In [1]:

Hurricane Path Prediction - IDALIA 2023

1. Preprocessing

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call d
rive.mount("/content/drive", force_remount=True).
```

In [3]:
```
data_path = '/content/drive/My Drive/Colab Notebooks/data'
data_path = '/content/drive/My Drive/Colab Notebooks/nhc_hurricane_idalia_file
```

In [5]:
```
import geopandas as gpd
import pandas as pd
import os

# Directory containing all hurricane shapefile ZIPs
data_mining_dir = '/content/drive/My Drive/Colab Notebooks/nhc_hurricane_idali

# List all .zip files in the directory
zip_files = [os.path.join(data_mining_dir, f) for f in os.listdir(data_mining_

# Load all shapefiles into a list of GeoDataFrames
gdf_list = [gpd.read_file(zip_file) for zip_file in zip_files]

# Combine all GeoDataFrames into one
combined_gdf = gpd.GeoDataFrame(pd.concat(gdf_list, ignore_index=True))

# Display columns and first few rows
print("Combined Hurricane Data Columns:", combined_gdf.columns)
print(combined_gdf.head())
```

```
/usr/local/lib/python3.11/dist-packages/pyogrio/geopandas.py:275: UserWarning:
More than one layer found in 'al102023_5day_002A.zip': 'al102023-002A_5day_lin'
(default), 'al102023-002A_5day_pgn', 'al102023-002A_5day_pts', 'al102023-002A_w
w_wwlin'. Specify layer parameter to avoid this warning.
  result = read_func(
/usr/local/lib/python3.11/dist-packages/pyogrio/geopandas.py:275: UserWarning:
More than one layer found in 'al102023_5day_002.zip': 'al102023-002_5day_lin'
(default), 'al102023-002_5day_pgn', 'al102023-002_5day_pts', 'al102023-002_ww_w
wlin'. Specify layer parameter to avoid this warning.
  result = read_func(
/usr/local/lib/python3.11/dist-packages/pyogrio/geopandas.py:275: UserWarning:
More than one layer found in 'al102023_5day_003.zip': 'al102023-003_5day_lin'
(default), 'al102023-003_5day_pgn', 'al102023-003_5day_pts', 'al102023-003_ww_w
wlin'. Specify layer parameter to avoid this warning.
  result = read_func(
/usr/local/lib/python3.11/dist-packages/pyogrio/geopandas.py:275: UserWarning:
More than one layer found in 'al102023_5day_001.zip': 'al102023-001_5day_lin'
(default), 'al102023-001_5day_pgn', 'al102023-001_5day_pts', 'al102023-001_ww_w
wlin'. Specify layer parameter to avoid this warning.
  result = read_func(
/usr/local/lib/python3.11/dist-packages/pyogrio/geopandas.py:275: UserWarning:
More than one layer found in 'al102023_5day_001A.zip': 'al102023-001A_5day_lin'
(default), 'al102023-001A_5day_pgn', 'al102023-001A_5day_pts', 'al102023-001A_w
w_wwlin'. Specify layer parameter to avoid this warning.
  result = read_func(
/usr/local/lib/python3.11/dist-packages/pyogrio/geopandas.py:275: UserWarning:
More than one layer found in 'al102023_5day_007.zip': 'al102023-007_5day_lin'
(default), 'al102023-007_5day_pgn', 'al102023-007_5day_pts', 'al102023-007_ww_w
wlin'. Specify layer parameter to avoid this warning.
  result = read_func(
/usr/local/lib/python3.11/dist-packages/pyogrio/geopandas.py:275: UserWarning:
More than one layer found in 'al102023_5day_006A.zip': 'al102023-006A_5day_lin'
(default), 'al102023-006A_5day_pgn', 'al102023-006A_5day_pts', 'al102023-006A_w
w_wwlin'. Specify layer parameter to avoid this warning.
  result = read_func(
/usr/local/lib/python3.11/dist-packages/pyogrio/geopandas.py:275: UserWarning:
More than one layer found in 'al102023_5day_004A.zip': 'al102023-004A_5day_lin'
(default), 'al102023-004A_5day_pgn', 'al102023-004A_5day_pts', 'al102023-004A_w
w_wwlin'. Specify layer parameter to avoid this warning.
  result = read_func(
/usr/local/lib/python3.11/dist-packages/pyogrio/geopandas.py:275: UserWarning:
More than one layer found in 'al102023_5day_004.zip': 'al102023-004_5day_lin'
(default), 'al102023-004_5day_pgn', 'al102023-004_5day_pts', 'al102023-004_ww_w
wlin'. Specify layer parameter to avoid this warning.
  result = read_func(
/usr/local/lib/python3.11/dist-packages/pyogrio/geopandas.py:275: UserWarning:
More than one layer found in 'al102023_5day_005A.zip': 'al102023-005A_5day_lin'
(default), 'al102023-005A_5day_pgn', 'al102023-005A_5day_pts', 'al102023-005A_w
w_wwlin'. Specify layer parameter to avoid this warning.
  result = read_func(
/usr/local/lib/python3.11/dist-packages/pyogrio/geopandas.py:275: UserWarning:
More than one layer found in 'al102023_5day_006.zip': 'al102023-006_5day_lin'
(default), 'al102023-006_5day_pgn', 'al102023-006_5day_pts', 'al102023-006_ww_w
wlin'. Specify layer parameter to avoid this warning.
```

```
  result = read_func(
/usr/local/lib/python3.11/dist-packages/pyogrio/geopandas.py:275: UserWarning:
More than one layer found in 'al102023_5day_003A.zip': 'al102023-003A_5day_lin'
(default), 'al102023-003A_5day_pgn', 'al102023-003A_5day_pts', 'al102023-003A_w
w_wwlin'. Specify layer parameter to avoid this warning.
  result = read_func(
Combined Hurricane Data Columns: Index(['STORMNAME', 'STORMTYPE', 'ADVDATE', 'A
DVISNUM', 'STORMNUM', 'FCSTPRD',
       'BASIN', 'geometry'],
      dtype='object')
  STORMNAME STORMTYPE                      ADVDATE ADVISNUM  STORMNUM  \
0       Ten        TD   100 AM CDT Sun Aug 27 2023       2A      10.0
1       Ten        TD  1000 PM CDT Sat Aug 26 2023        2      10.0
2       Ten        TD   400 AM CDT Sun Aug 27 2023        3      10.0
3       Ten        TD   400 PM CDT Sat Aug 26 2023        1      10.0
4       Ten        TD   700 PM CDT Sat Aug 26 2023       1A      10.0

   FCSTPRD BASIN                                           geometry
0    120.0    AL  LINESTRING (-86.8 20.7, -86.4 20.9, -86.2 20.9...
1    120.0    AL  LINESTRING (-86.4 21.1, -86.4 20.9, -86.2 20.9...
2    120.0    AL  LINESTRING (-86.8 20.1, -86.5 20.1, -86.3 20.6...
3    120.0    AL  LINESTRING (-86.1 21.1, -86.3 21, -86.2 20.9, ...
4    120.0    AL  LINESTRING (-86.2 21.2, -86.3 21, -86.2 20.9, ...
/usr/local/lib/python3.11/dist-packages/pyogrio/geopandas.py:275: UserWarning:
More than one layer found in 'al102023_5day_005.zip': 'al102023-005_5day_lin'
(default), 'al102023-005_5day_pgn', 'al102023-005_5day_pts', 'al102023-005_ww_w
wlin'. Specify layer parameter to avoid this warning.
  result = read_func(
```

In [6]:
```
layers = ['al102023-001_5day_lin', 'al102023-001_5day_pgn', 'al102023-001_5day
print(layers)
```

```
['al102023-001_5day_lin', 'al102023-001_5day_pgn', 'al102023-001_5day_pts', 'al
102023-001_ww_wwlin']
```

In [9]:
```
import os
import geopandas as gpd
!pip install fiona
import fiona
import pandas as pd

# Directory containing all hurricane ZIP shapefiles
data_mining_dir = '/content/drive/My Drive/Colab Notebooks/nhc_hurricane_idali
zip_files = [os.path.join(data_mining_dir, f) for f in os.listdir(data_mining_

# Define base layer name types
layer_types = ['_5day_lin', '_5day_pgn', '_5day_pts', '_ww_wwlin']

# Function to extract advisory number from filename
def get_advisory_number(zip_filename):
    # Example: 'al102023_5day_001.zip' → '001'
    return zip_filename.split('_')[-1].replace('.zip', '')
```

```python
# For each layer type, combine that layer from all ZIPs
combined_layers = {}

for layer_type in layer_types:
    gdf_list = []
    for zip_file in zip_files:
        zip_name = os.path.basename(zip_file)
        # Construct the correct base prefix: 'al102023-001'
        base_prefix = zip_name.replace('.zip', '').replace('_5day', '').replac
        layer_name = base_prefix + layer_type
        try:
            gdf = gpd.read_file(zip_file, layer=layer_name)
            gdf_list.append(gdf)
        except Exception as e:
            print(f"Could not load {layer_name} from {zip_file}: {e}")
    if gdf_list:
        combined_layers[layer_type] = gpd.GeoDataFrame(pd.concat(gdf_list, ign
    else:
        combined_layers[layer_type] = None

# Display each combined GeoDataFrame
for layer_type, gdf in combined_layers.items():
    print(f"\n=== Layer: {layer_type} ===")
    if gdf is not None:
        print(gdf.head())
    else:
        print("No data found for this layer.")
```

```
Requirement already satisfied: fiona in /usr/local/lib/python3.11/dist-packages
(1.10.1)
Requirement already satisfied: attrs>=19.2.0 in /usr/local/lib/python3.11/dist-
packages (from fiona) (25.3.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packag
es (from fiona) (2025.6.15)
Requirement already satisfied: click~=8.0 in /usr/local/lib/python3.11/dist-pac
kages (from fiona) (8.2.1)
Requirement already satisfied: click-plugins>=1.0 in /usr/local/lib/python3.11/
dist-packages (from fiona) (1.1.1.2)
Requirement already satisfied: cligj>=0.5 in /usr/local/lib/python3.11/dist-pac
kages (from fiona) (0.7.2)
Could not load al102023-020_ww_wwlin from /content/drive/My Drive/Colab Noteboo
ks/nhc_hurricane_idalia_files/data_mining/al102023_5day_020.zip: IllegalArgumen
tException: point array must contain 0 or >1 elements

Could not load al102023-019_ww_wwlin from /content/drive/My Drive/Colab Noteboo
ks/nhc_hurricane_idalia_files/data_mining/al102023_5day_019.zip: IllegalArgumen
tException: point array must contain 0 or >1 elements

Could not load al102023-018A_ww_wwlin from /content/drive/My Drive/Colab Notebo
oks/nhc_hurricane_idalia_files/data_mining/al102023_5day_018A.zip: IllegalArgum
entException: point array must contain 0 or >1 elements

Could not load al102023-019A_ww_wwlin from /content/drive/My Drive/Colab Notebo
oks/nhc_hurricane_idalia_files/data_mining/al102023_5day_019A.zip: IllegalArgum
entException: point array must contain 0 or >1 elements

Could not load al102023-020A_ww_wwlin from /content/drive/My Drive/Colab Notebo
oks/nhc_hurricane_idalia_files/data_mining/al102023_5day_020A.zip: IllegalArgum
entException: point array must contain 0 or >1 elements

Could not load al102023-018_ww_wwlin from /content/drive/My Drive/Colab Noteboo
ks/nhc_hurricane_idalia_files/data_mining/al102023_5day_018.zip: IllegalArgumen
tException: point array must contain 0 or >1 elements

Could not load al102023-014_ww_wwlin from /content/drive/My Drive/Colab Noteboo
ks/nhc_hurricane_idalia_files/data_mining/al102023_5day_014.zip: IllegalArgumen
tException: point array must contain 0 or >1 elements

Could not load al102023-013_ww_wwlin from /content/drive/My Drive/Colab Noteboo
ks/nhc_hurricane_idalia_files/data_mining/al102023_5day_013.zip: IllegalArgumen
tException: point array must contain 0 or >1 elements

Could not load al102023-013A_ww_wwlin from /content/drive/My Drive/Colab Notebo
oks/nhc_hurricane_idalia_files/data_mining/al102023_5day_013A.zip: IllegalArgum
entException: point array must contain 0 or >1 elements

Could not load al102023-014A_ww_wwlin from /content/drive/My Drive/Colab Notebo
oks/nhc_hurricane_idalia_files/data_mining/al102023_5day_014A.zip: IllegalArgum
entException: point array must contain 0 or >1 elements

Could not load al102023-029_ww_wwlin from /content/drive/My Drive/Colab Noteboo
ks/nhc_hurricane_idalia_files/data_mining/al102023_5day_029.zip: Layer 'al10202
```

```
3-029_ww_wwlin' could not be opened

=== Layer: _5day_lin ===
  STORMNAME STORMTYPE                      ADVDATE ADVISNUM  STORMNUM  \
0       Ten        TD   100 AM CDT Sun Aug 27 2023       2A      10.0
1       Ten        TD  1000 PM CDT Sat Aug 26 2023        2      10.0
2       Ten        TD   400 AM CDT Sun Aug 27 2023        3      10.0
3       Ten        TD   400 PM CDT Sat Aug 26 2023        1      10.0
4       Ten        TD   700 PM CDT Sat Aug 26 2023       1A      10.0

   FCSTPRD BASIN                                          geometry
0    120.0    AL  LINESTRING (-86.8 20.7, -86.4 20.9, -86.2 20.9...
1    120.0    AL  LINESTRING (-86.4 21.1, -86.4 20.9, -86.2 20.9...
2    120.0    AL  LINESTRING (-86.8 20.1, -86.5 20.1, -86.3 20.6...
3    120.0    AL  LINESTRING (-86.1 21.1, -86.3 21, -86.2 20.9, ...
4    120.0    AL  LINESTRING (-86.2 21.2, -86.3 21, -86.2 20.9, ...

=== Layer: _5day_pgn ===
  STORMNAME STORMTYPE                      ADVDATE ADVISNUM  STORMNUM  \
0       Ten        TD   100 AM CDT Sun Aug 27 2023       2A      10.0
1       Ten        TD  1000 PM CDT Sat Aug 26 2023        2      10.0
2       Ten        TD   400 AM CDT Sun Aug 27 2023        3      10.0
3       Ten        TD   400 PM CDT Sat Aug 26 2023        1      10.0
4       Ten        TD   700 PM CDT Sat Aug 26 2023       1A      10.0

   FCSTPRD BASIN                                          geometry
0    120.0    AL  POLYGON ((-86.40915 20.27286, -86.38139 20.268...
1    120.0    AL  POLYGON ((-86.39421 20.26079, -86.3663 20.2569...
2    120.0    AL  POLYGON ((-86.68235 19.68691, -86.65387 19.682...
3    120.0    AL  POLYGON ((-86.31667 19.91362, -86.2896 19.9098...
4    120.0    AL  POLYGON ((-86.31667 19.91362, -86.2896 19.9098...

=== Layer: _5day_pts ===
                    ADVDATE ADVISNUM BASIN      DATELBL DVLBL  FCSTPRD  \
0  100 AM CDT Sun Aug 27 2023       2A    AL  1:00 AM Sun     D    120.0
1  100 AM CDT Sun Aug 27 2023       2A    AL  7:00 AM Sun     S    120.0
2  100 AM CDT Sun Aug 27 2023       2A    AL  7:00 PM Sun     S    120.0
3  100 AM CDT Sun Aug 27 2023       2A    AL  7:00 AM Mon     S    120.0
4  100 AM CDT Sun Aug 27 2023       2A    AL  7:00 PM Mon     S    120.0

                     FLDATELBL  GUST   LAT   LON  ...  STORMNUM  \
0  2023-08-26 10:00 PM Sat CDT  40.0  20.7 -86.8  ...      10.0
1   2023-08-27 7:00 AM Sun CDT  45.0  20.9 -86.4  ...      10.0
2   2023-08-27 7:00 PM Sun CDT  50.0  20.9 -86.2  ...      10.0
3   2023-08-28 7:00 AM Mon CDT  55.0  21.6 -85.9  ...      10.0
4   2023-08-28 7:00 PM Mon CDT  65.0  23.1 -85.7  ...      10.0

           STORMSRC  STORMTYPE               TCDVLP   TAU   TCDIR   TCSPD  \
0  Tropical Cyclone         TD  Tropical Depression   0.0   230.0     2.0
1  Tropical Cyclone         TS       Tropical Storm  12.0  9999.0  9999.0
2  Tropical Cyclone         TS       Tropical Storm  24.0  9999.0  9999.0
3  Tropical Cyclone         TS       Tropical Storm  36.0  9999.0  9999.0
4  Tropical Cyclone         TS       Tropical Storm  48.0  9999.0  9999.0
```

```
  TIMEZONE  VALIDTIME           geometry
0      CDT    27/0300  POINT (-86.8 20.7)
1      CDT    27/1200  POINT (-86.4 20.9)
2      CDT    28/0000  POINT (-86.2 20.9)
3      CDT    28/1200  POINT (-85.9 21.6)
4      CDT    29/0000  POINT (-85.7 23.1)

[5 rows x 24 columns]

=== Layer: _ww_wwlin ===
  STORMNAME STORMTYPE                      ADVDATE ADVISNUM  STORMNUM  \
0       Ten        TD   100 AM CDT Sun Aug 27 2023       2A      10.0
1       Ten        TD   100 AM CDT Sun Aug 27 2023       2A      10.0
2       Ten        TD   100 AM CDT Sun Aug 27 2023       2A      10.0
3       Ten        TD  1000 PM CDT Sat Aug 26 2023        2      10.0
4       Ten        TD  1000 PM CDT Sat Aug 26 2023        2      10.0

   FCSTPRD BASIN TCWW                                          geometry
0    120.0    AL  TWA  LINESTRING (-82.52 21.6, -82.66 21.9, -83 21.9...
1    120.0    AL  TWR  LINESTRING (-88.18 21.61, -87.1 21.61, -86.82 ...
2    120.0    AL  TWR  LINESTRING (-83.08 22.49, -83.39 22.23, -84.49...
3    120.0    AL  TWA  LINESTRING (-82.52 21.6, -82.66 21.9, -83 21.9...
4    120.0    AL  TWR  LINESTRING (-88.18 21.61, -87.1 21.61, -86.82 ...
```

In [10]:

```python
# Print column names for each hurricane data layer
for layer_type, gdf in combined_layers.items():
    print(f"\n=== Columns in Layer: {layer_type} ===")
    if gdf is not None:
        for col in gdf.columns:
            print(col)
    else:
        print("No data found for this layer.")
```

```
=== Columns in Layer: _5day_lin ===
STORMNAME
STORMTYPE
ADVDATE
ADVISNUM
STORMNUM
FCSTPRD
BASIN
geometry

=== Columns in Layer: _5day_pgn ===
STORMNAME
STORMTYPE
ADVDATE
ADVISNUM
STORMNUM
FCSTPRD
BASIN
geometry

=== Columns in Layer: _5day_pts ===
ADVDATE
ADVISNUM
BASIN
DATELBL
DVLBL
FCSTPRD
FLDATELBL
GUST
LAT
LON
MAXWIND
MSLP
SSNUM
STORMNAME
STORMNUM
STORMSRC
STORMTYPE
TCDVLP
TAU
TCDIR
TCSPD
TIMEZONE
VALIDTIME
geometry

=== Columns in Layer: _ww_wwlin ===
STORMNAME
STORMTYPE
ADVDATE
ADVISNUM
STORMNUM
FCSTPRD
BASIN
```

TCWW
geometry

2. Training GIS Data - RF, RNN, LSTM

Note: MAE unit is 1 degree Lat/Long ~ 111 km

2.1 RF

In [14]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# List of layer types
layer_types = ['_5day_lin', '_5day_pgn', '_5day_pts', '_ww_wwlin']

results_all = []

for layer in layer_types:
    df = combined_layers[layer].copy()

    # Feature engineering for each layer
    if layer == '_5day_lin':
        # Extract first coordinate from the LINESTRING
        df['coords'] = df['geometry'].apply(lambda x: x.coords[0] if x is not
        df['LAT'] = df['coords'].apply(lambda x: x[1])
        df['LON'] = df['coords'].apply(lambda x: x[0])
        features = ['LAT', 'LON', 'FCSTPRD']
        target = ['LAT', 'LON']

    elif layer == '_5day_pgn':
        # CRS-safe centroid calculation
        if df.crs is None:
            raise ValueError("GeoDataFrame has no CRS defined")
        # Use UTM zone 16N (EPSG:32616) for Atlantic hurricanes; adjust if nee
        projected_crs = 'EPSG:32616'
        df_proj = df.to_crs(projected_crs)
        df_proj['centroid'] = df_proj.geometry.centroid
        # Reproject centroids back to original CRS
        df['centroid'] = df_proj['centroid'].to_crs(df.crs)
        df['LAT'] = df['centroid'].y
        df['LON'] = df['centroid'].x
        features = ['LAT', 'LON', 'FCSTPRD']
        target = ['LAT', 'LON']

    elif layer == '_5day_pts':
```

```
        features = ['LAT', 'LON', 'GUST', 'MAXWIND', 'MSLP', 'TAU', 'TCDIR', '
        target = ['LAT', 'LON']

    elif layer == '_ww_wwlin':
        # Extract first coordinate from the LINESTRING
        df['coords'] = df['geometry'].apply(lambda x: x.coords[0] if x is not
        df['LAT'] = df['coords'].apply(lambda x: x[1])
        df['LON'] = df['coords'].apply(lambda x: x[0])
        features = ['LAT', 'LON', 'FCSTPRD']
        target = ['LAT', 'LON']

    # Remove rows with missing values
    df = df.dropna(subset=features + target)

    # Prepare X and y for prediction (predict next point)
    X = df[features][:-1]
    y = df[target][1:]

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

    for n in [50, 100]:
        rf = RandomForestRegressor(n_estimators=n, random_state=42)
        rf.fit(X_train, y_train)
        y_pred = rf.predict(X_test)
        mae = mean_absolute_error(y_test, y_pred)
        # Normalized accuracy (1 - normalized MAE)
        accuracy = 1 - mae / (y_test.max().max() - y_test.min().min())

        results_all.append({
            'Layer': layer,
            'Model': 'Random Forest',
            'Hyperparameters': f'n_estimators={n}',
            'Accuracy': round(accuracy, 4),
            'MAE': round(mae, 4)
        })

# Display results in a DataFrame
results_df = pd.DataFrame(results_all)
print(results_df[['Layer', 'Model', 'Hyperparameters', 'Accuracy', 'MAE']])
```

