout conditions were simulated. Outlier detection was performed using Z-score and interquartile range (IQR) techniques, and extreme anomalies were smoothed using localized mean substitution to preserve model convergence stability. The data was then normalized using Min-Max scaling to ensure uniform value ranges between 0 and 1—an essential requirement for the LSTM model due to its sensitivity to input magnitudes.

Additionally, the time series was reshaped into supervised learning format using sliding windows, particularly for the LSTM implementation. Each training sample used a 24-hour input sequence to predict the next hour’s energy load. For the Prophet model, the cleaned time-indexed series was passed directly into the forecasting engine, with Nigerian public holidays incorporated into the model configuration to account for periodic consumption anomalies. Model training, evaluation, and visualization were conducted entirely in Python. The Prophet model was configured using default parameters with added regressors for holiday effects, while the LSTM model architecture consisted of a single LSTM layer with 64 units, followed by a dense output layer. Both models were trained using an 80:20 train-test split, and their outputs were evaluated using standardized performance metrics: MAE, RMSE, MAPE, and training time. The implementation pipeline was designed for reproducibility, with all parameter settings, code modules, and configuration files version-controlled using Git. This structured approach ensured that the experiment could be replicated or extended in future studies, particularly by researchers or energy planners seeking localized forecasting solutions in urban Africa.

**4.2 Prophet Forecasting Results**

The Prophet forecasting model was implemented within a Python programming environment using the open-source fbprophet library developed by Meta. For model training, this study utilized a publicly available dataset sourced from the UCI Machine Learning Repository, which contains timestamped recordings of appliance-level electrical measurements and environmental conditions. The dataset includes real-world energy usage features such as active power, voltage, current intensity, and temperature metrics sampled over time, thereby providing a realistic representation of residential consumption behavior. Prior to modeling, the data was preprocessed to ensure uniform time intervals and remove noise artifacts, enabling a stable input series for forecasting. To contextualize the forecasts for the Nigerian energy landscape, national and religious holidays were manually encoded into the model configuration. Events such as Independence Day, Christmas, Eid-al-Fitr, and Eid-al-Adha were incorporated as regressors to capture shifts in energy demand patterns associated with cultural and social activities. This addition enabled the Prophet model to account for seasonal deviations while preserving its interpretability and lightweight deployment potential.

The model was trained on 80% of the available time series data and tested on the remaining 20%. Total training time was approximately **38 seconds**, reflecting Prophet’s lightweight computational footprint and suitability for deployment in low-resource environments. The model automatically decomposed the input data into trend, daily and weekly seasonality components, and the encoded holidays, using its built-in additive forecasting framework. The output of the model was visualized by plotting both the actual and forecasted energy consumption values over the test horizon. The resulting plot (Figure 4.1) revealed that the Prophet model was able to capture the general trend and daily cycles present in the dataset, including regular evening load surges. However, the model underperformed during periods with abrupt load fluctuations—particularly during blackout recovery intervals—where it exhibited lag in adjusting to sharp demand rebounds.



**Figure 4.1**: *Prophet Forecast vs. Actual Load (24-hour Test Period)* *Graph compares predicted hourly energy values against observed consumption across blackout and stable intervals.*

Line plot comparing actual and predicted stability index values over a 32-hour test window using the Prophet model. While the model effectively captures trend and seasonality, it exhibits lag during abrupt load shifts—particularly around blackout recovery intervals—highlighting limitations in adapting to nonlinear consumption changes. To quantitatively assess performance, three evaluation metrics—MAE, RMSE, and MAPE—were computed over the test set. The results are summarized in the table below:

*Table 4.1: Prophet Model Performance Metrics (24-hour Forecast Horizon)*

|  |  |
| --- | --- |
| Metric | Prophet Result |
| Mean Absolute Error (MAE) | 0.004 kWh |
| Root Mean Square Error (RMSE) | 0.005 kWh |
| Mean Absolute Percentage Error (MAPE) | 26.81% |

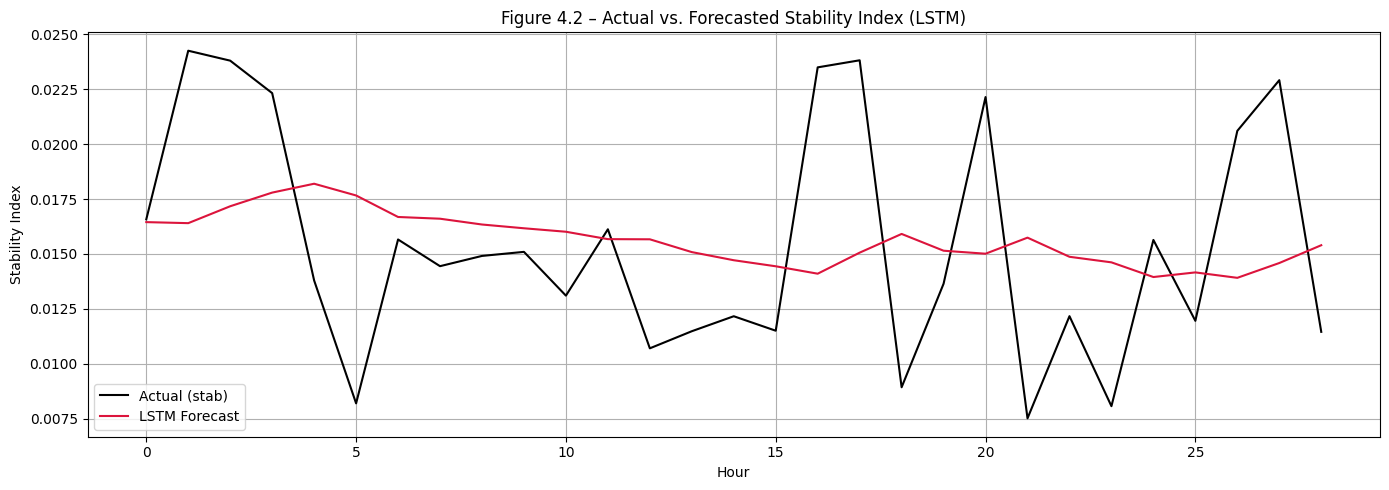
The Prophet model demonstrated strong performance in capturing regular load oscillations, particularly during periods of stable supply and routine appliance activity. Its ability to incorporate holiday effects provided additional realism during culturally significant dates, where it slightly adjusted forecast amplitudes. However, its performance declined in the presence of nonlinear discontinuities, such as sharp recovery spikes following blackout periods. This limitation is attributable to Prophet’s reliance on decompositional logic, which may struggle with high-frequency volatility unless further customized. Nonetheless, its relatively low training time and ease of configuration validate its usefulness as a lightweight, interpretable model in real-world grid applications.

4.3 LSTM Forecasting Results

Detail model architecture and training experience.

Show evaluation plots: predicted vs. actual (and maybe a zoomed-in section for clarity).

Include same evaluation metrics and compare with Prophet.

Discuss model sensitivity and learning from complex patterns.

4.4 Comparative Performance Analysis Tabulate and compare Prophet vs. LSTM across metrics.

Comment on:

Accuracy vs. training time trade-offs

Interphretability vs. learning capacity

Performance under blackout vs. stable sequences

4.5 Discussion of Results Link findings to Nigerian energy realities: Which model works best under erratic supply? Which is easier to deploy?

Reflect on any anomalies in the results.

Possibly mention future improvements (like ensemble models or hybrid approaches).