Analyzing and Forecasting Hourly Demand & Distribution Trends in the Electric Grid using EIA, LSTM, and Meta Prophet

Aniket Konkar
Department of Computer Science
George Washington University
aniket.konkar@gwu.edu

Andrew Paladino
Department of Computer Science
George Washington University
andypaladino@gwu.edu

***Abstract*—Energy-related catastrophes, such as hurricanes and industrial fires, increasingly threaten the stability and resilience of the U.S. energy sector. Recent events, including Hurricane Helene's infrastructure failures in North Carolina and the Conyers Chemical Plant fire in Georgia, have highlighted the critical need for timely insights into the cascading effects of such disasters on energy supply chains. This paper explores the potential of leveraging open-source datasets provided by the Energy Information Administration (EIA) to quantitatively measure the impacts of catastrophic events on domestic energy systems. By utilizing the EIA's robust API, we retrieve and analyze data across multiple energy sectors, focusing on cases like power plant disruptions and shifts in energy supply chains following infrastructure failures. Our methodology includes data scraping, cleaning, and visualization, with a forward-looking aim to incorporate predictive modeling to assess the potential impacts of future events. This study provides a foundational framework for understanding the interdependencies between natural disasters and energy infrastructure, offering insights to enhance preparedness and mitigate disruptions within the U.S. energy landscape.**

Keywords— Electric grid analysis, energy demand forecasting, EIA API, LSTM models, Meta Prophet, time-series forecasting, energy distribution trends

I. Introduction (*Heading 1*)

Energy-related catastrophes are on the rise around the globe, particularly within the United States, and each has major impacts on the US Energy Sector. A prime example is the devastating effects and aftermath of Hurricane Helene, which was exacerbated by failed infrastructure and left southern communities like Asheville, North Carolina without power, critical resources, and in total disrepair. Another example is the massive chemical fire at the Conyers Chemical Plant outside of Atlanta, caused by a fire that ignited on the roof of a building. Fallout aside, the fire is having massive effects on the local community, as schools are being cancelled due to airborne chlorine and smoke. What if it was possible to physically measure the impacts of these catastrophes on domestic energy with open-source datasets?

II. Problem Definition

The Energy Information Association (EIA)[1] is the largest repository of Energy-related data and publishes updates in near-real time for nearly 20 energy sectors in the United States. They have an open-source API with detailed endpoints for each of these sectors. Aniket and I plan to pick a few current catastrophic events, scrape relevant datasets via the EIA API, and determine whether we can physically measure any impacts on the energy industry and associated supply chain. For example, Hurricane Helene caused failures in a Tennessee dam and a nearby power plant. The results of these failures, and others alike, would likely lead to a shift in the domestic energy supply chain, meaning new suppliers to compensate for the increased demand due to the loss of existing resources.

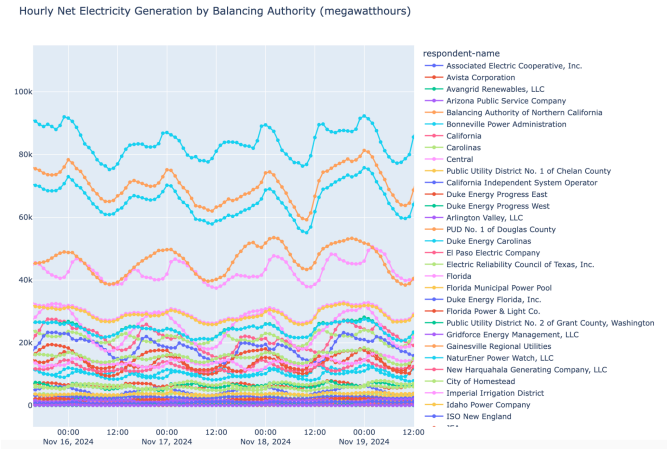
Definitions

Data Source: EIA Electricity API[1]
Electricity Power Operations (Hourly) –
Emissions Consumed - Refers to the amount of CO2 emissions attributed to the electricity consumed by the balancing authority.
Emissions Generated - The total amount of electricity consumed by the balancing authority.