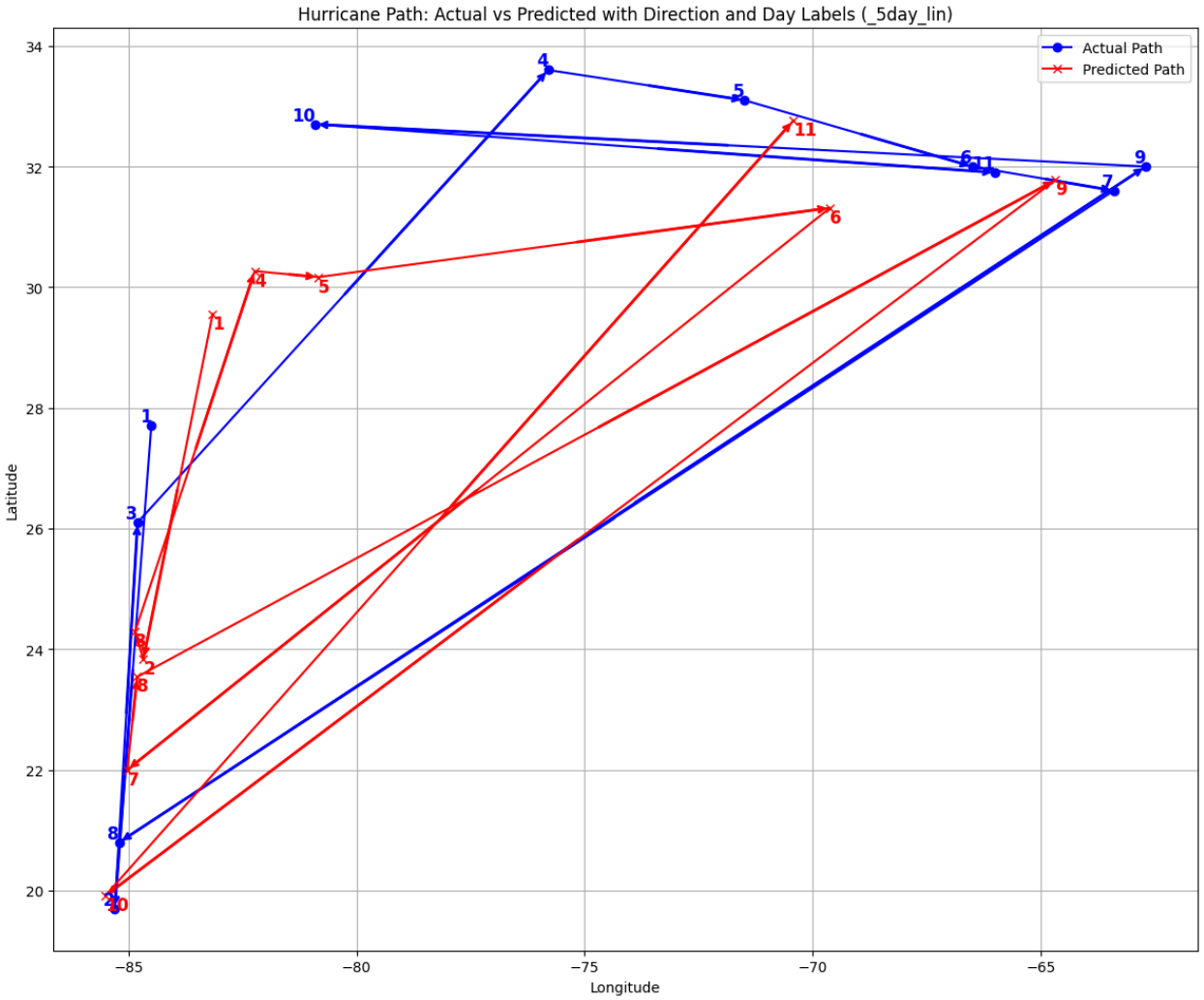
```
      Layer          Model   Hyperparameters  Accuracy     MAE
0  _5day_lin  Random Forest   n_estimators=50    0.9635  4.3402
1  _5day_lin  Random Forest  n_estimators=100    0.9637  4.3141
2  _5day_pgn  Random Forest   n_estimators=50    0.9728  3.1849
3  _5day_pgn  Random Forest  n_estimators=100    0.9727  3.1915
4  _5day_pts  Random Forest   n_estimators=50    0.9964  0.4658
5  _5day_pts  Random Forest  n_estimators=100    0.9965  0.4548
6  _ww_wwlin  Random Forest   n_estimators=50    0.9855  1.7999
7  _ww_wwlin  Random Forest  n_estimators=100    0.9854  1.8057
```

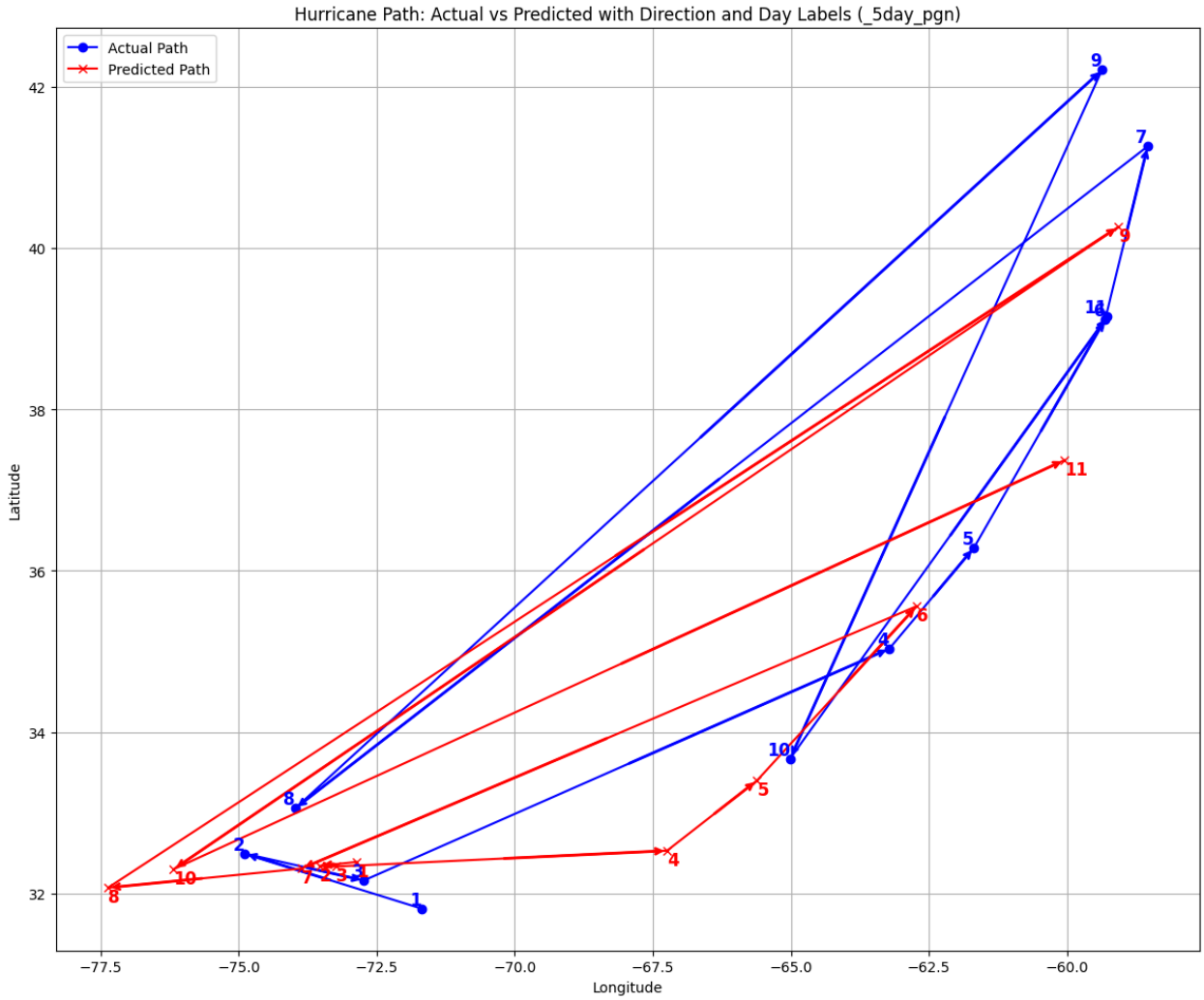
In [22]:
```
import matplotlib.pyplot as plt

import matplotlib.pyplot as plt
```

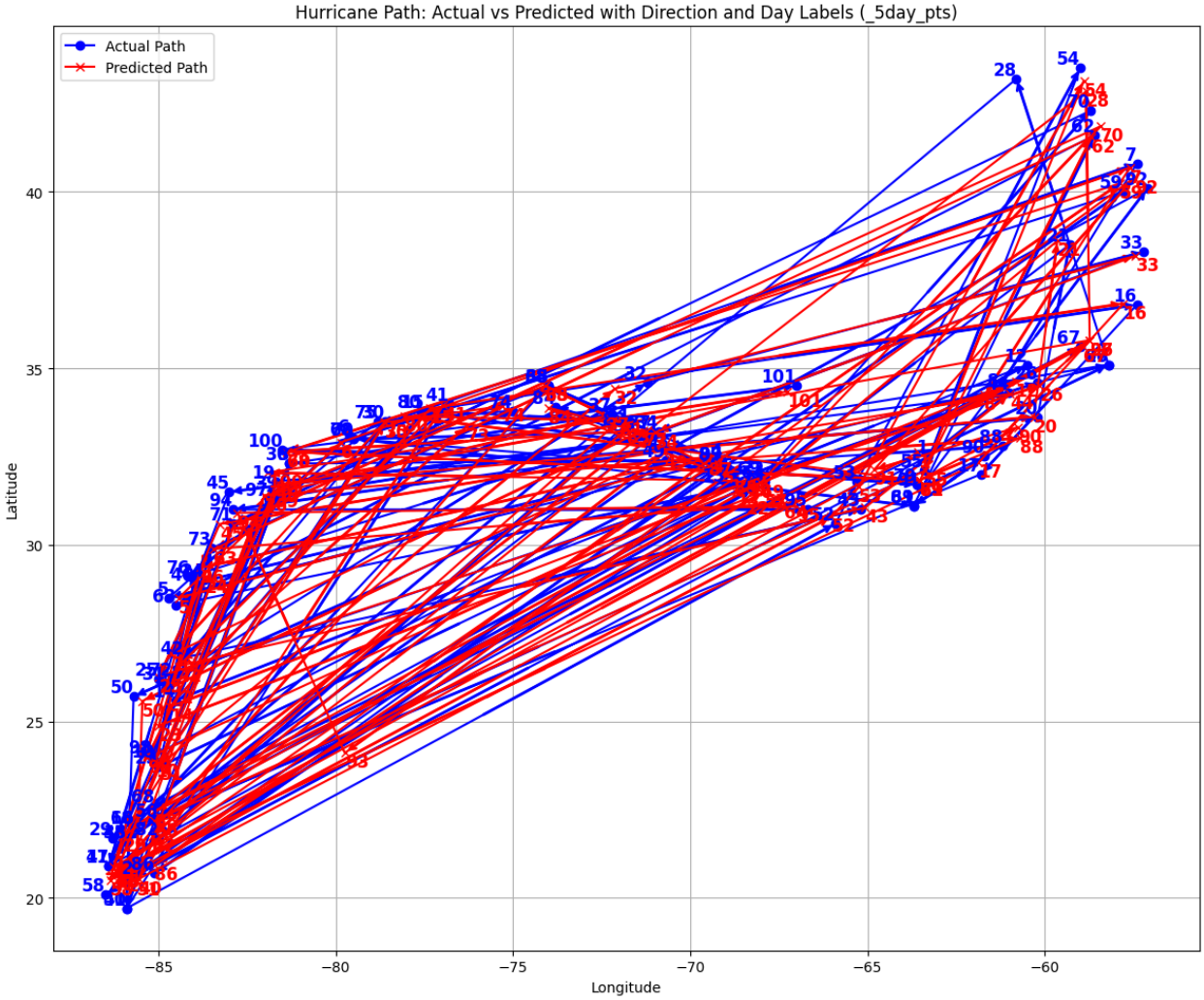
```
import matplotlib.pyplot as plt

def plot_paths_on_map_with_arrows_and_labels(actual_df, predicted_df, layer_na
    plt.figure(figsize=(12, 10))
    # Plot actual path
    plt.plot(actual_df['LON'], actual_df['LAT'], marker='o', color='blue', lab
    # Plot predicted path
    plt.plot(predicted_df['LON'], predicted_df['LAT'], marker='x', color='red'

    # Add arrows at midpoints for actual path
    for i in range(len(actual_df) - 1):
        x_start, y_start = actual_df['LON'].iloc[i], actual_df['LAT'].iloc[i]
        x_end, y_end = actual_df['LON'].iloc[i+1], actual_df['LAT'].iloc[i+1]
        x_mid = (x_start + x_end) / 2
        y_mid = (y_start + y_end) / 2
        dx = x_end - x_start
        dy = y_end - y_start
        # Scale arrows for geographic plots
        plt.annotate('', xy=(x_end, y_end), xytext=(x_mid, y_mid),
                     arrowprops=dict(arrowstyle='->', color='blue', lw=2), zor

    # Add arrows at midpoints for predicted path
    for i in range(len(predicted_df) - 1):
        x_start, y_start = predicted_df['LON'].iloc[i], predicted_df['LAT'].il
        x_end, y_end = predicted_df['LON'].iloc[i+1], predicted_df['LAT'].iloc
        x_mid = (x_start + x_end) / 2
        y_mid = (y_start + y_end) / 2
        dx = x_end - x_start
        dy = y_end - y_start
        plt.annotate('', xy=(x_end, y_end), xytext=(x_mid, y_mid),
                     arrowprops=dict(arrowstyle='->', color='red', lw=2), zord

    # Label each vertex with day number
    for i, (lat, lon) in enumerate(zip(actual_df['LAT'], actual_df['LON']), st
        plt.text(lon, lat, str(i), color='blue', fontsize=12, fontweight='bold
    for i, (lat, lon) in enumerate(zip(predicted_df['LAT'], predicted_df['LON'
        plt.text(lon, lat, str(i), color='red', fontsize=12, fontweight='bold'

    plt.title(f'Hurricane Path: Actual vs Predicted with Direction and Day Lab
    plt.xlabel('Longitude')
    plt.ylabel('Latitude')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()


from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import pandas as pd

layer_types = ['_5day_lin', '_5day_pgn', '_5day_pts', '_ww_wwlin']
```

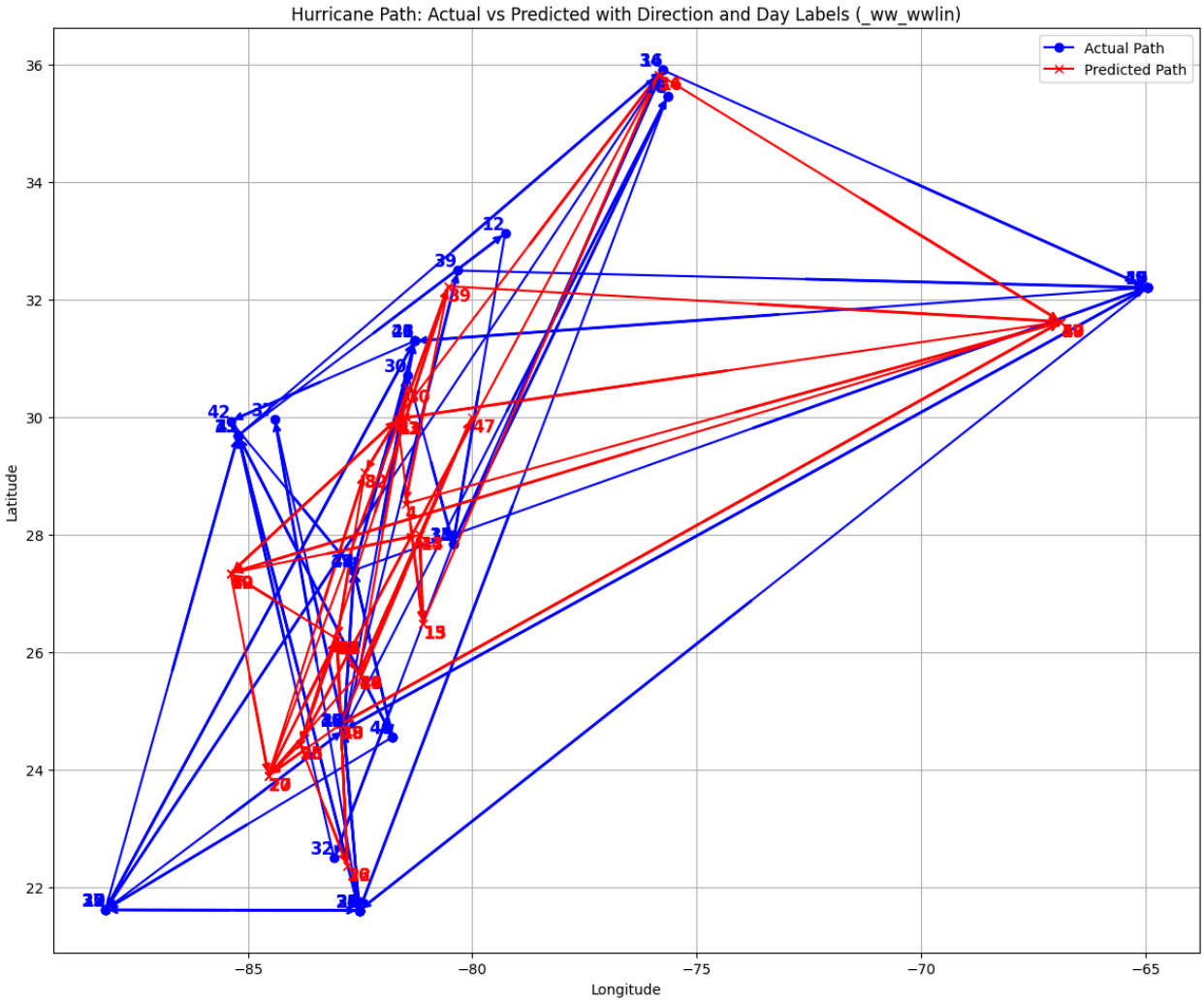
```
for layer in layer_types:
    df = combined_layers[layer].copy()

    if layer == '_5day_lin':
        df['coords'] = df['geometry'].apply(lambda x: x.coords[0] if x is not
        df['LAT'] = df['coords'].apply(lambda x: x[1])
        df['LON'] = df['coords'].apply(lambda x: x[0])
        features = ['LAT', 'LON', 'FCSTPRD']
        target = ['LAT', 'LON']

    elif layer == '_5day_pgn':
        if df.crs is None:
            raise ValueError("GeoDataFrame has no CRS defined")
        projected_crs = 'EPSG:32616'
        df_proj = df.to_crs(projected_crs)
        df_proj['centroid'] = df_proj.geometry.centroid
        df['centroid'] = df_proj['centroid'].to_crs(df.crs)
        df['LAT'] = df['centroid'].y
        df['LON'] = df['centroid'].x
        features = ['LAT', 'LON', 'FCSTPRD']
        target = ['LAT', 'LON']

    elif layer == '_5day_pts':
        features = ['LAT', 'LON', 'GUST', 'MAXWIND', 'MSLP', 'TAU', 'TCDIR', '
        target = ['LAT', 'LON']

    elif layer == '_ww_wwlin':
        df['coords'] = df['geometry'].apply(lambda x: x.coords[0] if x is not
        df['LAT'] = df['coords'].apply(lambda x: x[1])
        df['LON'] = df['coords'].apply(lambda x: x[0])
        features = ['LAT', 'LON', 'FCSTPRD']
        target = ['LAT', 'LON']

    # Remove rows with missing values
    df = df.dropna(subset=features + target)

    # Prepare X and y for prediction (predict next point)
    X = df[features][:-1]
    y = df[target][1:]

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

    # Train the model
    rf = RandomForestRegressor(n_estimators=100, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)

    # Prepare DataFrames for plotting
    actual_path = y_test.reset_index(drop=True)
    predicted_path = pd.DataFrame(y_pred, columns=['LAT', 'LON'])
```

```
    # Plot with arrows for directionality
    plot_paths_on_map_with_arrows_and_labels(actual_path, predicted_path, laye
```




Hurricane Path: Actual vs Predicted with Direction and Day Labels (_5day_pgn)
Actual Path
Predicted Path
Latitude
Longitude
42
40
38
36
34
32
−77.5
−75.0
−72.5
−70.0
−67.5
−65.0
−62.5
−60.0


Hurricane Path: Actual vs Predicted with Direction and Day Labels (_5day_pts)
Actual Path
Predicted Path
Latitude
Longitude
40
35
30
25
20
−85
−80
−75
−70
−65
−60



2.2 RNN

In [23]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import mean_absolute_error

layer_types = ['_5day_lin', '_5day_pgn', '_5day_pts', '_ww_wwlin']
rnn_results = []

for layer in layer_types:
    df = combined_layers[layer].copy()
    # Feature engineering (same as before)
    if layer == '_5day_lin' or layer == '_ww_wwlin':
        df['coords'] = df['geometry'].apply(lambda x: x.coords[0] if x is not
        df['LAT'] = df['coords'].apply(lambda x: x[1])
        df['LON'] = df['coords'].apply(lambda x: x[0])
```

```
        features = ['LAT', 'LON', 'FCSTPRD']
    elif layer == '_5day_pgn':
        if df.crs is None:
            raise ValueError("GeoDataFrame has no CRS defined")
        projected_crs = 'EPSG:32616'
        df_proj = df.to_crs(projected_crs)
        df_proj['centroid'] = df_proj.geometry.centroid
        df['centroid'] = df_proj['centroid'].to_crs(df.crs)
        df['LAT'] = df['centroid'].y
        df['LON'] = df['centroid'].x
        features = ['LAT', 'LON', 'FCSTPRD']
    elif layer == '_5day_pts':
        features = ['LAT', 'LON', 'GUST', 'MAXWIND', 'MSLP', 'TAU', 'TCDIR', '

    df = df.dropna(subset=features)
    sequence_length = 5
    X, y = [], []
    arr = df[features].values
    target = df[['LAT', 'LON']].values
    for i in range(len(arr) - sequence_length):
        X.append(arr[i:i+sequence_length])
        y.append(target[i+sequence_length])
    X = np.array(X)
    y = np.array(y)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

    for units in [32, 64]:
        model = Sequential()
        model.add(SimpleRNN(units, input_shape=(X_train.shape[1], X_train.shap
        model.add(Dropout(0.2))
        model.add(Dense(16, activation='relu'))
        model.add(Dense(2))
        model.compile(optimizer=Adam(0.001), loss='mae')
        model.fit(X_train, y_train, epochs=30, batch_size=16, verbose=0)
        y_pred = model.predict(X_test)
        mae = mean_absolute_error(y_test, y_pred)
        accuracy = 1 - mae / (y_test.max() - y_test.min())
        rnn_results.append({
            'Layer': layer,
            'Model': 'Simple RNN',
            'Hyperparameters': f'units={units}',
            'Accuracy': round(accuracy, 4),
            'MAE': round(mae, 4)
        })

rnn_results_df = pd.DataFrame(rnn_results)
print(rnn_results_df[['Layer', 'Model', 'Hyperparameters', 'Accuracy', 'MAE']]
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 221ms/step
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
```
1/1 ━━━━━━━━━━━━━━━━━━━━ **1s** 591ms/step
```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
```
1/1 ━━━━━━━━━━━━━━━━━━━━ **0s** 304ms/step
```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
```
1/1 ━━━━━━━━━━━━━━━━━━━━ **0s** 193ms/step
```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.mak
e_predict_function.<locals>.one_step_on_data_distributed at 0x7f8b4f51d760> tri
ggered tf.function retracing. Tracing is expensive and the excessive number of
tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) pa
ssing tensors with different shapes, (3) passing Python objects instead of tens
ors. For (1), please define your @tf.function outside of the loop. For (2), @t
f.function has reduce_retracing=True option that can avoid unnecessary retracin
g. For (3), please refer to https://www.tensorflow.org/guide/function#controlli
ng_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  mo
re details.
```
1/4 ━━━━━━━━━━━━━━━━━━━━ **0s** 230ms/step
```
WARNING:tensorflow:6 out of the last 8 calls to <function TensorFlowTrainer.mak
e_predict_function.<locals>.one_step_on_data_distributed at 0x7f8b4f51d760> tri
ggered tf.function retracing. Tracing is expensive and the excessive number of
tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) pa
ssing tensors with different shapes, (3) passing Python objects instead of tens
ors. For (1), please define your @tf.function outside of the loop. For (2), @t
f.function has reduce_retracing=True option that can avoid unnecessary retracin
g. For (3), please refer to https://www.tensorflow.org/guide/function#controlli
ng_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  mo
re details.
```
4/4 ━━━━━━━━━━━━━━━━━━━━ **0s** 67ms/step
```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
```
4/4 ━━━━━━━━━━━━━━━━━━━━ **1s** 236ms/step

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 196ms/step
```

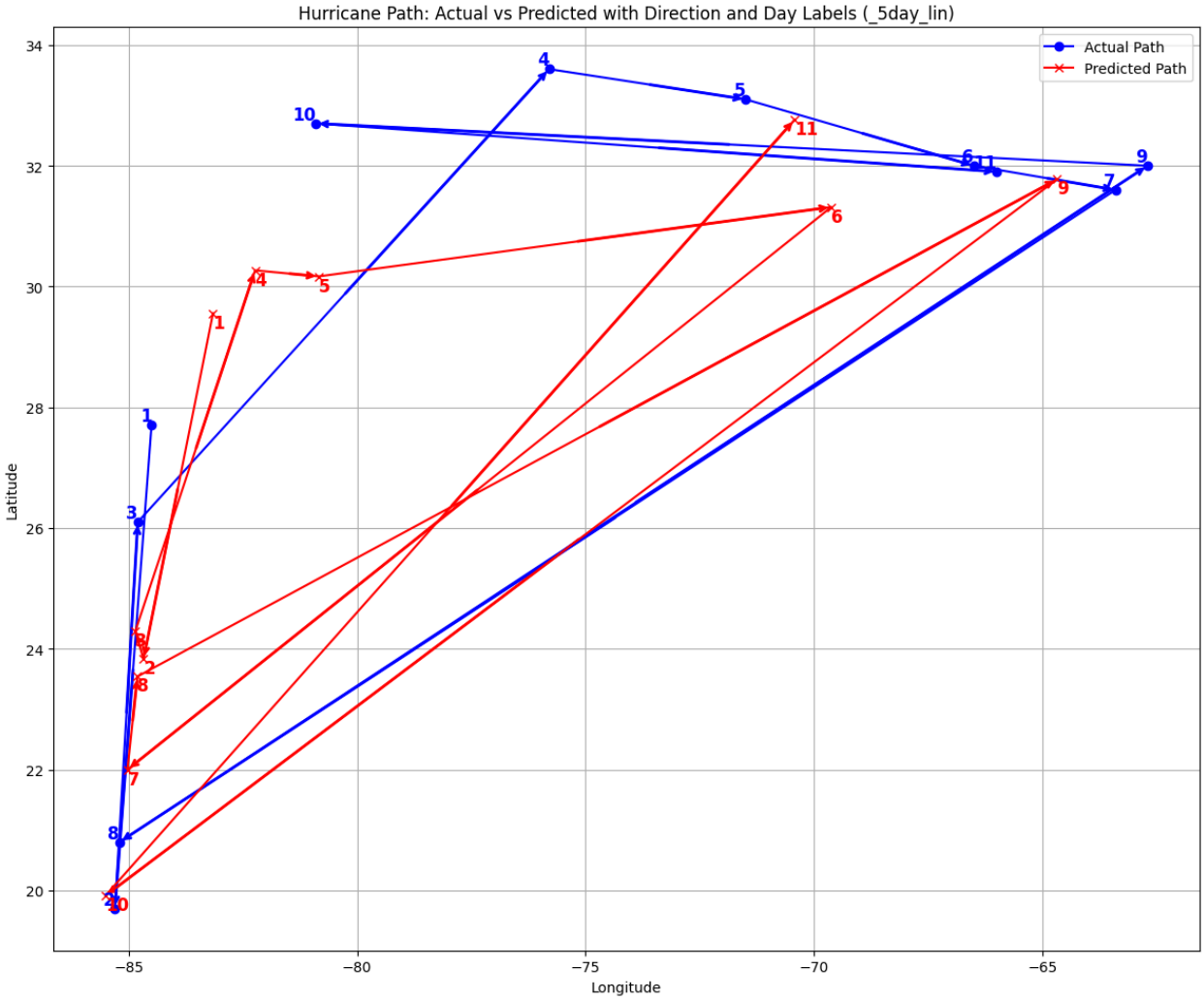
	Layer	Model	Hyperparameters	Accuracy	MAE
0	_5day_lin	Simple RNN	units=32	0.5791	50.2951
1	_5day_lin	Simple RNN	units=64	0.6772	38.5709
2	_5day_pgn	Simple RNN	units=32	0.6635	40.2800
3	_5day_pgn	Simple RNN	units=64	0.6642	40.1957
4	_5day_pts	Simple RNN	units=32	0.9474	6.8666
5	_5day_pts	Simple RNN	units=64	0.9540	6.0097
6	_ww_wwlin	Simple RNN	units=32	0.9728	3.3725
7	_ww_wwlin	Simple RNN	units=64	0.9740	3.2215

In [26]:
```python
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import pandas as pd

# List of layer types to process
layer_types = ['_5day_lin', '_5day_pgn', '_5day_pts', '_ww_wwlin']

def plot_paths_on_map_with_arrows_and_labels(actual_df, predicted_df, layer_na
    plt.figure(figsize=(12, 10))
    # Plot actual path
    plt.plot(actual_df['LON'], actual_df['LAT'], marker='o', color='blue', lab
    # Plot predicted path
    plt.plot(predicted_df['LON'], predicted_df['LAT'], marker='x', color='red'
    # Add arrows at midpoints for actual path
    for i in range(len(actual_df) - 1):
        x_start, y_start = actual_df['LON'].iloc[i], actual_df['LAT'].iloc[i]
        x_end, y_end = actual_df['LON'].iloc[i+1], actual_df['LAT'].iloc[i+1]
        x_mid = (x_start + x_end) / 2
        y_mid = (y_start + y_end) / 2
        plt.annotate('', xy=(x_end, y_end), xytext=(x_mid, y_mid),
                     arrowprops=dict(arrowstyle='->', color='blue', lw=2), zor
    # Add arrows at midpoints for predicted path
    for i in range(len(predicted_df) - 1):
        x_start, y_start = predicted_df['LON'].iloc[i], predicted_df['LAT'].il
        x_end, y_end = predicted_df['LON'].iloc[i+1], predicted_df['LAT'].iloc
        x_mid = (x_start + x_end) / 2
        y_mid = (y_start + y_end) / 2
        plt.annotate('', xy=(x_end, y_end), xytext=(x_mid, y_mid),
                     arrowprops=dict(arrowstyle='->', color='red', lw=2), zord
    # Label each vertex with day number
```

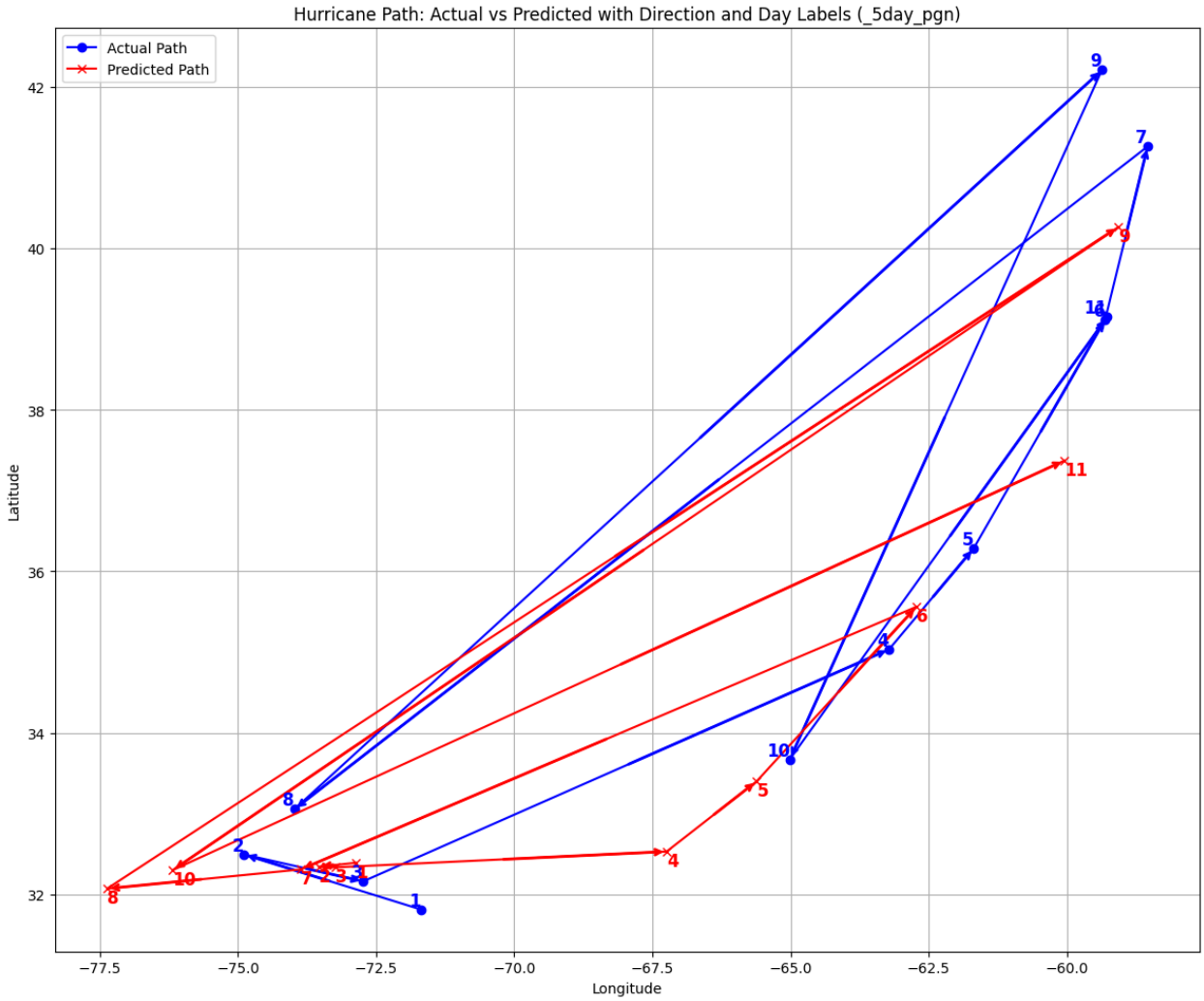
```
    for i, (lat, lon) in enumerate(zip(actual_df['LAT'], actual_df['LON']), st
        plt.text(lon, lat, str(i), color='blue', fontsize=12, fontweight='bold
    for i, (lat, lon) in enumerate(zip(predicted_df['LAT'], predicted_df['LON'
        plt.text(lon, lat, str(i), color='red', fontsize=12, fontweight='bold'
    plt.title(f'Hurricane Path: Actual vs Predicted with Direction and Day Lab
    plt.xlabel('Longitude')
    plt.ylabel('Latitude')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

# Loop over all layers, train model, predict and plot
for layer in layer_types:
    df = combined_layers[layer].copy()

    if layer == '_5day_lin':
        df['coords'] = df['geometry'].apply(lambda x: x.coords[0] if x is not
        df['LAT'] = df['coords'].apply(lambda x: x[1])
        df['LON'] = df['coords'].apply(lambda x: x[0])
        features = ['LAT', 'LON', 'FCSTPRD']
        target = ['LAT', 'LON']

    elif layer == '_5day_pgn':
        if df.crs is None:
            raise ValueError("GeoDataFrame has no CRS defined")
        projected_crs = 'EPSG:32616'
        df_proj = df.to_crs(projected_crs)
        df_proj['centroid'] = df_proj.geometry.centroid
        df['centroid'] = df_proj['centroid'].to_crs(df.crs)
        df['LAT'] = df['centroid'].y
        df['LON'] = df['centroid'].x
        features = ['LAT', 'LON', 'FCSTPRD']
        target = ['LAT', 'LON']

    elif layer == '_5day_pts':
        features = ['LAT', 'LON', 'GUST', 'MAXWIND', 'MSLP', 'TAU', 'TCDIR', '
        target = ['LAT', 'LON']

    elif layer == '_ww_wwlin':
        df['coords'] = df['geometry'].apply(lambda x: x.coords[0] if x is not
        df['LAT'] = df['coords'].apply(lambda x: x[1])
        df['LON'] = df['coords'].apply(lambda x: x[0])
        features = ['LAT', 'LON', 'FCSTPRD']
        target = ['LAT', 'LON']

    # Remove rows with missing values
    df = df.dropna(subset=features + target)

    # Prepare X and y for prediction (predict next point)
    X = df[features][:-1]
    y = df[target][1:]
```

```
    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

    # Train the Random Forest model
    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Prepare DataFrames for plotting
    actual_df = y_test.reset_index(drop=True)
    predicted_df = pd.DataFrame(y_pred, columns=['LAT', 'LON'])

    print(f"Plotting for layer: {layer}")
    plot_paths_on_map_with_arrows_and_labels(actual_df, predicted_df, layer)
```

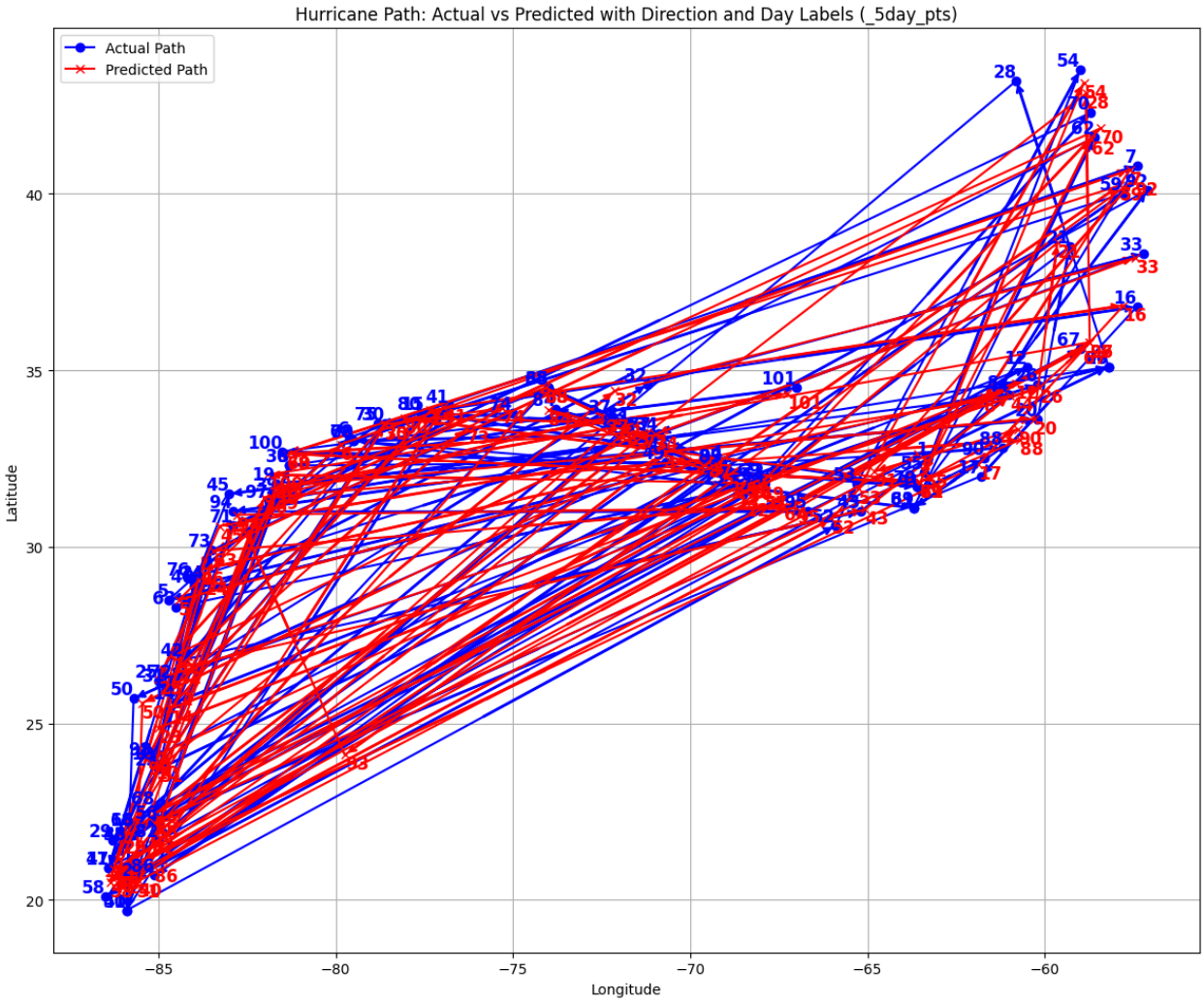
```
Plotting for layer: _5day_lin
```



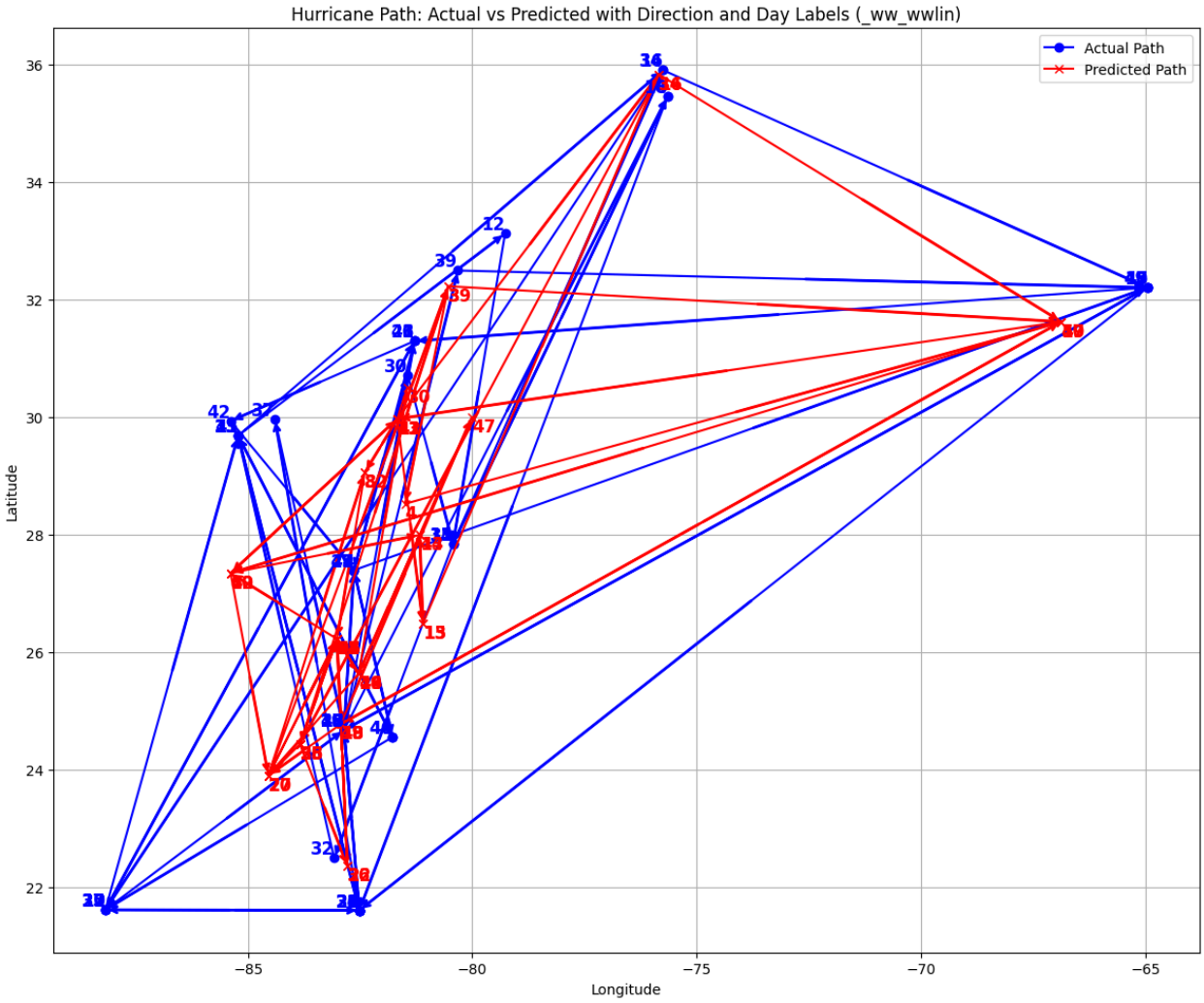

```
Plotting for layer: _5day_pgn
```



Plotting for layer: _5day_pts



Plotting for layer: _ww_wwlin



2.3 LSTM

In [27]:

```
from tensorflow.keras.layers import LSTM

lstm_results = []

for layer in layer_types:
    df = combined_layers[layer].copy()
    if layer == '_5day_lin' or layer == '_ww_wwlin':
        df['coords'] = df['geometry'].apply(lambda x: x.coords[0] if x is not
        df['LAT'] = df['coords'].apply(lambda x: x[1])
        df['LON'] = df['coords'].apply(lambda x: x[0])
        features = ['LAT', 'LON', 'FCSTPRD']
    elif layer == '_5day_pgn':
        if df.crs is None:
            raise ValueError("GeoDataFrame has no CRS defined")
        projected_crs = 'EPSG:32616'
        df_proj = df.to_crs(projected_crs)
        df_proj['centroid'] = df_proj.geometry.centroid
        df['centroid'] = df_proj['centroid'].to_crs(df.crs)
```

```
        df['LAT'] = df['centroid'].y
        df['LON'] = df['centroid'].x
        features = ['LAT', 'LON', 'FCSTPRD']
    elif layer == '_5day_pts':
        features = ['LAT', 'LON', 'GUST', 'MAXWIND', 'MSLP', 'TAU', 'TCDIR', '

    df = df.dropna(subset=features)
    sequence_length = 5
    X, y = [], []
    arr = df[features].values
    target = df[['LAT', 'LON']].values
    for i in range(len(arr) - sequence_length):
        X.append(arr[i:i+sequence_length])
        y.append(target[i+sequence_length])
    X = np.array(X)
    y = np.array(y)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

    for units in [32, 64]:
        model = Sequential()
        model.add(LSTM(units, input_shape=(X_train.shape[1], X_train.shape[2]))
        model.add(Dropout(0.2))
        model.add(Dense(16, activation='relu'))
        model.add(Dense(2))
        model.compile(optimizer=Adam(0.001), loss='mae')
        model.fit(X_train, y_train, epochs=30, batch_size=16, verbose=0)
        y_pred = model.predict(X_test)
        mae = mean_absolute_error(y_test, y_pred)
        accuracy = 1 - mae / (y_test.max() - y_test.min())
        lstm_results.append({
            'Layer': layer,
            'Model': 'LSTM',
            'Hyperparameters': f'units={units}',
            'Accuracy': round(accuracy, 4),
            'MAE': round(mae, 4)
        })

lstm_results_df = pd.DataFrame(lstm_results)
print(lstm_results_df[['Layer', 'Model', 'Hyperparameters', 'Accuracy', 'MAE']
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 283ms/step
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 249ms/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 223ms/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 121ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 121ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 239ms/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 245ms/step
       Layer Model Hyperparameters  Accuracy      MAE
0  _5day_lin  LSTM        units=32    0.5804  50.1413
1  _5day_lin  LSTM        units=64    0.6400  43.0246
2  _5day_pgn  LSTM        units=32    0.5944  48.5458
3  _5day_pgn  LSTM        units=64    0.6104  46.6328
4  _5day_pts  LSTM        units=32    0.9455   7.1221
5  _5day_pts  LSTM        units=64    0.9547   5.9180
6  _ww_wwlin  LSTM        units=32    0.9200   9.9307
7  _ww_wwlin  LSTM        units=64    0.9742   3.2038
```