Emissions Exported - CO2 emissions associated with the electricity exported to other balancing authorities.
Generated Emissions Rate - the rate of CO2 emissions per unit of electricity generated
Emissions Imported - CO2 emissions associated with the electricity imported from other balancing authorities.
Consumed Electricity - the amount of electricity consumed by the balancing authority
Demand - The actual electricity demand within the balancing authority, typically measured in megawatts (MW).
Day Ahead Demand Forecast - The predicted electricity demand for the next day, typically used for planning and balancing operations.
Net Generation - The total electricity generation within the balancing authority minus the electricity used internally for plant operations (auxiliary power).
Total Interchange - the net flow of electricity exchanged between the balancing authority and neighboring areas. Positive values indicate net imports, while negative values indicate net exports.

III. Data Processing & Used Models

The figures below show what the clean data looks like for one category.



	period	respondent	respondent-name	type	type-name	value	value-units
8	2024-12-01T00	AECI	Associated Electric Cooperative, Inc.	NG	Net generation	2840.0	megawatthours
18	2024-12-01T00	AVA	Avista Corporation	NG	Net generation	724.0	megawatthours
25	2024-12-01T00	AVRN	Avangrid Renewables, LLC	NG	Net generation	584.0	megawatthours
35	2024-12-01T00	AZPS	Arizona Public Service Company	NG	Net generation	2305.0	megawatthours
45	2024-12-01T00	BANC	Balancing Authority of Northern California	NG	Net generation	1658.0	megawatthours
...	...	...	...	...	...	...	...
499154	2024-11-01T00	WACM	Western Area Power Administration - Rocky Moun...	NG	Net generation	3529.0	megawatthours
499158	2024-11-01T00	WALC	Western Area Power Administration - Desert Sou...	NG	Net generation	924.0	megawatthours
499162	2024-11-01T00	WAUW	Western Area Power Administration - Upper Grea...	NG	Net generation	4.0	megawatthours
499164	2024-11-01T00	WWA	NaturEner Wind Watch, LLC	NG	Net generation	2.0	megawatthours
499166	2024-11-01T00	YAD	Alcoa Power Generating, Inc. - Yadkin Division	NG	Net generation	124.0	megawatthours

53319 rows × 7 columns

This data is 30 days of 1-hour interval electricity supplied information that is passed throughout the country. Some of its attributes are when chosen to model showcase a seasonal trend on the graph hence, we chose the Prophet[4] model for modeling those attributes. The rest are modeled using the Long Short-Term Memory(LSTM)[2] architecture as those are very non-linear and are high dimensional.

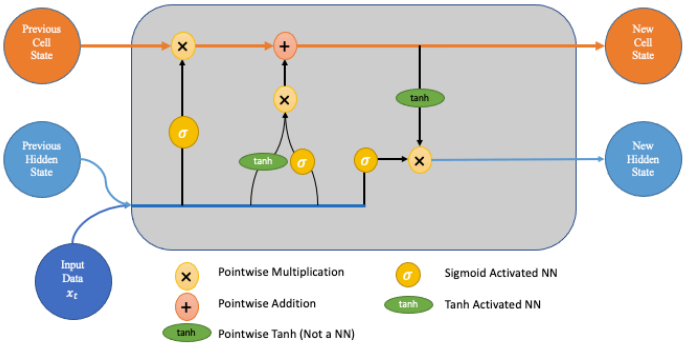
Why does Long Short-Term Memory suit this use case?

- Recurrent Neural Network (RNN)
- Long, complex, non-linear modeling using sequencing
- Arguably the best machine learning algorithm for time series forecasting and prediction that can be trained on complex or multivariate datasets like electricity generation and consumption datasets.

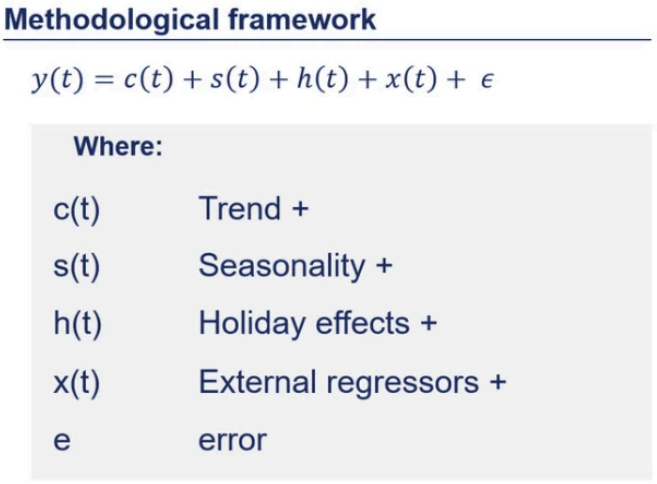
Why does the Prophet model work (For other attributes)?

- Piecewise Linear Regression
- Simple, quick, seasonal or trend-based modeling that includes noise/error metrics
- Arguably the most trendy (non-traditional, but new on the scene) time series forecasting model that is quickly being used across the industry.
- Fantastic for forecasting trends while evaluating seasonality, holidays, weekend/weekday
- Wanted to provide a simple approach

LSTM



Prophet



Data Fetching & Normalization

We built a function eia_report to pull data from the Electrical Power Operations > Electric Grid Monitor Endpoint. The function ran on a front-end notebook with input parameters. The result was 500K Records.

Data Cleaning & Filtering

Data cleaning to change time series values into timestamps and objects representing numbers into values. Data Filtering to separate each category in prep for input into the Prophet or LSTM model

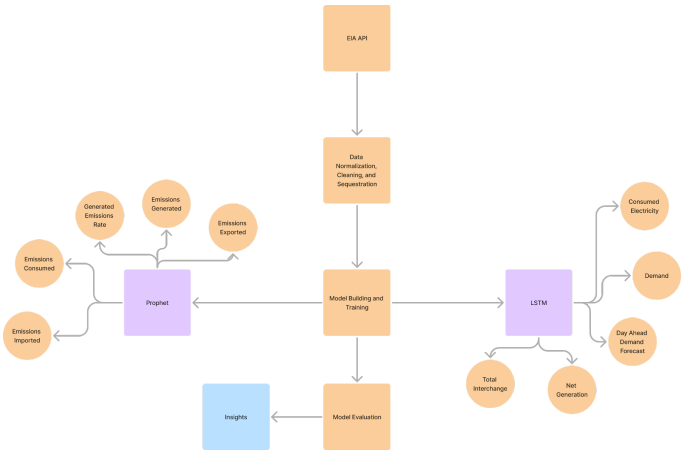
```
##############################
#### preliminary cleaning ####
##############################

hourly_electricity['period'] = pd.to_datetime(hourly_electricity['period'])
hourly_electricity['value'] = hourly_electricity['value'].astype(float)

##################
##### prophet #####
##################

emmissions_consumed = hourly_electricity[hourly_electricity['type-name'] == 'CO2 Emissions Consumed'] #Prophet
emmissions_generated = hourly_electricity[hourly_electricity['type-name'] == 'CO2 Emissions Generated'] #Prophet
emmissions_exported = hourly_electricity[hourly_electricity['type-name'] == 'CO2 Emissions Exported'] #Prophet
emissions_imported = hourly_electricity[hourly_electricity['type-name'] == 'CO2 Emissions Imported'] #Prophet
generated_emissions_rate = hourly_electricity[hourly_electricity['type-name'] == 'Generated CO2 Emissions Rate'] #Prophet


##############
#### LSTM ####
##############


consumed_electricity = hourly_electricity[hourly_electricity['type-name'] == 'Consumed Electricity'] #LSTM
demand = hourly_electricity[hourly_electricity['type-name'] == 'Demand'] #LSTM
day_ahead_demand_forecast = hourly_electricity[hourly_electricity['type-name'] == 'Day-ahead demand forecast'] #LSTM
net_generation = hourly_electricity[hourly_electricity['type-name'] == 'Net generation'] #LSTM
total_interchange = hourly_electricity[hourly_electricity['type-name'] == 'Total interchange'] #LSTM
```

3.2s

IV. Objective & Plan of Action

For each electric grid category, and for each operating authority, use 30 days of historic hourly data (November, 2024) to predict, analyze and forecast trends for the next 30 days (December). In the case of LSTM (72 hours)