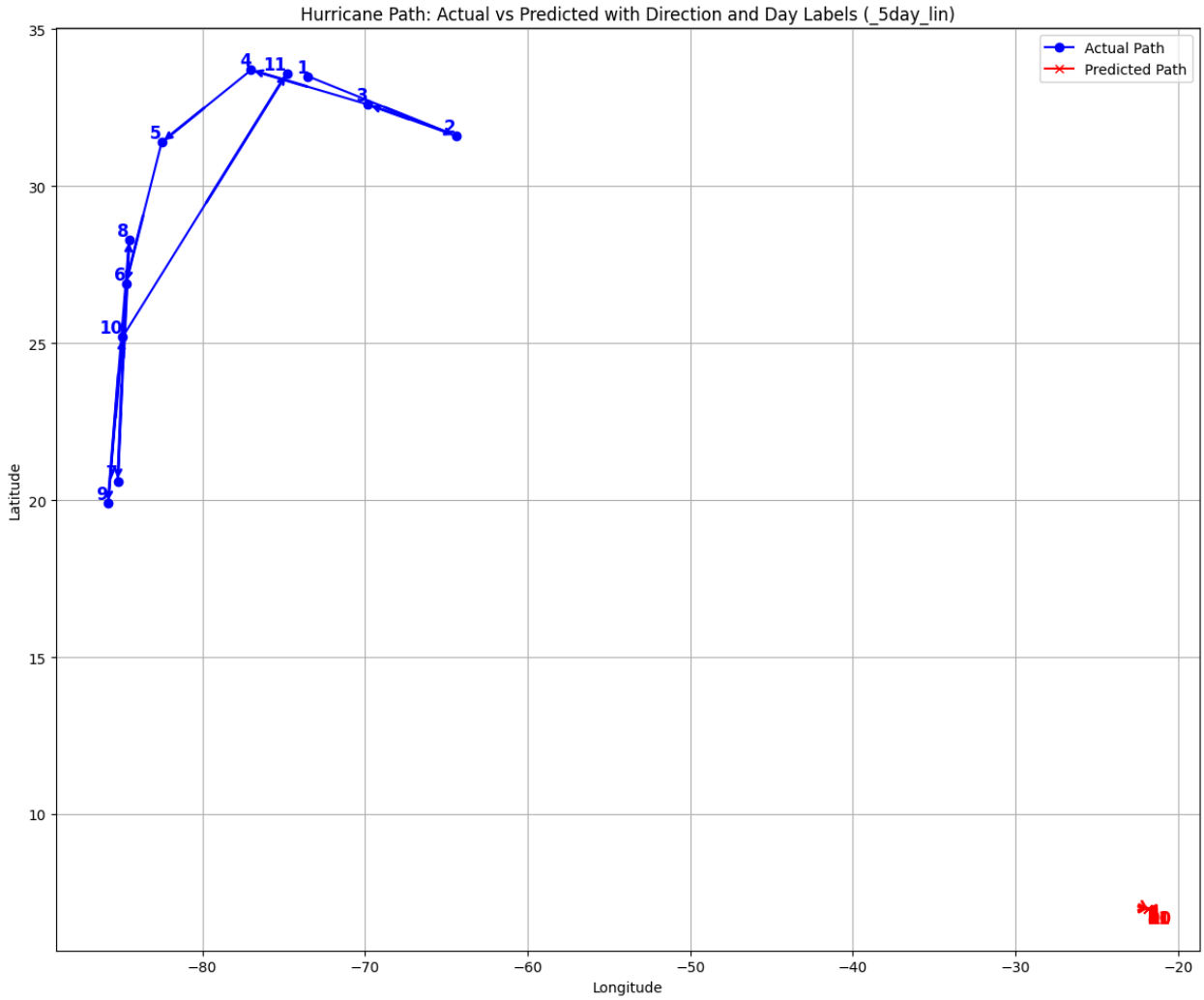
In [28]:

```python
def plot_paths_on_map_with_arrows_and_labels(actual_df, predicted_df, layer_na
    plt.figure(figsize=(12, 10))
    plt.plot(actual_df['LON'], actual_df['LAT'], marker='o', color='blue', lab
    plt.plot(predicted_df['LON'], predicted_df['LAT'], marker='x', color='red'
    # Arrows for actual path
    for i in range(len(actual_df) - 1):
        x_start, y_start = actual_df['LON'].iloc[i], actual_df['LAT'].iloc[i]
        x_end, y_end = actual_df['LON'].iloc[i+1], actual_df['LAT'].iloc[i+1]
        x_mid = (x_start + x_end) / 2
        y_mid = (y_start + y_end) / 2
        plt.annotate('', xy=(x_end, y_end), xytext=(x_mid, y_mid),
                     arrowprops=dict(arrowstyle='->', color='blue', lw=2), zor
    # Arrows for predicted path
    for i in range(len(predicted_df) - 1):
        x_start, y_start = predicted_df['LON'].iloc[i], predicted_df['LAT'].il
        x_end, y_end = predicted_df['LON'].iloc[i+1], predicted_df['LAT'].iloc
```

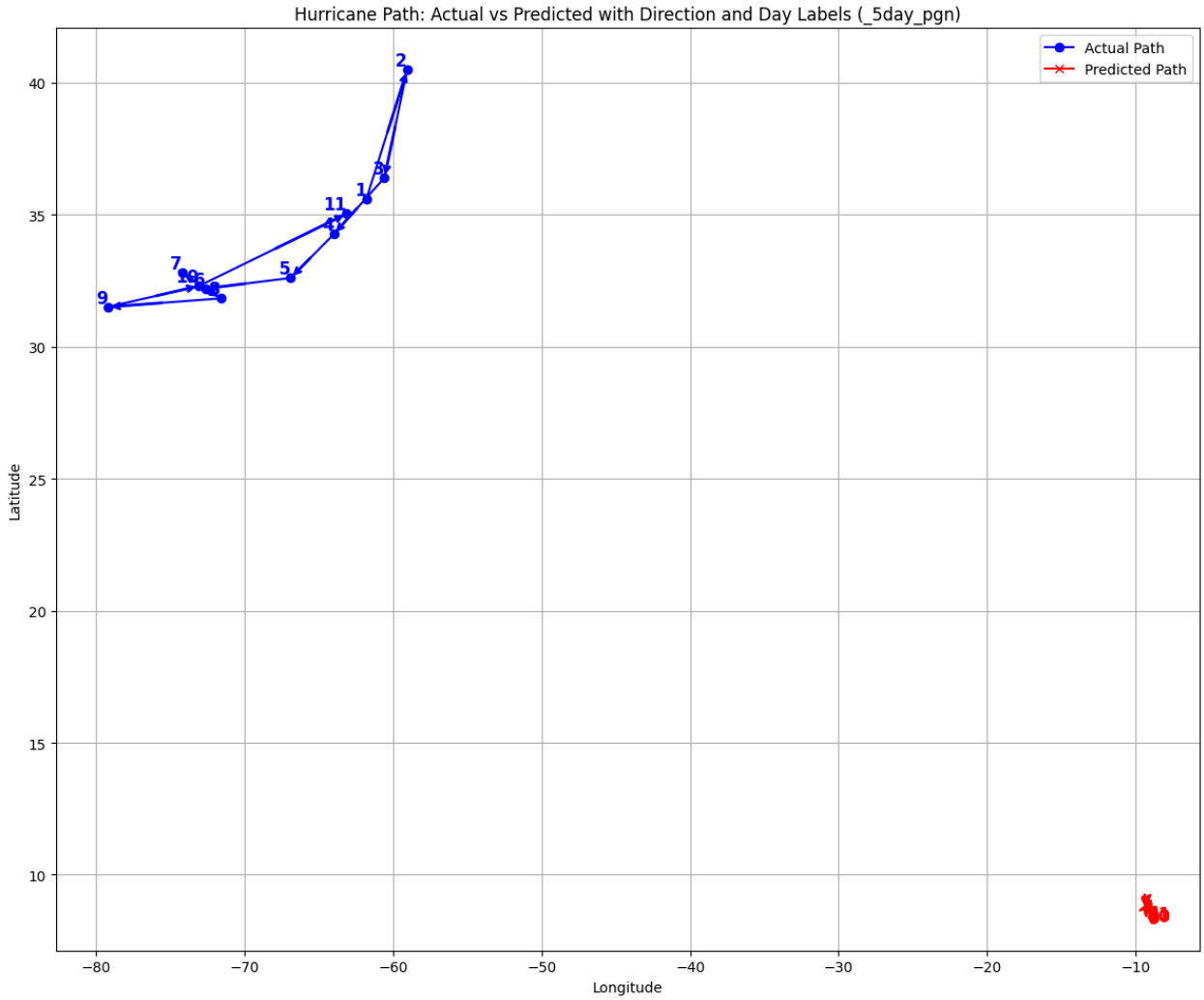
```
        x_mid = (x_start + x_end) / 2
        y_mid = (y_start + y_end) / 2
        plt.annotate('', xy=(x_end, y_end), xytext=(x_mid, y_mid),
                     arrowprops=dict(arrowstyle='->', color='red', lw=2), zord
    # Day number labels
    for i, (lat, lon) in enumerate(zip(actual_df['LAT'], actual_df['LON']), st
        plt.text(lon, lat, str(i), color='blue', fontsize=12, fontweight='bold
    for i, (lat, lon) in enumerate(zip(predicted_df['LAT'], predicted_df['LON'
        plt.text(lon, lat, str(i), color='red', fontsize=12, fontweight='bold'
    plt.title(f'Hurricane Path: Actual vs Predicted with Direction and Day Lab
    plt.xlabel('Longitude')
    plt.ylabel('Latitude')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

layer_types = ['_5day_lin', '_5day_pgn', '_5day_pts', '_ww_wwlin']

for layer in layer_types:
    df = combined_layers[layer].copy()

    # Feature engineering
    if layer == '_5day_lin':
        df['coords'] = df['geometry'].apply(lambda x: x.coords[0] if x is not
        df['LAT'] = df['coords'].apply(lambda x: x[1])
        df['LON'] = df['coords'].apply(lambda x: x[0])
        features = ['LAT', 'LON', 'FCSTPRD']
    elif layer == '_5day_pgn':
        if df.crs is None:
            raise ValueError("GeoDataFrame has no CRS defined")
        projected_crs = 'EPSG:32616'
        df_proj = df.to_crs(projected_crs)
        df_proj['centroid'] = df_proj.geometry.centroid
        df['centroid'] = df_proj['centroid'].to_crs(df.crs)
        df['LAT'] = df['centroid'].y
        df['LON'] = df['centroid'].x
        features = ['LAT', 'LON', 'FCSTPRD']
    elif layer == '_5day_pts':
        features = ['LAT', 'LON', 'GUST', 'MAXWIND', 'MSLP', 'TAU', 'TCDIR', '
    elif layer == '_ww_wwlin':
        df['coords'] = df['geometry'].apply(lambda x: x.coords[0] if x is not
        df['LAT'] = df['coords'].apply(lambda x: x[1])
        df['LON'] = df['coords'].apply(lambda x: x[0])
        features = ['LAT', 'LON', 'FCSTPRD']

    df = df.dropna(subset=features)
    sequence_length = 5
    X, y = [], []
    arr = df[features].values
    target = df[['LAT', 'LON']].values
    for i in range(len(arr) - sequence_length):
        X.append(arr[i:i+sequence_length])
```

```
        y.append(target[i+sequence_length])
    X = np.array(X)
    y = np.array(y)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

    # LSTM Model
    model = Sequential()
    model.add(LSTM(64, input_shape=(X_train.shape[1], X_train.shape[2]), retur
    model.add(Dropout(0.2))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(2))
    model.compile(optimizer=Adam(0.001), loss='mae')
    model.fit(X_train, y_train, epochs=30, batch_size=16, verbose=0)
    y_pred = model.predict(X_test)

    # Prepare DataFrames for plotting
    actual_df = pd.DataFrame(y_test, columns=['LAT', 'LON'])
    predicted_df = pd.DataFrame(y_pred, columns=['LAT', 'LON'])

    print(f"Plotting for layer: {layer}")
    plot_paths_on_map_with_arrows_and_labels(actual_df, predicted_df, layer)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 256ms/step
Plotting for layer: _5day_lin
```



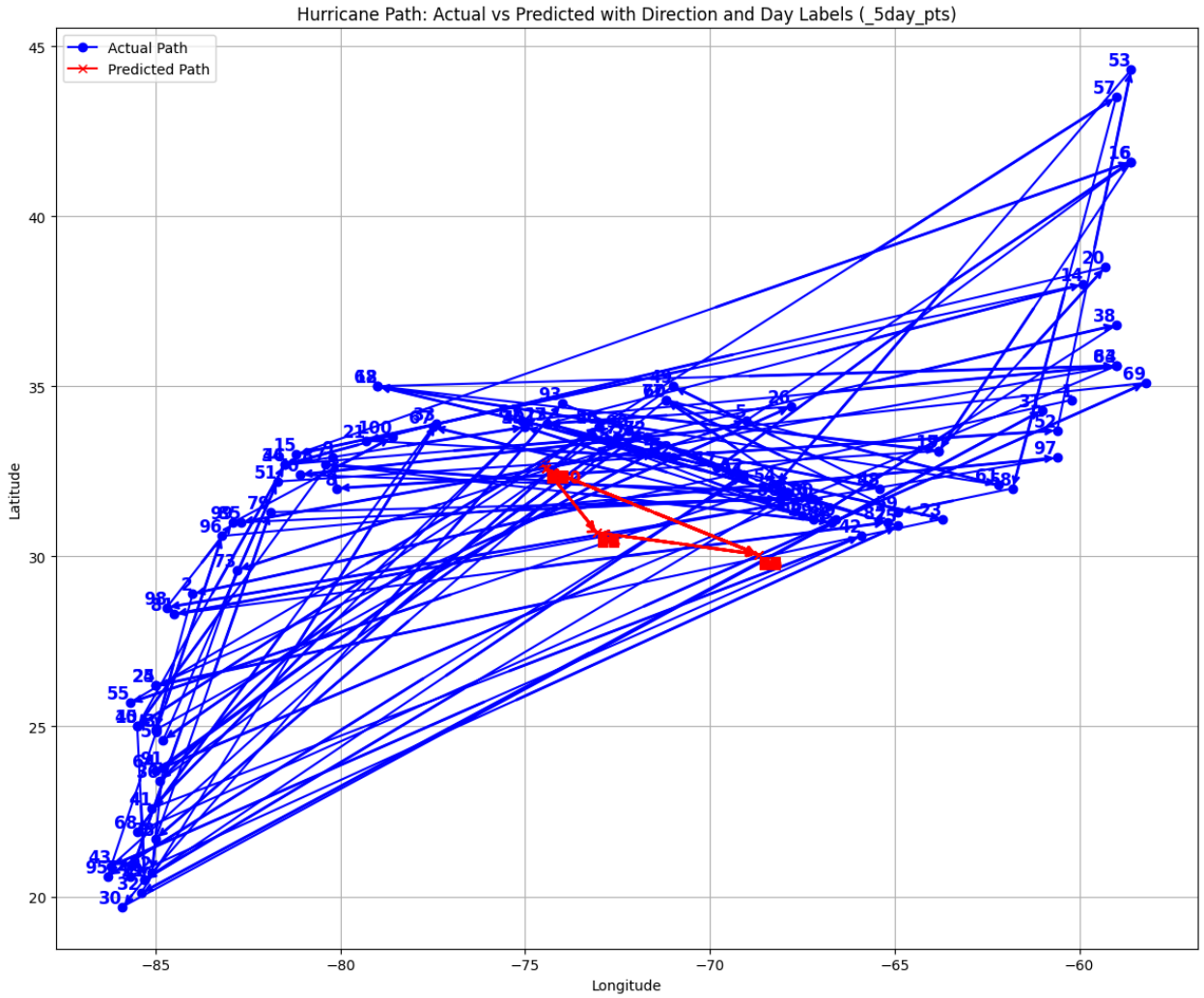

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 363ms/step
Plotting for layer: _5day_pgn
```



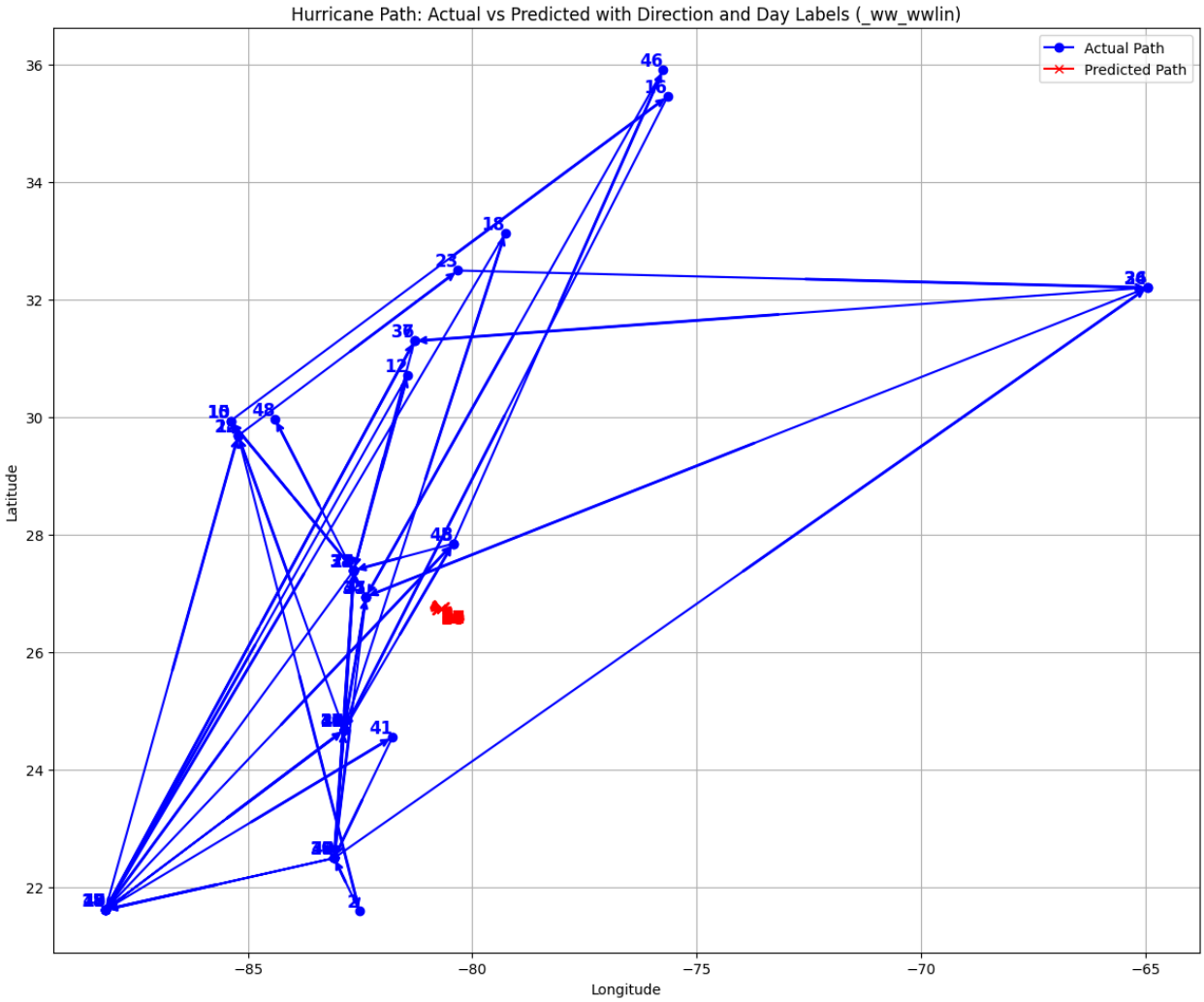

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
```

4/4 ━━━━━━━━━━━━━━━━━━━━ **1s** 191ms/step

Plotting for layer: _5day_pts




```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 239ms/step
Plotting for layer: _ww_wwlin
```



Obeservation: RF significantly outperforms RNN, LSTM in linear, polygon layers which both perform almost equal

Conclusion: Noted its small data, RF performs better

3. **Forecasting IDALIA** (1-step ahead Forecast)

Based on previous hurricane data (not IDALIA) (refer report)

In [30]:
```
import pandas as pd
import numpy as np

def parse_hurdat2_custom(file_path):
    storms = []
    with open(file_path, 'r') as f:
        lines = f.readlines()
    i = 0
    while i < len(lines):
```

```
        line = lines[i].strip()
        # Detect header line (storm id, name, entry count)
        if line and (line.startswith('AL') or line.startswith('EP') or line.st
            header = [x.strip() for x in line.split(',')]
            storm_id = header[0]
            storm_name = header[1]
            num_entries = int(header[2])
            storm_data = []
            for j in range(i+1, i+1+num_entries):
                data_line = [x.strip() for x in lines[j].strip().split(',')]
                storm_data.append(data_line)
            df_storm = pd.DataFrame(storm_data, columns=[
                'Date', 'Time', 'Record Identifier', 'Status', 'Latitude', 'Lo
                'Max Wind', 'Min Pressure', '34kt NE', '34kt SE', '34kt SW', '
                '50kt NE', '50kt SE', '50kt SW', '50kt NW', '64kt NE', '64kt S
            ])
            df_storm['Storm ID'] = storm_id
            df_storm['Storm Name'] = storm_name
            storms.append(df_storm)
            i += num_entries + 1
        else:
            i += 1
    df_all = pd.concat(storms, ignore_index=True)

    # Clean and convert columns
    df_all['Date'] = pd.to_datetime(df_all['Date'], format='%Y%m%d')

    def convert_lat(lat_str):
        lat_str = lat_str.replace(' ', '')
        if lat_str.endswith('N'):
            return float(lat_str[:-1])
        elif lat_str.endswith('S'):
            return -float(lat_str[:-1])
        return np.nan

    def convert_lon(lon_str):
        lon_str = lon_str.replace(' ', '')
        if lon_str.endswith('W'):
            return -float(lon_str[:-1])
        elif lon_str.endswith('E'):
            return float(lon_str[:-1])
        return np.nan

    df_all['Latitude'] = df_all['Latitude'].apply(convert_lat)
    df_all['Longitude'] = df_all['Longitude'].apply(convert_lon)

    numeric_cols = ['Max Wind', 'Min Pressure', '34kt NE', '34kt SE', '34kt SW
                    '50kt NE', '50kt SE', '50kt SW', '50kt NW', '64kt NE', '64
    for col in numeric_cols:
        df_all[col] = pd.to_numeric(df_all[col], errors='coerce')
        df_all.loc[df_all[col] == -999, col] = np.nan

    return df_all
```

```
# usage:
hurdat2_df = parse_hurdat2_custom('/content/drive/My Drive/Colab Notebooks/hur
```