1. Data Normalization, Cleaning, & Sequestering
2. Using the **Prophet model** on data categories that have **clear trends** and **change seasonally** such as CO2 emissions consumed, CO2 emissions generated, CO2 Emissions exported, generated CO2 Emission Rate, CO2 emissions imported and Total Electricity Interchange
3. Train **LSTM** on categories that are influenced by **external factors** such as natural disasters, or categories having **non-linear patterns** that are **complex to model** and align with real demand.
4. Compare one category with both models to analyze differences in forecast and performance
5. Model Evaluation



Prophet Model & The Algorithm

Function 1: Prophet Forecast

```
from prophet import Prophet
import pandas as pd

def prophet_forecast(data, periods):
    # Prepare data for Prophet
    df_prophet = data.rename(columns={'period': 'ds', 'value': 'y'})

    # Initialize and fit the model
    model = Prophet()
    model.fit(df_prophet)

    # Make future predictions
    future = model.make_future_dataframe(periods=periods, freq='H')
    forecast = model.predict(future)

    return forecast
```

Function 2: Prophet Prediction

```
def prophet_prediction(dataframe=None, periods=None):

    dataframe_forecast = pd.DataFrame()

    for oem in dataframe.respondent.unique():
        oem_df = emmissions_consumed[emmissions_consumed['respondent'] == oem]
        forecast = prophet_forecast(oem_df, periods=periods) #### predicting the next 30 days or 720 hours
        forecast['name'] = oem
        dataframe_forecast = pd.concat([forecast, dataframe_forecast])

    return dataframe_forecast


prophet_prediction(emmissions_consumed, periods=400)
```

LSTM Model Training & The Algorithm

Function 1: Create Sequences

```
from datetime import timedelta
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense
import numpy as np
scaler = MinMaxScaler()


def create_sequences(data, sequence_length):
    sequences, targets = [], []
    for i in range(len(data) - sequence_length):
        sequences.append(data[i:i+sequence_length])
        targets.append(data[i+sequence_length])
    return np.array(sequences), np.array(targets)
```

Training Time: for all OEMs, for one category: 45 minutes for 30 days as specified by the horizon forecast (720 hours = 30 days), 90 minutes for 2 categories and so on.

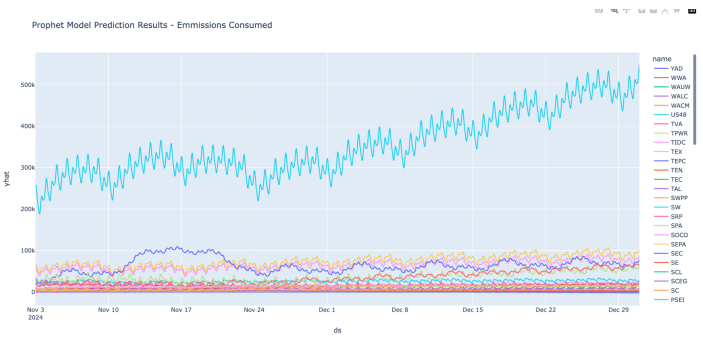
Downside to LSTM, sequential sequencing, multiple gates, and backpropagation through time

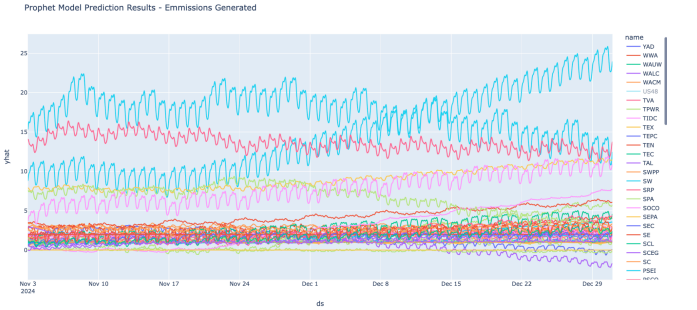
Function 2: LSTM Forecast

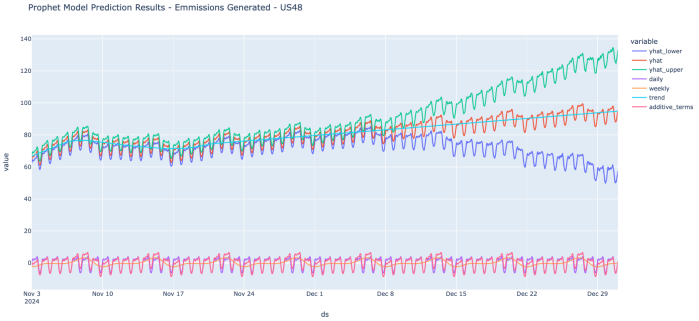
```python
def lstm_forecast(data, sequence_length, epochs, batch_size, scaler, forecast_horizon=720):
    # Prepare data for LSTM
    sequences, targets = create_sequences(data, sequence_length)
    X = sequences.reshape((sequences.shape[0], sequences.shape[1], 1))
    y = targets

    # Define LSTM model
    model = Sequential()
    model.add(LSTM(50, activation='relu', input_shape=(sequence_length, 1)))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')

    # Train the model
    model.fit(X, y, epochs=epochs, batch_size=batch_size, verbose=1)

    # Multi-step forecasting
    forecast = []
    last_sequence = X[-1]  # Start with the last observed sequence
    for _ in range(forecast_horizon):
        # Predict the next step
        next_step = model.predict(last_sequence.reshape(1, sequence_length, 1))
        forecast.append(next_step[0][0])  # Append the prediction

        # Update the sequence by appending the predicted value and removing the oldest value
        last_sequence = np.append(last_sequence[1:], next_step, axis=0)

    # Convert forecast back to original scale
    forecast_original = scaler.inverse_transform(np.array(forecast).reshape(-1, 1))
    return forecast_original
```

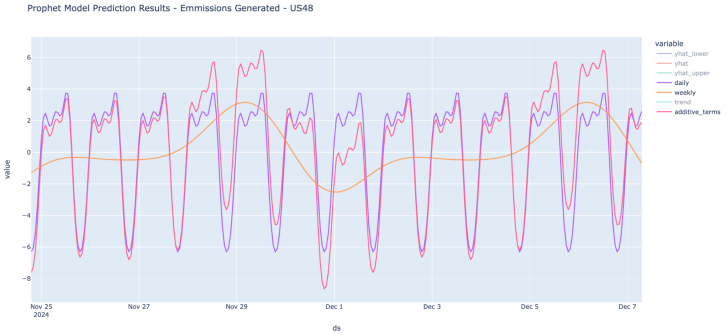
Function 3: LSTM Prediction

```python
def lstm_prediction(dataframe=None, forecast_horizon=720, sequence_length=24, epochs=20, batch_size=32, scaler=None):
    from datetime import timedelta
    scaler = MinMaxScaler()

    dataframe['value_scaled'] = scaler.fit_transform(dataframe[['value']])

    all_forecasts = []  # Store all forecasts for all respondents

    for oem in dataframe['respondent'].unique():
        print(f"Processing respondent: {oem}")

        # Filter data for the current respondent
        oem_df = dataframe[dataframe['respondent'] == oem]
        data_array = oem_df['value_scaled'].values.reshape(-1, 1)  # Reshape for compatibility

        # Skip respondents with insufficient data
        if len(data_array) <= sequence_length:
            print(f"Skipping respondent {oem} due to insufficient data.")
            continue

        # Generate LSTM forecast
        forecast = lstm_forecast(
            data_array,
            sequence_length=sequence_length,
            epochs=epochs,
            batch_size=batch_size,
            scaler=scaler,
            forecast_horizon=forecast_horizon
        )

        # Generate forecast timestamps
        forecast_start = oem_df['period'].max() + timedelta(hours=1)
        forecast_timestamps = [forecast_start + timedelta(hours=i) for i in range(forecast_horizon)]

        # Combine timestamps and forecasts into a DataFrame for this respondent
        forecast_df = pd.DataFrame({
            'period': forecast_timestamps,
            'respondent': oem,
            'forecast': forecast.flatten()
        })

        # Append to the list of all forecasts
        all_forecasts.append(forecast_df)


    final_forecast_df = pd.concat(all_forecasts, ignore_index=True)
    return final_forecast_df
```