3.1 RF

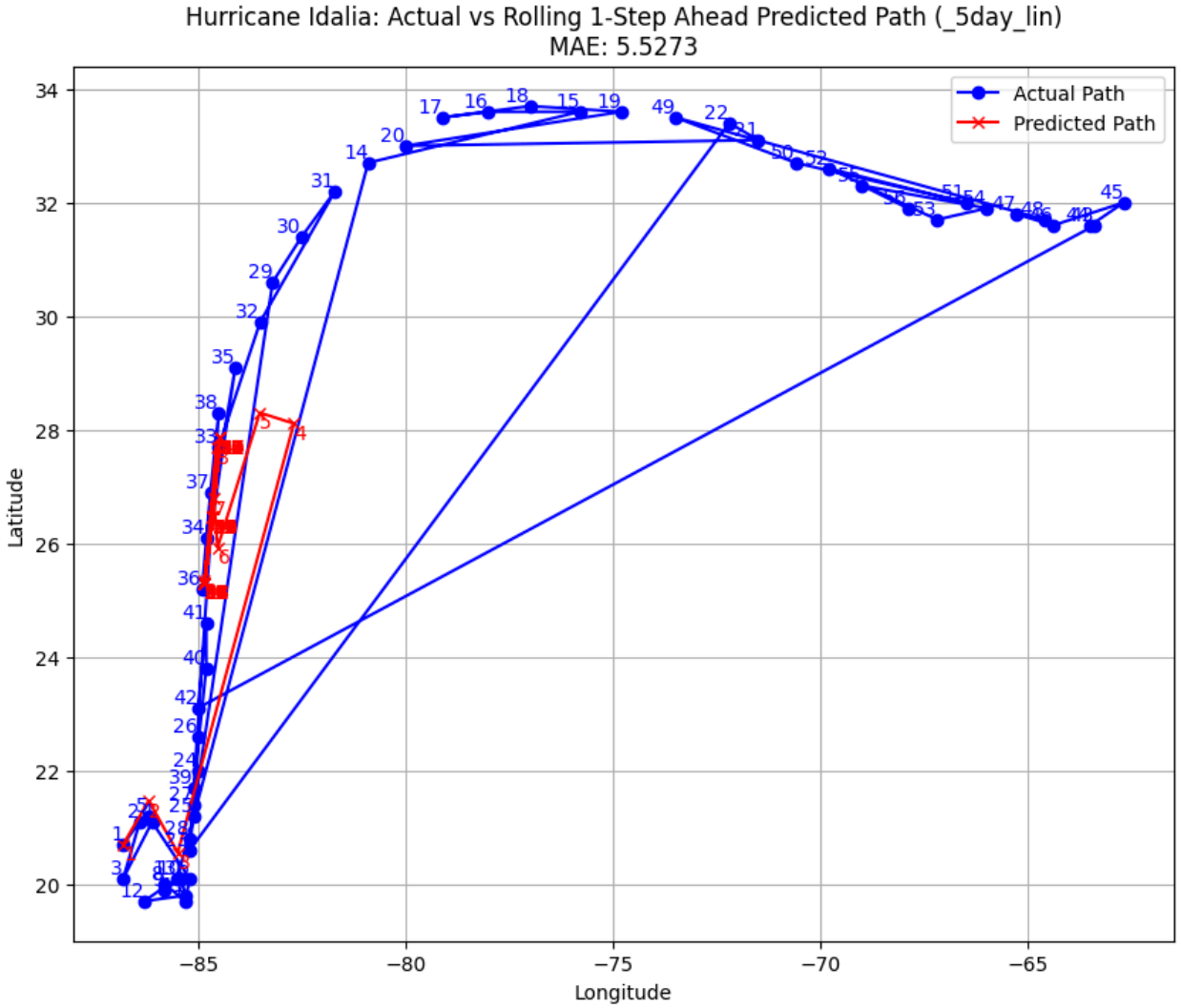
In [47]:
```
# Get Idalia's track from your combined_layers (e.g., '_5day_pts')
idalia_df = combined_layers['_5day_pts'].copy()
idalia_df = idalia_df.sort_values(['TAU'])  # Ensure chronological order

idalia_features = ['LAT', 'LON', 'MAXWIND', 'MSLP']
idalia_track = idalia_df[idalia_features].values

predicted_path = []
current_state = idalia_track[0].copy()
predicted_path.append(current_state[:2])

for i in range(1, len(idalia_track)):
    next_pred = rf_model.predict(current_state.reshape(1, -1))[0]
    predicted_path.append(next_pred)
    # For rolling forecast, update with predicted lat/lon and true wind/pressu
    if i < len(idalia_track):
        current_state = np.array([
            next_pred[0], next_pred[1], idalia_track[i][2], idalia_track[i][3]
        ])

predicted_path = np.array(predicted_path)
```

In [49]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split

layer_types = ['_5day_lin', '_5day_pgn', '_5day_pts', '_ww_wwlin']

for layer in layer_types:
    print(f"\nProcessing layer: {layer}")
    df = combined_layers[layer].copy()

    # Feature engineering for each layer
    if layer == '_5day_lin':
        df['coords'] = df['geometry'].apply(lambda x: x.coords[0] if x is not
        df['LAT'] = df['coords'].apply(lambda x: x[1])
        df['LON'] = df['coords'].apply(lambda x: x[0])
        features = ['LAT', 'LON', 'FCSTPRD']
    elif layer == '_5day_pgn':
        if df.crs is None:
            raise ValueError("GeoDataFrame has no CRS defined")
        projected_crs = 'EPSG:32616'
        df_proj = df.to_crs(projected_crs)
```

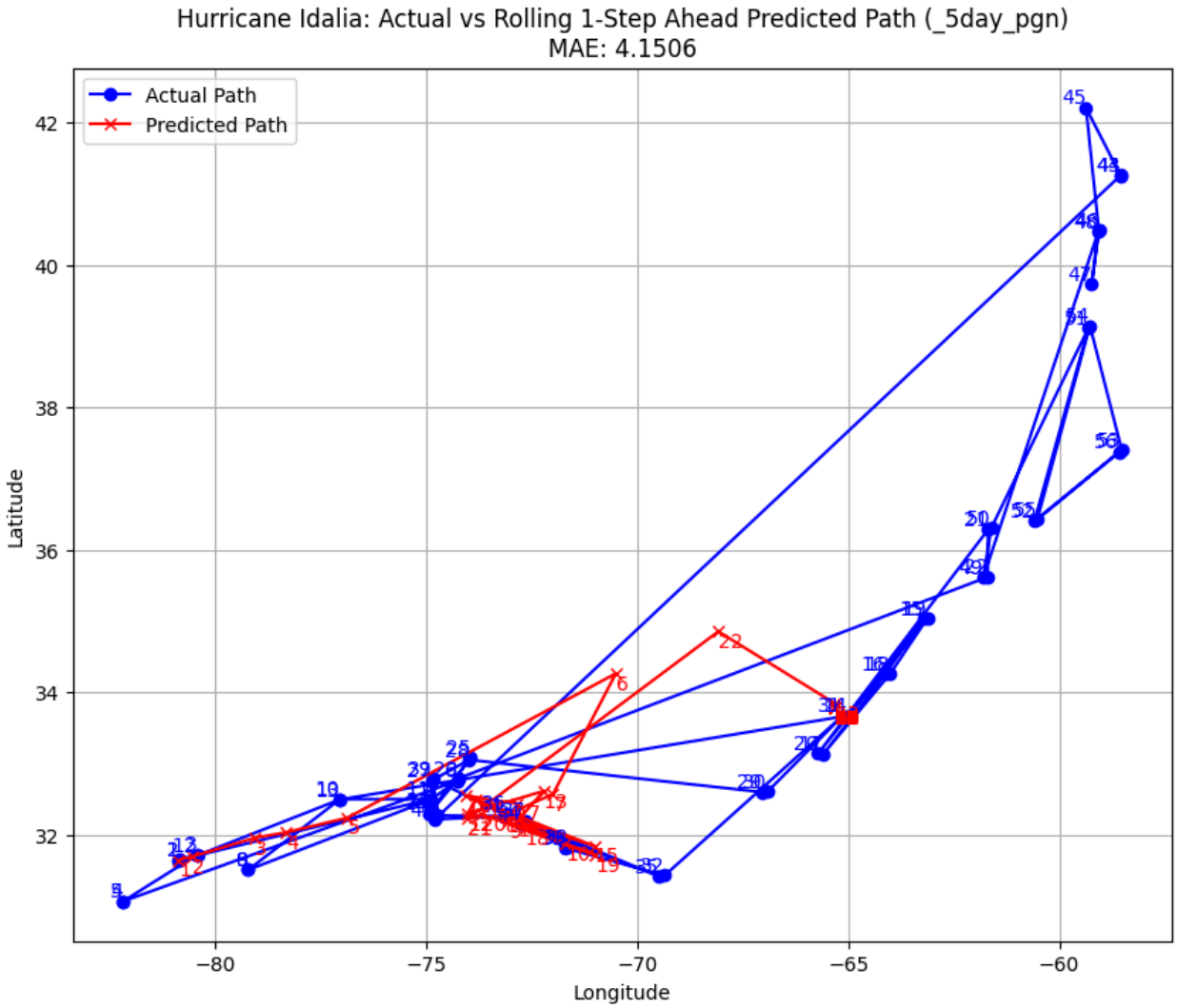
```
        df_proj['centroid'] = df_proj.geometry.centroid
        df['centroid'] = df_proj['centroid'].to_crs(df.crs)
        df['LAT'] = df['centroid'].y
        df['LON'] = df['centroid'].x
        features = ['LAT', 'LON', 'FCSTPRD']
    elif layer == '_5day_pts':
        features = ['LAT', 'LON', 'MAXWIND', 'MSLP']
    elif layer == '_ww_wwlin':
        df['coords'] = df['geometry'].apply(lambda x: x.coords[0] if x is not
        df['LAT'] = df['coords'].apply(lambda x: x[1])
        df['LON'] = df['coords'].apply(lambda x: x[0])
        features = ['LAT', 'LON', 'FCSTPRD']

    # Remove rows with missing values
    df = df.dropna(subset=features)

    # Prepare data for model training (1-step ahead)
    X, y = [], []
    arr = df[features].values
    for i in range(len(arr) - 1):
        X.append(arr[i])
        y.append(arr[i+1][:2])  # Next step's [LAT, LON]
    X = np.array(X)
    y = np.array(y)

    # Train a separate Random Forest model for this layer
    rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
    rf_model.fit(X, y)

    # Prepare Idalia's track for rolling forecast (use the same features)
    idalia_track = df[features].values

    predicted_path = []
    current_state = idalia_track[0].copy()
    predicted_path.append(current_state[:2])

    for i in range(1, len(idalia_track)):
        next_pred = rf_model.predict(current_state.reshape(1, -1))[0]
        predicted_path.append(next_pred)
        if i < len(idalia_track):
            # For 3-feature layers
            if len(features) == 3:
                current_state = np.array([next_pred[0], next_pred[1], idalia_t
            # For 4-feature layers
            elif len(features) == 4:
                current_state = np.array([next_pred[0], next_pred[1], idalia_t

    predicted_path = np.array(predicted_path)

    actual_lats = idalia_track[:,0]
    actual_lons = idalia_track[:,1]
    pred_lats = predicted_path[:,0]
    pred_lons = predicted_path[:,1]
```

```
    # Calculate MAE for [LAT, LON] pairs
    mae = mean_absolute_error(np.column_stack((actual_lats, actual_lons)), pre

    plt.figure(figsize=(10,8))
    plt.plot(actual_lons, actual_lats, 'bo-', label='Actual Path')
    plt.plot(pred_lons, pred_lats, 'rx-', label='Predicted Path')
    for j, (lon, lat) in enumerate(zip(actual_lons, actual_lats), 1):
        plt.text(lon, lat, str(j), color='blue', fontsize=10, ha='right', va='
    for j, (lon, lat) in enumerate(zip(pred_lons, pred_lats), 1):
        plt.text(lon, lat, str(j), color='red', fontsize=10, ha='left', va='to
    plt.xlabel('Longitude')
    plt.ylabel('Latitude')
    plt.legend()
    plt.title(f'Hurricane Idalia: Actual vs Rolling 1-Step Ahead Predicted Pat
    plt.grid(True)
    plt.show()
```

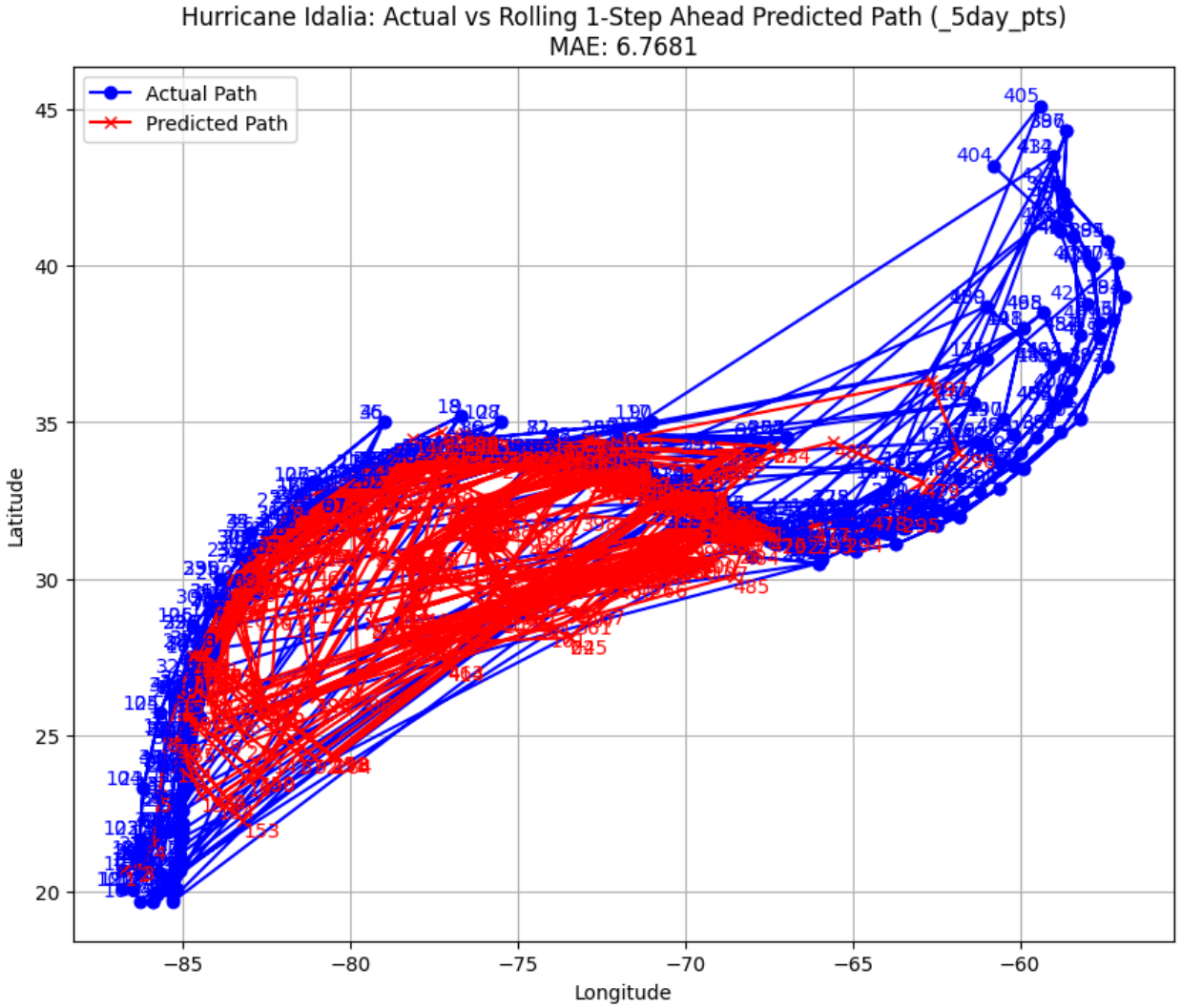
```
Processing layer: _5day_lin
```



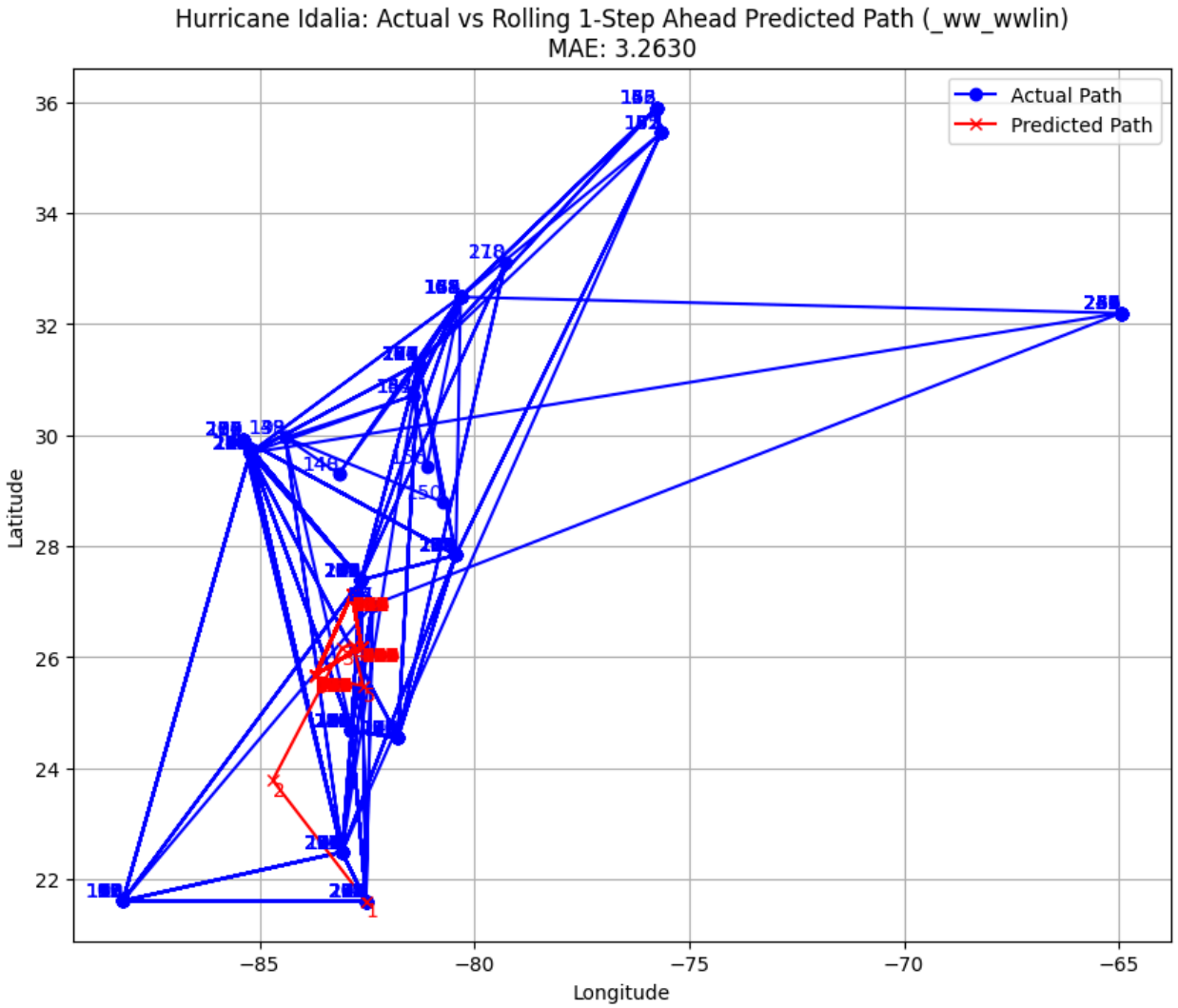

```
Processing layer: _5day_pgn
```




```
Processing layer: _5day_pts
```




```
Processing layer: _ww_wwlin
```



3.2 LSTM

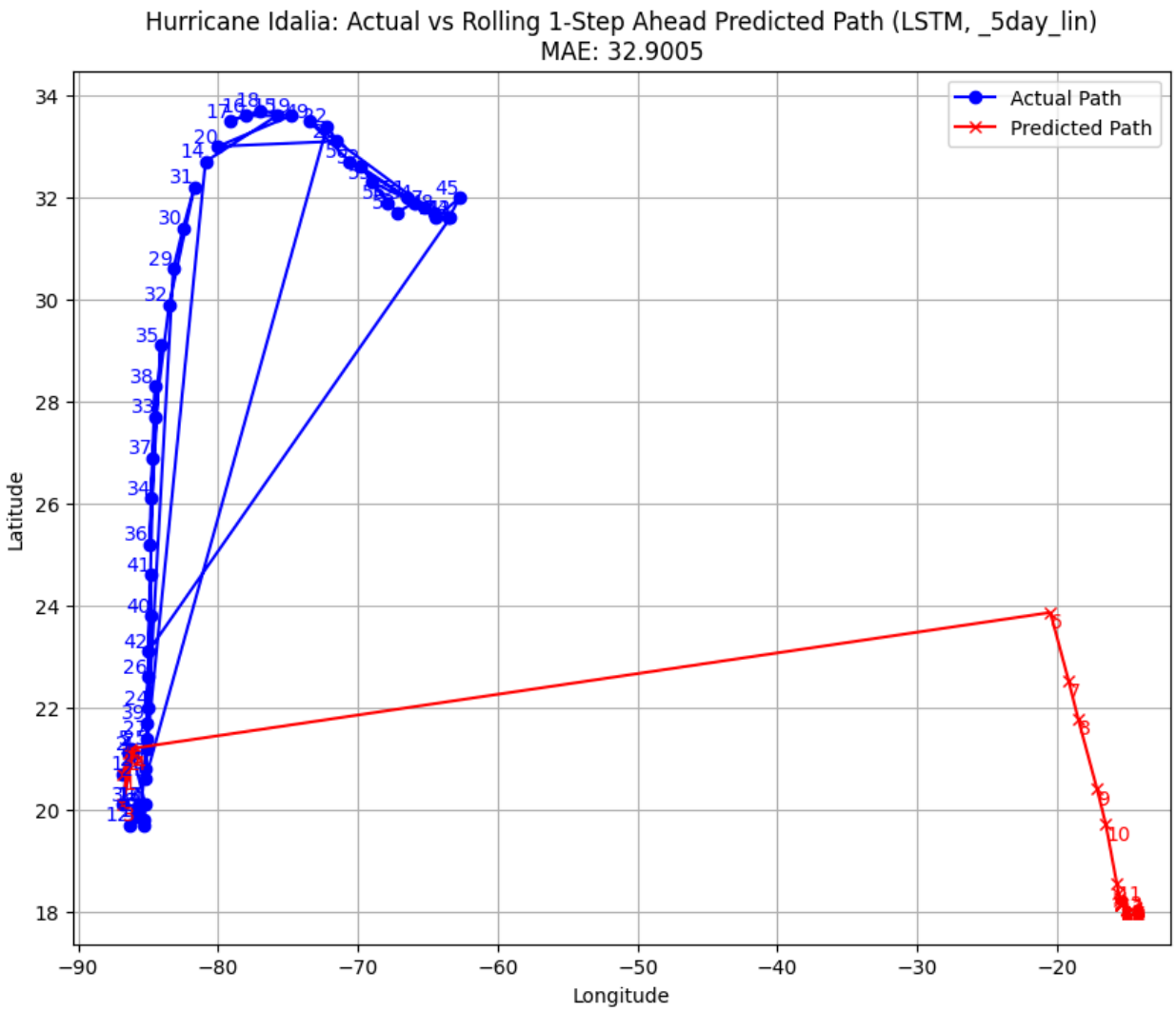
In [50]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import mean_absolute_error

layer_types = ['_5day_lin', '_5day_pgn', '_5day_pts', '_ww_wwlin']

for layer in layer_types:
    print(f"\nProcessing layer: {layer}")
    df = combined_layers[layer].copy()

    # Feature engineering for each layer
    if layer == '_5day_lin':
        df['coords'] = df['geometry'].apply(lambda x: x.coords[0] if x is not
```