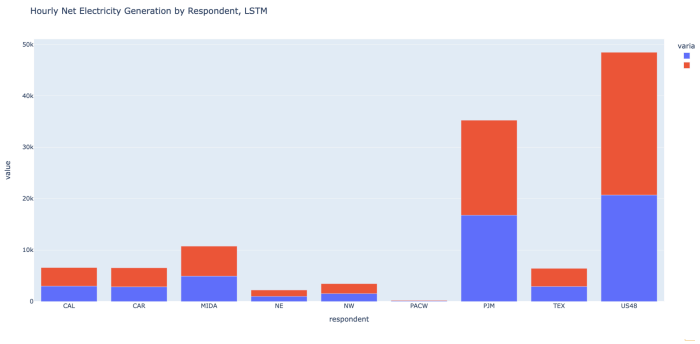


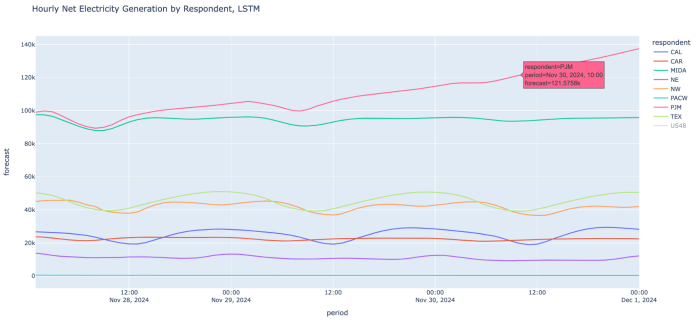
V. Evaluation & Results

Prophet Model Evaluation & Results

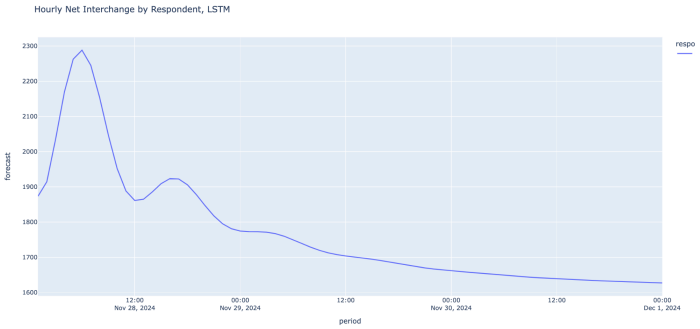
LSTM Model Evaluation Results

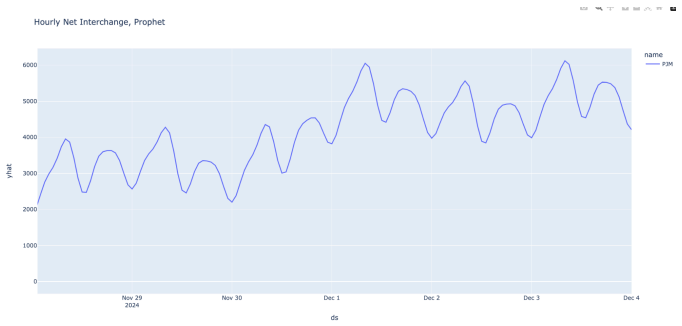
Mean Absolute Error vs. Root Mean Squared Error





LSTM vs. Prophet: PJM Interconnection Total Net Interchange: Comparing Results





VI. Conclusion

We conclude by modeling the previous 30 days of data and can now predict the next 30 days for these OEMs. We also observed that the Prophet model is more suited to seasonal/cyclic data as it can better estimate those trends, and the LSTM model is a better architecture for modeling non-linear, complex, patterned, and high-dimensional data.

VII. References

[1] U.S. Energy Information Administration (EIA). (2023). "EIA Data Overview and API Documentation." *(Official API documentation for accessing U.S. energy data.)*

[2] Hochreiter, S., & Schmidhuber, J. (1997). "Long Short-Term Memory." *Neural Computation*, 9(8), 1735–1780. *(Foundational paper on LSTM.)*

[3] Goodfellow, I., Bengio, Y., & Courville, A. (2016). "Deep Learning." MIT Press. *(Comprehensive reference on deep learning applications in energy sectors.)*

[4] Taylor, S. J., & Letham, B. (2018). "Forecasting at Scale." *American Statistician*, 72(1), 37–45. *(Original publication on the Prophet model.)*