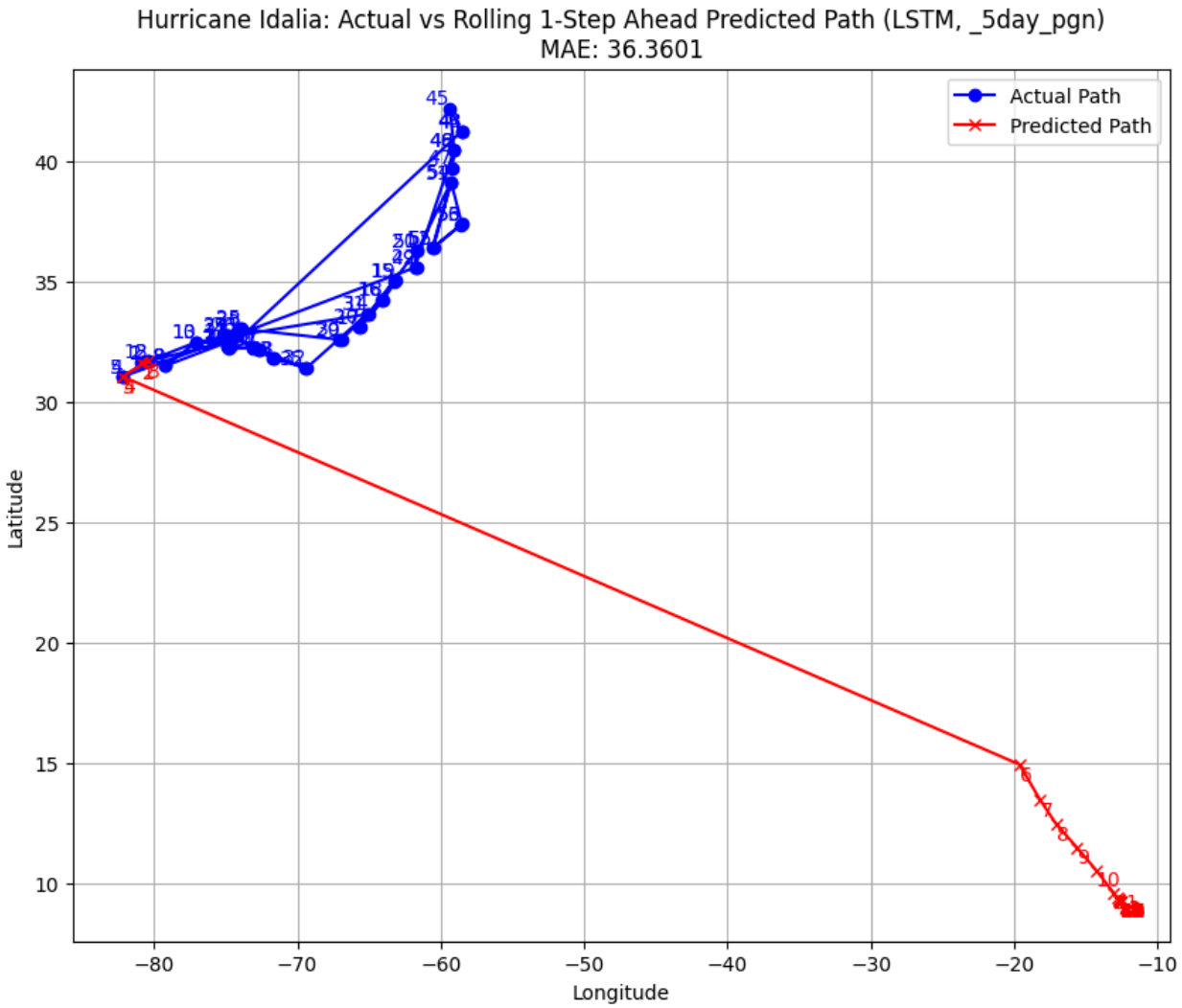
```
        df['LAT'] = df['coords'].apply(lambda x: x[1])
        df['LON'] = df['coords'].apply(lambda x: x[0])
        features = ['LAT', 'LON', 'FCSTPRD']
    elif layer == '_5day_pgn':
        if df.crs is None:
            raise ValueError("GeoDataFrame has no CRS defined")
        projected_crs = 'EPSG:32616'
        df_proj = df.to_crs(projected_crs)
        df_proj['centroid'] = df_proj.geometry.centroid
        df['centroid'] = df_proj['centroid'].to_crs(df.crs)
        df['LAT'] = df['centroid'].y
        df['LON'] = df['centroid'].x
        features = ['LAT', 'LON', 'FCSTPRD']
    elif layer == '_5day_pts':
        features = ['LAT', 'LON', 'MAXWIND', 'MSLP']
    elif layer == '_ww_wwlin':
        df['coords'] = df['geometry'].apply(lambda x: x.coords[0] if x is not
        df['LAT'] = df['coords'].apply(lambda x: x[1])
        df['LON'] = df['coords'].apply(lambda x: x[0])
        features = ['LAT', 'LON', 'FCSTPRD']

    df = df.dropna(subset=features)
    arr = df[features].values
    target = df[['LAT', 'LON']].values

    # Prepare sequences for LSTM
    sequence_length = 5
    X_seq, y_seq = [], []
    for i in range(len(arr) - sequence_length):
        X_seq.append(arr[i:i+sequence_length])
        y_seq.append(target[i+sequence_length])
    X_seq = np.array(X_seq)
    y_seq = np.array(y_seq)

    # Train LSTM model
    model = Sequential()
    model.add(LSTM(64, input_shape=(X_seq.shape[1], X_seq.shape[2]), return_se
    model.add(Dropout(0.2))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(2))
    model.compile(optimizer=Adam(0.001), loss='mae')
    model.fit(X_seq, y_seq, epochs=30, batch_size=16, verbose=0)

    # Rolling 1-step ahead forecast for Idalia
    idalia_track = arr
    predicted_path = []
    # Start with the first sequence
    current_seq = idalia_track[:sequence_length].copy()
    predicted_path.extend(current_seq[:, :2])

    for i in range(sequence_length, len(idalia_track)):
        next_pred = model.predict(current_seq.reshape(1, sequence_length, -1),
        predicted_path.append(next_pred)
```

```
        if i + 1 < len(idalia_track):
            # Use predicted lat/lon, true other features for next step
            if len(features) == 3:
                next_features = [idalia_track[i][2]]
                next_input = np.array([next_pred[0], next_pred[1]] + next_feat
            elif len(features) == 4:
                next_features = [idalia_track[i][2], idalia_track[i][3]]
                next_input = np.array([next_pred[0], next_pred[1]] + next_feat
            current_seq = np.vstack([current_seq[1:], next_input])

    predicted_path = np.array(predicted_path)[:len(idalia_track)]
    actual_lats = idalia_track[:,0]
    actual_lons = idalia_track[:,1]
    pred_lats = predicted_path[:,0]
    pred_lons = predicted_path[:,1]
    mae = mean_absolute_error(np.column_stack((actual_lats, actual_lons)), pre

    # Visualization
    plt.figure(figsize=(10,8))
    plt.plot(actual_lons, actual_lats, 'bo-', label='Actual Path')
    plt.plot(pred_lons, pred_lats, 'rx-', label='Predicted Path')
    for j, (lon, lat) in enumerate(zip(actual_lons, actual_lats), 1):
        plt.text(lon, lat, str(j), color='blue', fontsize=10, ha='right', va='
    for j, (lon, lat) in enumerate(zip(pred_lons, pred_lats), 1):
        plt.text(lon, lat, str(j), color='red', fontsize=10, ha='left', va='to
    plt.xlabel('Longitude')
    plt.ylabel('Latitude')
    plt.legend()
    plt.title(f'Hurricane Idalia: Actual vs Rolling 1-Step Ahead Predicted Pat
    plt.grid(True)
    plt.show()
```

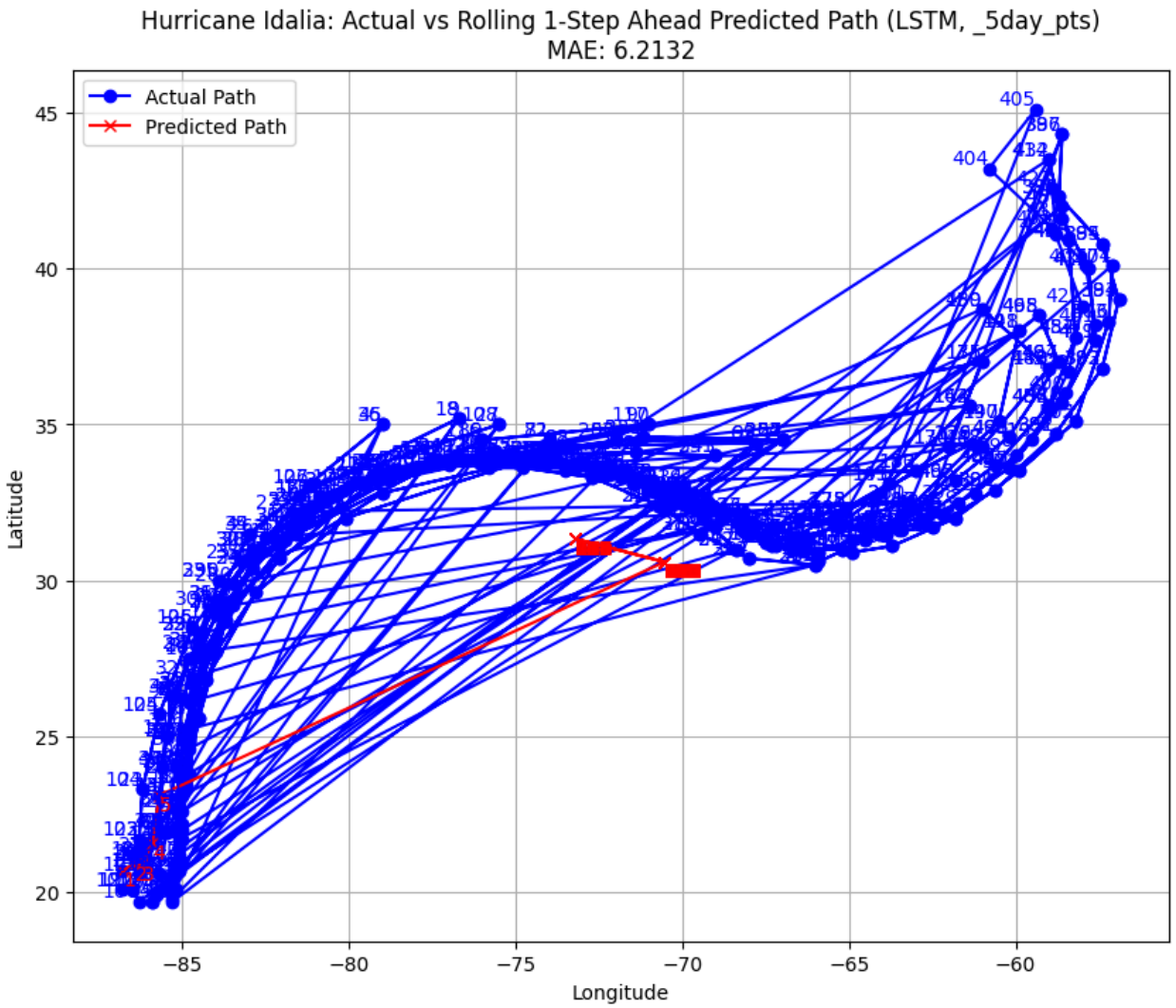
```
Processing layer: _5day_lin
```




```
Processing layer: _5day_pgn
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
```

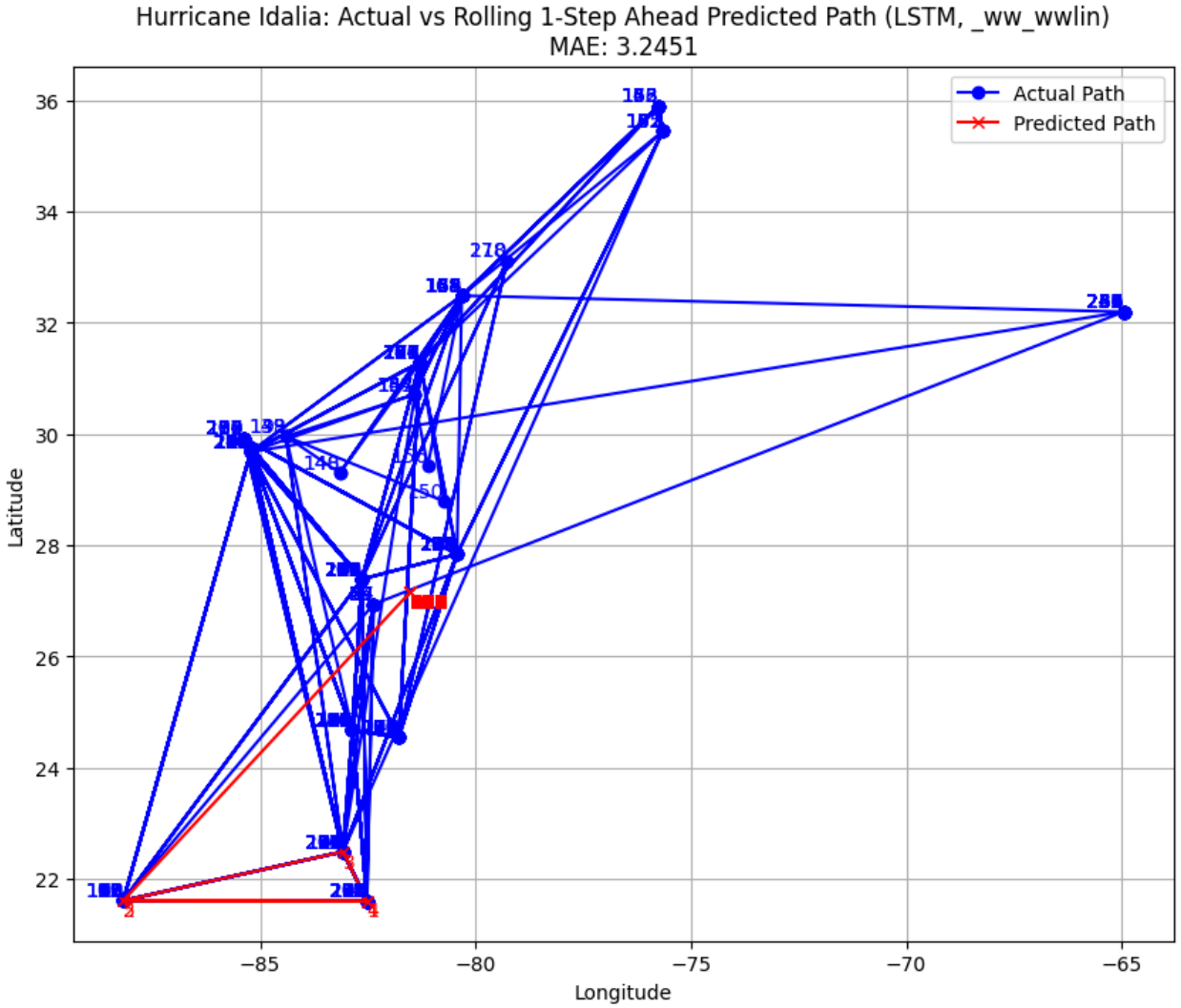



```
Processing layer: _5day_pts
```




```
Processing layer: _ww_wwlin
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
```



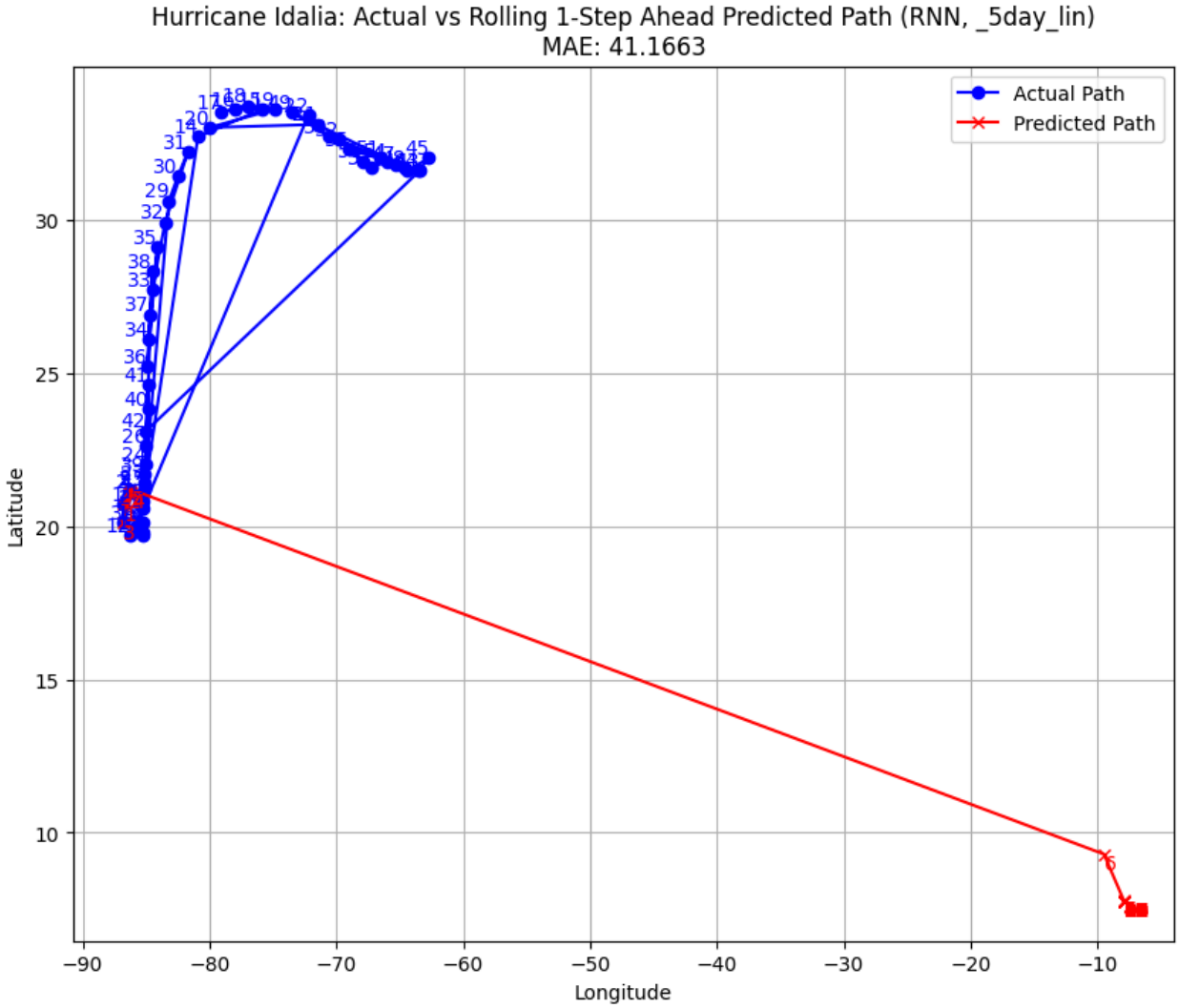
3.3 RNN

In [51]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import mean_absolute_error

layer_types = ['_5day_lin', '_5day_pgn', '_5day_pts', '_ww_wwlin']

for layer in layer_types:
    print(f"\nProcessing layer: {layer}")
    df = combined_layers[layer].copy()

    # Feature engineering for each layer
    if layer == '_5day_lin':
        df['coords'] = df['geometry'].apply(lambda x: x.coords[0] if x is not
```

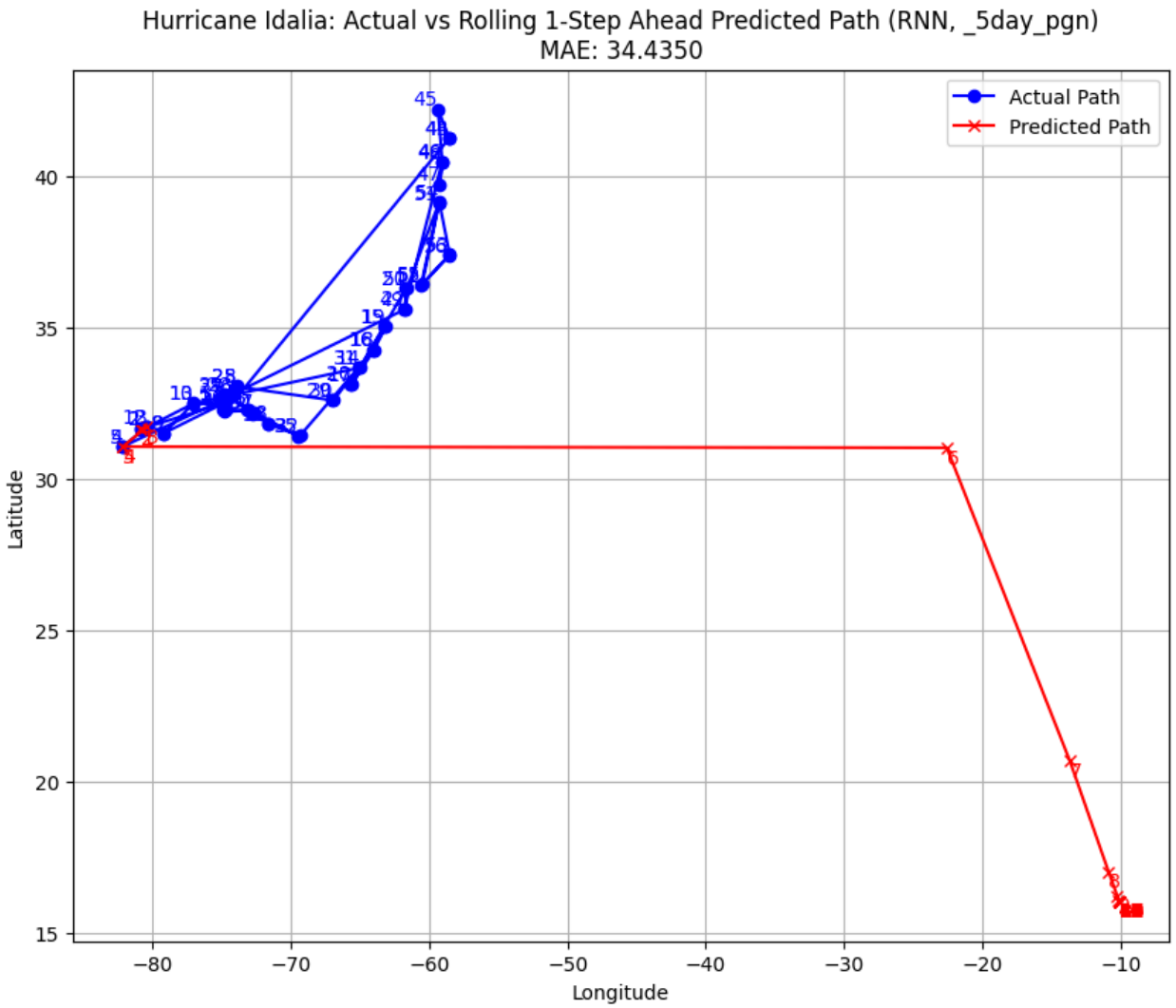
```
        df['LAT'] = df['coords'].apply(lambda x: x[1])
        df['LON'] = df['coords'].apply(lambda x: x[0])
        features = ['LAT', 'LON', 'FCSTPRD']
    elif layer == '_5day_pgn':
        if df.crs is None:
            raise ValueError("GeoDataFrame has no CRS defined")
        projected_crs = 'EPSG:32616'
        df_proj = df.to_crs(projected_crs)
        df_proj['centroid'] = df_proj.geometry.centroid
        df['centroid'] = df_proj['centroid'].to_crs(df.crs)
        df['LAT'] = df['centroid'].y
        df['LON'] = df['centroid'].x
        features = ['LAT', 'LON', 'FCSTPRD']
    elif layer == '_5day_pts':
        features = ['LAT', 'LON', 'MAXWIND', 'MSLP']
    elif layer == '_ww_wwlin':
        df['coords'] = df['geometry'].apply(lambda x: x.coords[0] if x is not
        df['LAT'] = df['coords'].apply(lambda x: x[1])
        df['LON'] = df['coords'].apply(lambda x: x[0])
        features = ['LAT', 'LON', 'FCSTPRD']

    df = df.dropna(subset=features)
    arr = df[features].values
    target = df[['LAT', 'LON']].values

    # Prepare sequences for RNN
    sequence_length = 5
    X_seq, y_seq = [], []
    for i in range(len(arr) - sequence_length):
        X_seq.append(arr[i:i+sequence_length])
        y_seq.append(target[i+sequence_length])
    X_seq = np.array(X_seq)
    y_seq = np.array(y_seq)

    # Train RNN model
    model = Sequential()
    model.add(SimpleRNN(64, input_shape=(X_seq.shape[1], X_seq.shape[2]), retu
    model.add(Dropout(0.2))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(2))
    model.compile(optimizer=Adam(0.001), loss='mae')
    model.fit(X_seq, y_seq, epochs=30, batch_size=16, verbose=0)

    # Rolling 1-step ahead forecast for Idalia
    idalia_track = arr
    predicted_path = []
    current_seq = idalia_track[:sequence_length].copy()
    predicted_path.extend(current_seq[:, :2])

    for i in range(sequence_length, len(idalia_track)):
        next_pred = model.predict(current_seq.reshape(1, sequence_length, -1),
        predicted_path.append(next_pred)
        if i + 1 < len(idalia_track):
```

```
            if len(features) == 3:
                next_features = [idalia_track[i][2]]
                next_input = np.array([next_pred[0], next_pred[1]] + next_feat
            elif len(features) == 4:
                next_features = [idalia_track[i][2], idalia_track[i][3]]
                next_input = np.array([next_pred[0], next_pred[1]] + next_feat
            current_seq = np.vstack([current_seq[1:], next_input])

    predicted_path = np.array(predicted_path)[:len(idalia_track)]
    actual_lats = idalia_track[:,0]
    actual_lons = idalia_track[:,1]
    pred_lats = predicted_path[:,0]
    pred_lons = predicted_path[:,1]
    mae = mean_absolute_error(np.column_stack((actual_lats, actual_lons)), pre

    # Visualization
    plt.figure(figsize=(10,8))
    plt.plot(actual_lons, actual_lats, 'bo-', label='Actual Path')
    plt.plot(pred_lons, pred_lats, 'rx-', label='Predicted Path')
    for j, (lon, lat) in enumerate(zip(actual_lons, actual_lats), 1):
        plt.text(lon, lat, str(j), color='blue', fontsize=10, ha='right', va='
    for j, (lon, lat) in enumerate(zip(pred_lons, pred_lats), 1):
        plt.text(lon, lat, str(j), color='red', fontsize=10, ha='left', va='to
    plt.xlabel('Longitude')
    plt.ylabel('Latitude')
    plt.legend()
    plt.title(f'Hurricane Idalia: Actual vs Rolling 1-Step Ahead Predicted Pat
    plt.grid(True)
    plt.show()
```

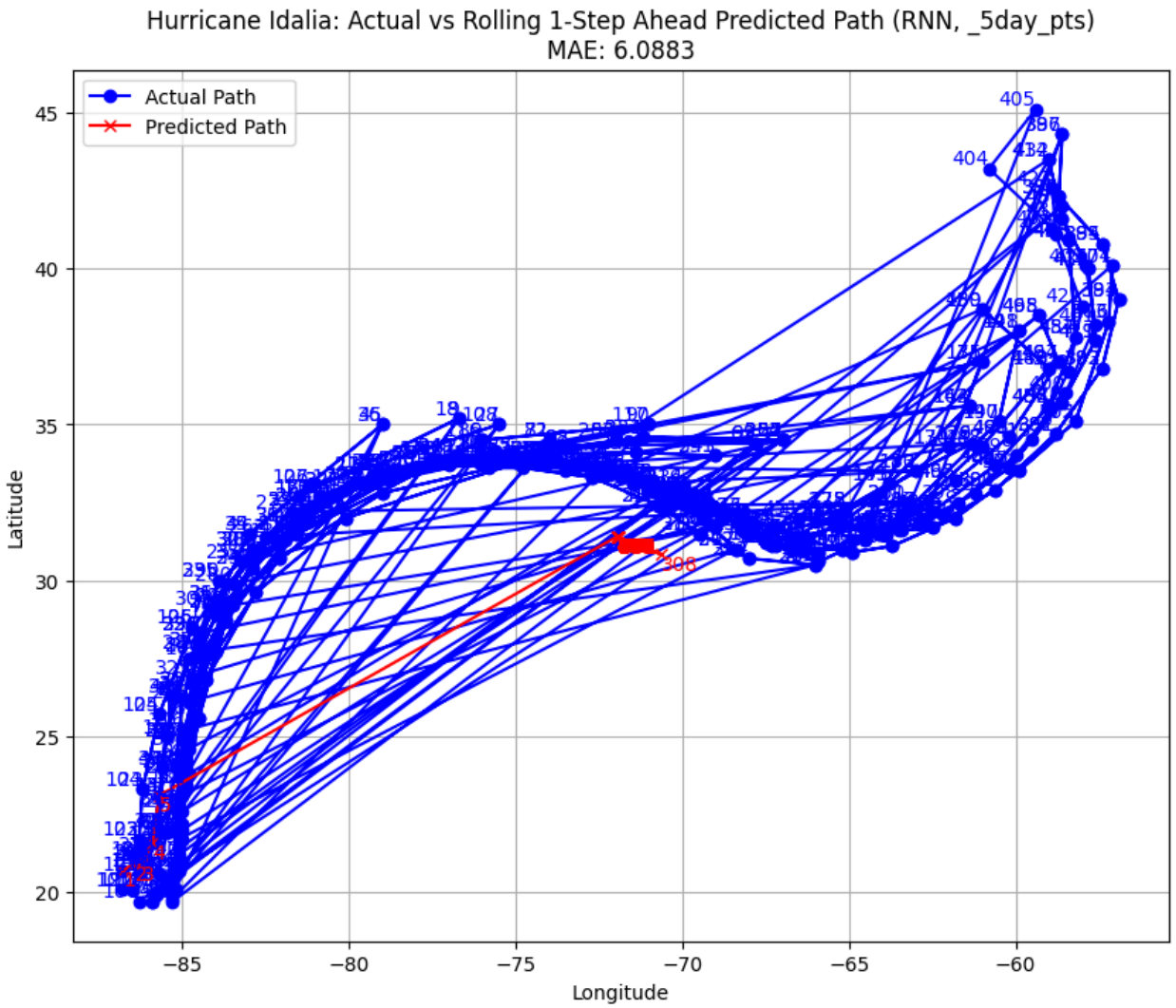
```
Processing layer: _5day_lin
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
```




```
Processing layer: _5day_pgn
```

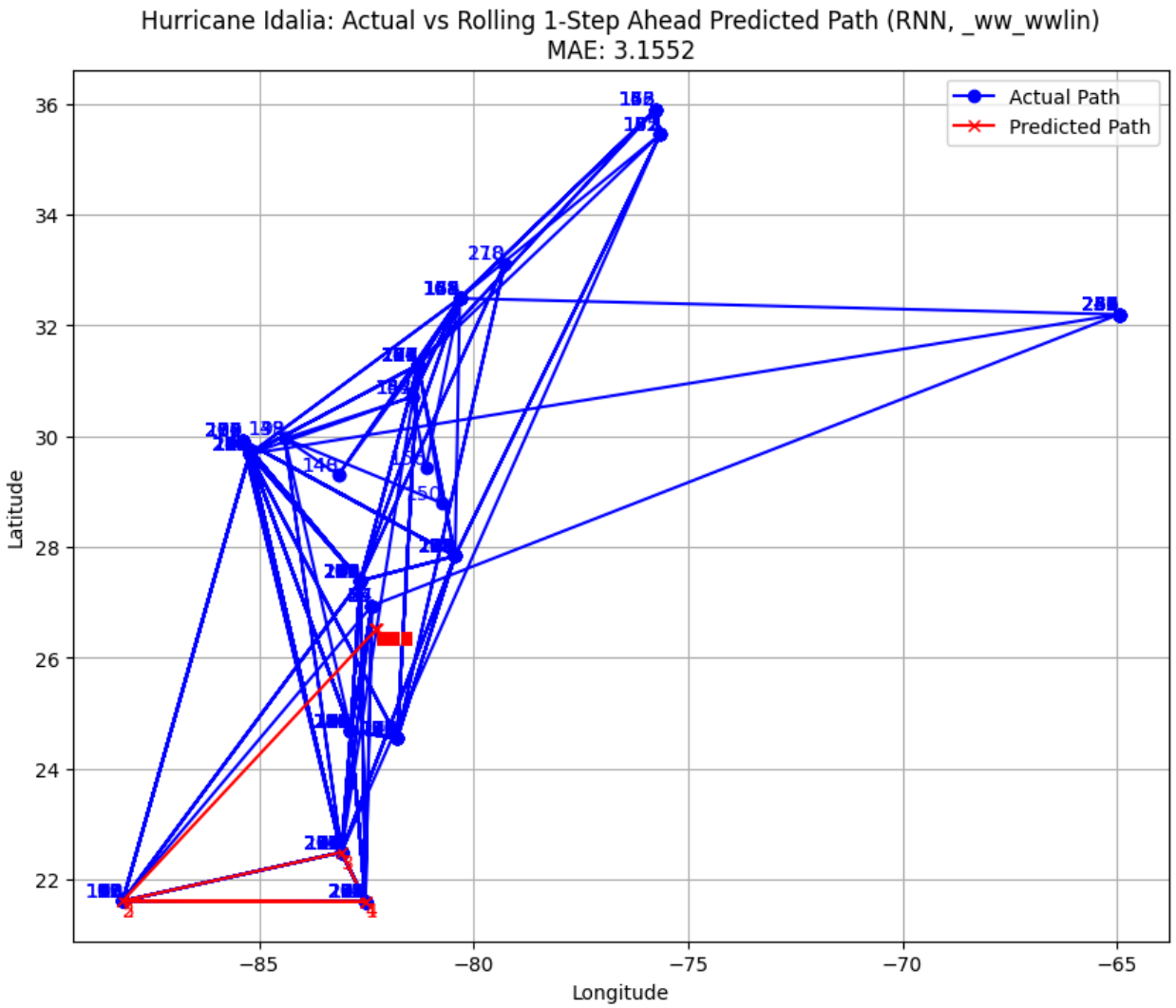



```
Processing layer: _5day_pts
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
```




```
Processing layer: _ww_wwlin
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
```



Observation: RF significantly outperforms RNN, LSTM which both perform almost equal. This is especially true for linear & polygon layers

Conclusion: Noted its a small data, RF performs better

In [52]:
```
for layer_name, df in combined_layers.items():
    if df is not None:
        print(f"Layer: {layer_name} - Number of rows: {len(df)}")
    else:
        print(f"Layer: {layer_name} - No data loaded.")
```

```
Layer: _5day_lin - Number of rows: 56
Layer: _5day_pgn - Number of rows: 56
Layer: _5day_pts - Number of rows: 504
Layer: _ww_wwlin - Number of rows: 243
```

APPENDIX: Graph at point-level forecast

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.preprocessing import StandardScaler, LabelEncoder
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
import warnings
warnings.filterwarnings('ignore')


from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
def preprocess_data(df):
    """Preprocess the hurricane data with robust coordinate parsing"""
    data = df.copy()

    # Convert date and time to datetime
    data['DateTime'] = pd.to_datetime(
        data['Date'].astype(str) + ' ' + data['Time'].astype(str).str.zfill(4),
        format='%Y%m%d %H%M',
        errors='coerce'
    )

    def parse_coordinate(coord_str, coord_type='lat'):
        """Parse coordinate string handling various formats"""
        if pd.isna(coord_str):
            return np.nan

        coord_str = str(coord_str).strip().strip('-')
        if not coord_str or coord_str == '-999':
            return np.nan

        if coord_type == 'lat':
            multiplier = -1 if 'S' in coord_str else 1
            numeric_part = coord_str.replace('S', '').replace('N', '').strip()
        else:
            multiplier = -1 if 'W' in coord_str else 1
            numeric_part = coord_str.replace('W', '').replace('E', '').strip()

        numeric_part = ''.join(c for c in numeric_part if c.isdigit() or c == '.')

        try:
            return float(numeric_part) * multiplier
        except:
            return np.nan

    # Parse latitude and longitude
    data['Lat'] = data['Latitude'].apply(lambda x: parse_coordinate(x, 'lat'))
    data['Lon'] = data['Longitude'].apply(lambda x: parse_coordinate(x, 'lon'))

    # Handle missing values in key numeric columns
    numeric_columns = ['Max_Sustained_Wind_kt', 'Min_Pressure_mb', 'Radius_of_Max_Wind_nm']
    for col in numeric_columns:
        data[col] = data[col].replace(-999, np.nan)

    # Create storm identifier
    data['Storm_ID'] = data['Basin'] + '_' + data['Cyclone_Number'].astype(str) + '_' + data['Year'].astype(str)

    # Drop rows with invalid coordinates
    initial_len = len(data)
    data = data.dropna(subset=['Lat', 'Lon'])
    removed_rows = initial_len - len(data)
    if removed_rows > 0:
        print(f"Removed {removed_rows} rows with invalid coordinates")

    # Sort by Storm and DateTime
    data = data.sort_values(['Storm_ID', 'DateTime'])

    return data


df = pd.read_csv('/content/data.csv')  # Make sure the name matches

# Preprocess the data
processed_data = preprocess_data(df)

# Show the first few rows
processed_data.sample(10)
```

	Basin	Cyclone_Number	Year	Storm_Name	Date	Time	Record_ID	Status	Latitude	Longitude	...	50kt_NW_Radius_nm	64kt_NE_Radius_nm	64k
32473	AL	14	1975	FAYE	19750923	600	NaN	TS	20.3N	53.4W	...	-999	-999	
12789	AL	3	1915	UNNAMED	19150830	0	NaN	HU	27.4N	54.6W	...	-999	-999	
14435	AL	5	1923	UNNAMED	19231004	0	NaN	EX	58.5N	43.0W	...	-999	-999	
7290	AL	6	1893	UNNAMED	18930815	1800	NaN	TS	17.6N	20.9W	...	-999	-999	
2577	AL	3	1874	UNNAMED	18740903	1800	NaN	HU	23.6N	58.9W	...	-999	-999	
50764	AL	15	2017	MARIA	20170921	600	NaN	HU	19.4N	68.2W	...	70	50	
46303	AL	8	2008	HANNA	20080829	600	NaN	TS	20.9N	61.5W	...	0	0	
37642	AL	11	1989	HUGO	19890910	1800	NaN	TD	13.3N	21.8W	...	-999	-999	
15431	AL	3	1927	UNNAMED	19270923	0	NaN	TS	11.3N	37.0W	...	-999	-999	
16144	AL	4	1931	UNNAMED	19310818	600	NaN	TD	20.8N	68.4W	...	-999	-999	

10 rows × 29 columns

```
processed_data.head(5)
```

	Basin	Cyclone_Number	Year	Storm_Name	Date	Time	Record_ID	Status	Latitude	Longitude	...	50kt_NW_Radius_nm	64kt_NE_Radius_nm	64kt
1795	AL	10	1869	UNNAMED	18691004	0	NaN	HU	31.5N	75.5W	...	-999	-999	
1796	AL	10	1869	UNNAMED	18691004	600	NaN	HU	34.5N	73.0W	...	-999	-999	
1797	AL	10	1869	UNNAMED	18691004	1200	NaN	HU	37.7N	71.5W	...	-999	-999	
1798	AL	10	1869	UNNAMED	18691004	1800	NaN	HU	40.7N	70.6W	...	-999	-999	
1799	AL	10	1869	UNNAMED	18691004	1900	L	HU	41.3N	70.5W	...	-999	-999	

5 rows × 29 columns

```
def create_flexible_sequences(df, min_sequence_length=2, max_sequence_length=10, forecast_horizon=4):
    """
    Create training sequences of variable length and corresponding future targets.

    Args:
        df: Preprocessed hurricane DataFrame.
        min_sequence_length: Minimum number of past observations.
        max_sequence_length: Maximum number of past observations to use.
        forecast_horizon: How many 6-hour steps ahead to forecast.

    Returns:
        A list of dictionaries with input sequences and targets.
    """
    sequences = []

    for storm_id, group in df.groupby('Storm_ID'):
        group = group.reset_index(drop=True)
        storm_length = len(group)

        # Skip storms that are too short
        if storm_length < min_sequence_length + 1:
            continue

        for i in range(storm_length - min_sequence_length):
            available_past = i + 1
            sequence_length = min(available_past, max_sequence_length)
```

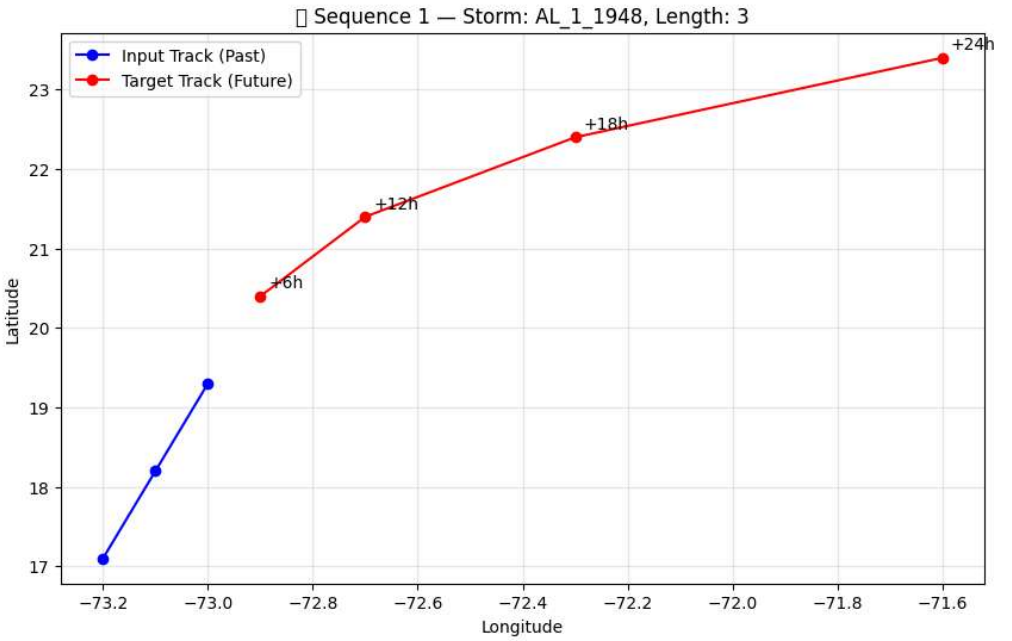
```
            if sequence_length < min_sequence_length:
                continue

            # Input sequence: the past `sequence_length` observations
            start_idx = max(0, i + 1 - sequence_length)
            input_seq = group.iloc[start_idx:i + 1]

            # Targets: lat/lon for future steps (up to forecast_horizon)
            target_positions = []
            for j in range(1, min(forecast_horizon + 1, storm_length - i)):
                target_idx = i + j
                if target_idx < storm_length:
                    target_positions.append({
                        'lat': group.iloc[target_idx]['Lat'],
                        'lon': group.iloc[target_idx]['Lon'],
                        'step': j
                    })

            if target_positions:
                sequences.append({
                    'storm_id': storm_id,
                    'input_sequence': input_seq,
                    'targets': target_positions,
                    'sequence_length': len(input_seq)
                })

    return sequences



# Set the desired parameters
MIN_SEQ_LEN = 2
MAX_SEQ_LEN = 10
FORECAST_HORIZON = 4

# Generate sequences
sequences = create_flexible_sequences(
    processed_data,
    min_sequence_length=MIN_SEQ_LEN,
    max_sequence_length=MAX_SEQ_LEN,
    forecast_horizon=FORECAST_HORIZON
)

# Show a sample sequence
print(f"Total sequences created: {len(sequences)}\n")
print("Sample sequence (index 0):")
sample = sequences[0]

print(f"Storm ID: {sample['storm_id']}")
print(f"Input sequence length: {sample['sequence_length']}")
print("Input sequence columns:", list(sample['input_sequence'].columns))
print("Target steps:", sample['targets'])
```

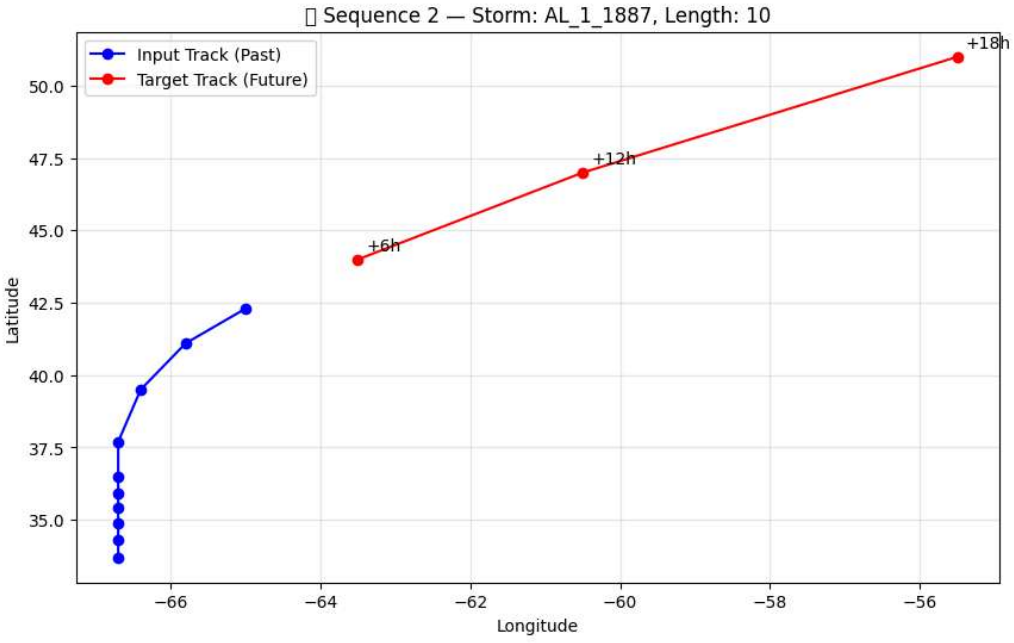
```
Total sequences created: 49320

Sample sequence (index 0):
Storm ID: AL_10_1869
Input sequence length: 2
Input sequence columns: ['Basin', 'Cyclone_Number', 'Year', 'Storm_Name', 'Date', 'Time', 'Record_ID', 'Status', 'Latitude', 'Longitude', 'Max_Sustai
Target steps: [{'lat': np.float64(37.7), 'lon': np.float64(-71.5), 'step': 1}, {'lat': np.float64(40.7), 'lon': np.float64(-70.6), 'step': 2}, {'lat
```

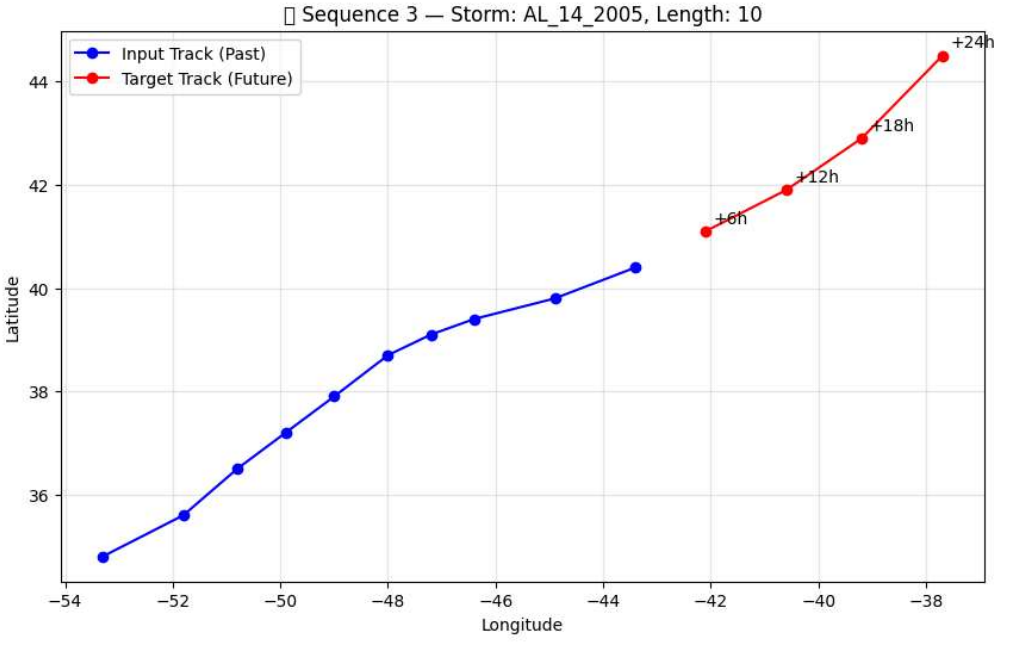
```
import random

# Ensure reproducibility
random.seed(42)

# Select 5 random sequences
sample_sequences = random.sample(sequences, min(5, len(sequences)))

# Display each sequence
for idx, seq in enumerate(sample_sequences):
    print("="*60)
    print(f"🌀 Sequence {idx + 1}")
    print(f"Storm ID: {seq['storm_id']}")
    print(f"Input Sequence Length: {seq['sequence_length']}")

    print("\n📌 Input Observations:")
    display_columns = ['DateTime', 'Lat', 'Lon', 'Max_Sustained_Wind_kt', 'Min_Pressure_mb']
    print(seq['input_sequence'][display_columns])

    print("\n🎯 Targets (Future Predictions):")
    for target in seq['targets']:
        print(f"  +{target['step']*6}h → Lat: {target['lat']:.2f}, Lon: {target['lon']:.2f}")
```

```
🎯 Targets (Future Predictions):
  +6h → Lat: 51.00, Lon: -38.00
  +12h → Lat: 51.00, Lon: -32.00
  +18h → Lat: 51.00, Lon: -26.00
  +24h → Lat: 51.00, Lon: -21.00
============================================================
🌀 Sequence 4
Storm ID: AL_9_2005
Input Sequence Length: 10

📌 Input Observations:
              DateTime   Lat   Lon  Max_Sustained_Wind_kt  Min_Pressure_mb
1  2005-08-05 00:00:00  13.6 -34.5                     25           1009.0
2  2005-08-05 06:00:00  14.6 -35.5                     25           1009.0
3  2005-08-05 12:00:00  15.6 -36.8                     30           1008.0
4  2005-08-05 18:00:00  16.6 -38.4                     30           1008.0
5  2005-08-06 00:00:00  17.2 -39.8                     30           1008.0
6  2005-08-06 06:00:00  17.7 -40.5                     30           1008.0
7  2005-08-06 12:00:00  18.2 -41.6                     30           1008.0
8  2005-08-06 18:00:00  18.8 -42.8                     30           1008.0
9  2005-08-07 00:00:00  19.3 -43.5                     30           1008.0
10 2005-08-07 06:00:00  19.7 -44.2                     30           1007.0

🎯 Targets (Future Predictions):
  +6h → Lat: 20.20, Lon: -45.00
  +12h → Lat: 20.80, Lon: -46.00
  +18h → Lat: 21.30, Lon: -47.20
  +24h → Lat: 21.80, Lon: -48.30
============================================================
🌀 Sequence 5
Storm ID: AL_20_2023
Input Sequence Length: 10

📌 Input Observations:
             DateTime   Lat   Lon  Max_Sustained_Wind_kt  Min_Pressure_mb
0 2023-10-18 18:00:00  12.9 -51.0                     35           1007.0
1 2023-10-19 00:00:00  13.0 -52.5                     35           1006.0
2 2023-10-19 06:00:00  13.2 -54.0                     45           1004.0
3 2023-10-19 12:00:00  13.4 -55.3                     50           1004.0
4 2023-10-19 18:00:00  13.5 -56.4                     50           1002.0
5 2023-10-20 00:00:00  13.6 -57.2                     50           1001.0
6 2023-10-20 06:00:00  13.7 -57.9                     50           1000.0
7 2023-10-20 12:00:00  13.9 -58.4                     60            997.0
8 2023-10-20 18:00:00  14.1 -58.9                     65            991.0
9 2023-10-21 00:00:00  14.5 -59.6                     70            991.0

🎯 Targets (Future Predictions):
  +6h → Lat: 14.90, Lon: -60.30
  +12h → Lat: 15.60, Lon: -60.60
  +18h → Lat: 16.60, Lon: -61.00
  +24h → Lat: 17.50, Lon: -61.70
```

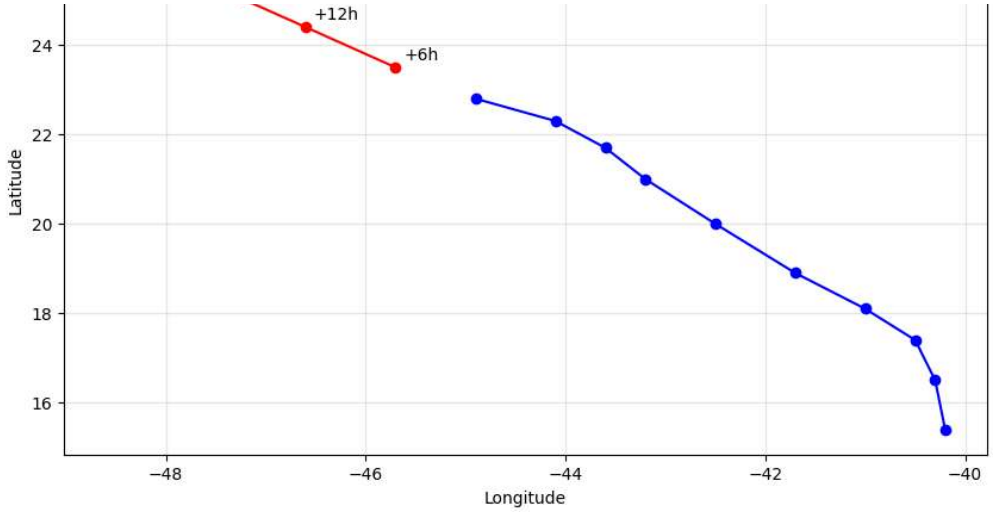
```python
import matplotlib.pyplot as plt

def plot_hurricane_sequence(seq, title="Hurricane Track Sequence"):
    input_seq = seq['input_sequence']
    targets = seq['targets']

    input_lats = input_seq['Lat'].values
    input_lons = input_seq['Lon'].values

    target_lats = [t['lat'] for t in targets]
    target_lons = [t['lon'] for t in targets]
    forecast_hours = [t['step'] * 6 for t in targets]

    plt.figure(figsize=(10, 6))

    # Plot input sequence
    plt.plot(input_lons, input_lats, 'bo-', label='Input Track (Past)', markersize=6)

    # Plot future predictions (targets)
    plt.plot(target_lons, target_lats, 'ro-', label='Target Track (Future)', markersize=6)

    # Annotate forecast steps
    for i, (lon, lat, hour) in enumerate(zip(target_lons, target_lats, forecast_hours)):
        plt.annotate(f'+{hour}h', (lon, lat), textcoords='offset points', xytext=(5, 5))

    plt.xlabel("Longitude")
    plt.ylabel("Latitude")
    plt.title(title)
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.show()


# Plot 5 random sequences
sample_sequences = random.sample(sequences, min(5, len(sequences)))

for idx, seq in enumerate(sample_sequences):
    title = f"🌀 Sequence {idx+1} — Storm: {seq['storm_id']}, Length: {seq['sequence_length']}"
    plot_hurricane_sequence(seq, title=title)
```

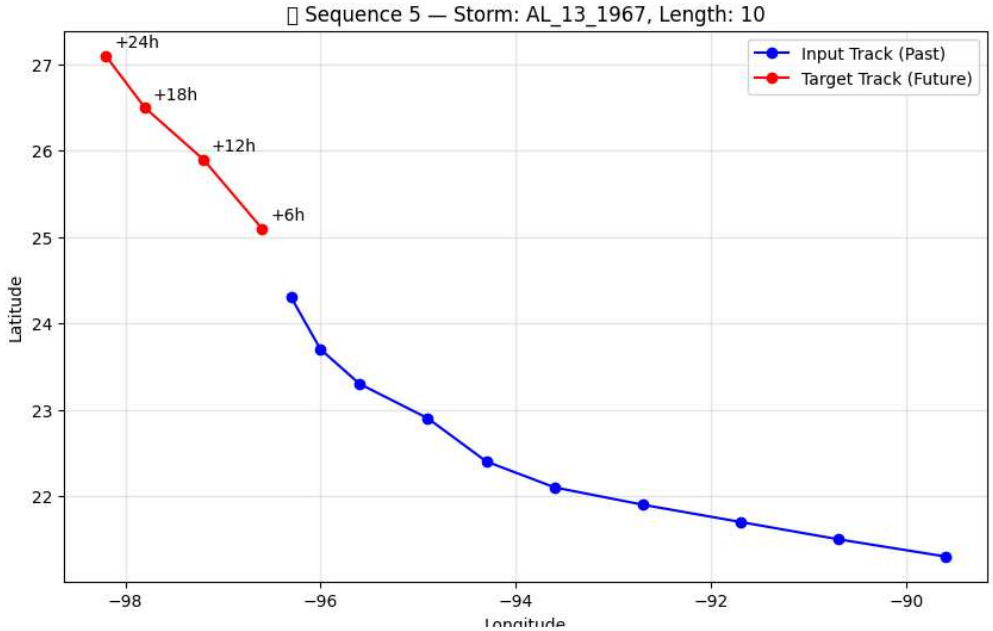

Sequence 1 — Storm: AL_1_1948, Length: 3
Input Track (Past)
Target Track (Future)
+6h
+12h
+18h
+24h
Latitude
Longitude


Sequence 2 — Storm: AL_1_1887, Length: 10
Input Track (Past)
Target Track (Future)
+6h
+12h
+18h
Latitude
Longitude


Sequence 3 — Storm: AL_14_2005, Length: 10
Input Track (Past)
Target Track (Future)
+6h
+12h
+18h
+24h
Latitude
Longitude


Sequence 4 — Storm: AL_9_1990, Length: 10
Input Track (Past)
Target Track (Future)
+18h
+24h


+12h
+6h
24
22
20
18
16
Latitude
−48
−46
−44
−42
−40
Longitude


Sequence 5 — Storm: AL_13_1967, Length: 10
Input Track (Past)
Target Track (Future)
+24h
+18h
+12h
+6h
27
26
25
24
23
22
Latitude
−98
−96
−94
−92
−90
Longitude

```
from sklearn.preprocessing import LabelEncoder

# Create and fit encoder
basin_encoder = LabelEncoder()
basin_encoder.fit(processed_data['Basin'].unique())
```

```
LabelEncoder
LabelEncoder()
```

```
def extract_features(sequence, basin_encoder, max_sequence_length=6):
    features = []
    seq_len = len(sequence)

    # Pad/truncate observations up to max_sequence_length
    for i in range(max_sequence_length):
        if i < seq_len:
            obs = sequence.iloc[i]

            # Positional features
            features.extend([obs['Lat'], obs['Lon']])

            # Intensity features with fallback
            features.extend([
                obs['Max_Sustained_Wind_kt'] if pd.notna(obs['Max_Sustained_Wind_kt']) else 50,
                obs['Min_Pressure_mb'] if pd.notna(obs['Min_Pressure_mb']) else 1000,
                obs['Radius_of_Max_Wind_nm'] if pd.notna(obs['Radius_of_Max_Wind_nm']) else 30
            ])

            # Time features
            if pd.notna(obs['DateTime']):
                features.extend([
                    obs['DateTime'].hour,
                    obs['DateTime'].dayofyear,
                    obs['DateTime'].month
                ])
            else:
                features.extend([12, 200, 7])
        else:
            # Padding for missing observations (8 features per step)
            features.extend([0] * 8)

    # Motion features (velocity, speed, direction)
    if seq_len >= 2:
        lat_vel = sequence.iloc[-1]['Lat'] - sequence.iloc[-2]['Lat']
        lon_vel = sequence.iloc[-1]['Lon'] - sequence.iloc[-2]['Lon']
        speed = np.sqrt(lat_vel**2 + lon_vel**2)
        direction = np.arctan2(lat_vel, lon_vel)
        features.extend([lat_vel, lon_vel, speed, direction])

        # Acceleration
        if seq_len >= 3:
            prev_lat_vel = sequence.iloc[-2]['Lat'] - sequence.iloc[-3]['Lat']
            prev_lon_vel = sequence.iloc[-2]['Lon'] - sequence.iloc[-3]['Lon']
            lat_acc = lat_vel - prev_lat_vel
            lon_acc = lon_vel - prev_lon_vel
            features.extend([lat_acc, lon_acc])
        else:
            features.extend([0, 0])
    else:
        features.extend([0, 0, 0, 0, 0, 0])  # Padding if not enough data

    # Add basin and sequence length
    basin_encoded = basin_encoder.transform([sequence.iloc[0]['Basin']])[0]
    features.extend([basin_encoded, seq_len])

    return features



# Try feature extraction on the first sequence
features = extract_features(sequences[0]['input_sequence'], basin_encoder, max_sequence_length=6)

print(f"Feature vector length: {len(features)}")
print("First 10 features:", features[:10])
```

```
Feature vector length: 56
First 10 features: [np.float64(31.5), np.float64(-75.5), np.int64(90), 1000, 30, 0, 277, 10, np.float64(34.5), np.float64(-73.0)]
```

```
# Randomly sample 10 sequences
sample_sequences = random.sample(sequences, min(10, len(sequences)))

print("="*70)
print("🔍 Showing extracted features for 10 random sequences\n")

for idx, seq in enumerate(sample_sequences):
    print("="*70)
    print(f"🌀 Sequence {idx+1}")
    print(f"Storm ID: {seq['storm_id']}")
```

```
    print(f"Input Sequence Length: {seq['sequence_length']}")

    # Extract features
    feat = extract_features(seq['input_sequence'], basin_encoder, max_sequence_length=6)

    # Display features
    print(f"\n🧩 Feature Vector Length: {len(feat)}")
    print(f"📋 First 10 Features: {feat[:10]}")
    print(f"📋 Last 5 Features (Velocity/Accel/Basin/Len): {feat[-7:]}")
```

```
📋 Last 5 Features (Velocity/Accel/Basin/Len): [np.float64(1.2000000000000028), np.float64(1.341640786499877), np.float64(0.4636476090008061), np.f
======================================================================
🌀 Sequence 4
Storm ID: AL_12_2011
Input Sequence Length: 6

🧩 Feature Vector Length: 56
📋 First 10 Features: [np.float64(9.5), np.float64(-19.0), np.int64(20), np.float64(1012.0), 30, 0, 240, 8, np.float64(9.4), np.float64(-20.3)]
📋 Last 5 Features (Velocity/Accel/Basin/Len): [np.float64(-1.4000000000000021), np.float64(1.4142135623730971), np.float64(2.99969559898563), np.f
======================================================================
🌀 Sequence 5
Storm ID: AL_6_1962
Input Sequence Length: 9

🧩 Feature Vector Length: 56
📋 First 10 Features: [np.float64(32.2), np.float64(-77.7), np.int64(40), 1000, 30, 0, 180, 6, np.float64(32.4), np.float64(-77.0)]
📋 Last 5 Features (Velocity/Accel/Basin/Len): [np.float64(0.20000000000000284), np.float64(0.282842712474618), np.float64(-0.7853981633974305), np
======================================================================
🌀 Sequence 6
Storm ID: AL_3_2013
Input Sequence Length: 10

🧩 Feature Vector Length: 56
📋 First 10 Features: [np.float64(9.3), np.float64(-41.4), np.int64(35), np.float64(1011.0), 30, 12, 188, 7, np.float64(9.5), np.float64(-43.8)]
📋 Last 5 Features (Velocity/Accel/Basin/Len): [np.float64(-2.6000000000000014), np.float64(2.7513632984395224), np.float64(2.8083483524781205), np
======================================================================
🌀 Sequence 7
Storm ID: AL_10_2016
Input Sequence Length: 10

🧩 Feature Vector Length: 56
📋 First 10 Features: [np.float64(20.5), np.float64(-49.3), np.int64(35), np.float64(1007.0), 30, 6, 256, 9, np.float64(21.2), np.float64(-50.2)]
📋 Last 5 Features (Velocity/Accel/Basin/Len): [np.float64(-0.5), np.float64(1.6763054614240223), np.float64(1.8736811951698678), np.float64(3.5527
======================================================================
🌀 Sequence 8
Storm ID: AL_10_2009
Input Sequence Length: 3

🧩 Feature Vector Length: 56
📋 First 10 Features: [np.float64(15.1), np.float64(-49.2), np.int64(30), np.float64(1008.0), 30, 0, 279, 10, np.float64(16.1), np.float64(-50.5)]
📋 Last 5 Features (Velocity/Accel/Basin/Len): [np.float64(-1.3999999999999986), np.float64(1.6643316977093219), np.float64(2.5702551737561667), np
======================================================================
🌀 Sequence 9
Storm ID: AL_12_2024
Input Sequence Length: 5

🧩 Feature Vector Length: 56
📋 First 10 Features: [np.float64(13.8), np.float64(-32.1), np.int64(30), np.float64(1006.0), np.float64(60.0), 18, 273, 9, np.float64(13.8), np.fl
📋 Last 5 Features (Velocity/Accel/Basin/Len): [np.float64(-1.0), np.float64(1.0440306508910548), np.float64(2.8501358591119272), np.float64(0.4999
======================================================================
🌀 Sequence 10
Storm ID: AL_1_1866
Input Sequence Length: 7

🧩 Feature Vector Length: 56
📋 First 10 Features: [np.float64(28.0), np.float64(-87.3), np.int64(90), 1000, 30, 12, 192, 7, np.float64(28.0), np.float64(-87.8)]
📋 Last 5 Features (Velocity/Accel/Basin/Len): [np.float64(-0.5999999999999943), np.float64(0.5999999999999943), np.float64(3.141592653589793), np
```

```
def prepare_training_data(sequences, basin_encoder, max_sequence_length=6):
    X = []
    y_lat = []
    y_lon = []

    for seq_data in sequences:
        features = extract_features(seq_data['input_sequence'], basin_encoder, max_sequence_length)

        for target in seq_data['targets']:
            # Append forecast step to the end
            X.append(features + [target['step']])
            y_lat.append(target['lat'])
            y_lon.append(target['lon'])

    return np.array(X), np.array(y_lat), np.array(y_lon)


from sklearn.model_selection import train_test_split

# Get all unique storm IDs
storm_ids = list(set([seq['storm_id'] for seq in sequences]))

# Train/test split by storm ID
train_storm_ids, test_storm_ids = train_test_split(
    storm_ids, test_size=0.2, random_state=42
)
```

```
# Separate sequences
train_sequences = [seq for seq in sequences if seq['storm_id'] in train_storm_ids]
test_sequences = [seq for seq in sequences if seq['storm_id'] in test_storm_ids]

print(f"✅ Training storms: {len(train_storm_ids)}, Testing storms: {len(test_storm_ids)}")
print(f"✅ Training sequences: {len(train_sequences)}, Testing sequences: {len(test_sequences)}")
```

```
✅ Training storms: 1566, Testing storms: 392
✅ Training sequences: 39262, Testing sequences: 10058
```

```
# Build datasets
X_train, y_lat_train, y_lon_train = prepare_training_data(train_sequences, basin_encoder, max_sequence_length=6)
X_test, y_lat_test, y_lon_test = prepare_training_data(test_sequences, basin_encoder, max_sequence_length=6)

print(f"✅ Training samples: {len(X_train)}, Testing samples: {len(X_test)}")
print(f"🧩 Feature dimension: {X_train.shape[1]}")
```

```
✅ Training samples: 152360, Testing samples: 39057
🧩 Feature dimension: 57
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
from sklearn.ensemble import RandomForestRegressor

# Initialize models
lat_model = RandomForestRegressor(n_estimators=200, random_state=42, max_depth=20,n_jobs=-1)
lon_model = RandomForestRegressor(n_estimators=200, random_state=42, max_depth=20,n_jobs=-1)

# Train
print("🎯 Training latitude model...")
lat_model.fit(X_train_scaled, y_lat_train)

print("🎯 Training longitude model...")
lon_model.fit(X_train_scaled, y_lon_train)
```

```
🎯 Training latitude model...
🎯 Training longitude model...
RandomForestRegressor(max_depth=20, n_estimators=200, n_jobs=-1,
                      random_state=42)
```

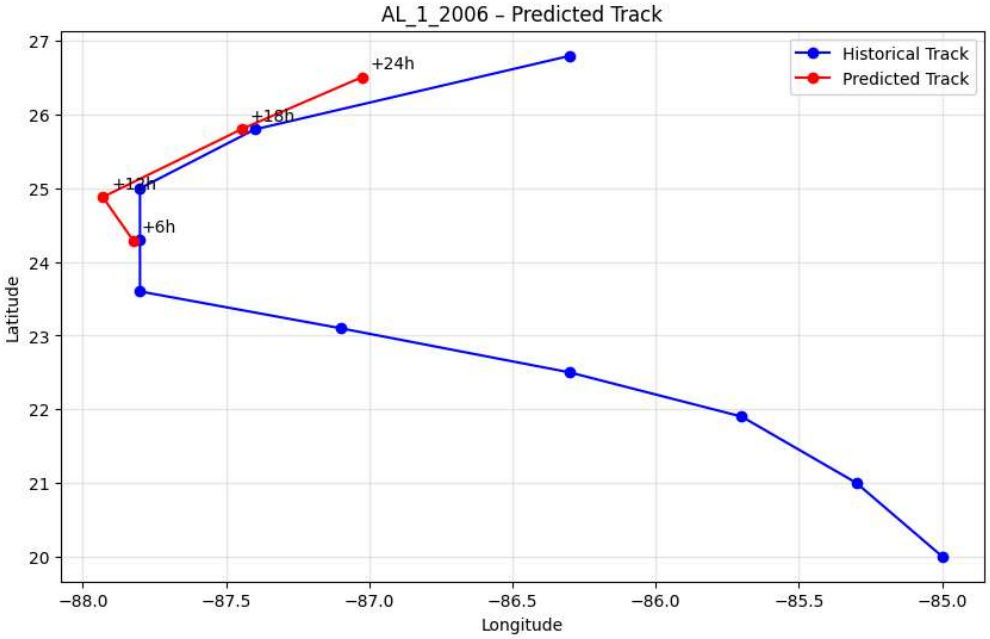
```
from sklearn.metrics import mean_absolute_error
import numpy as np

# Predict
lat_pred = lat_model.predict(X_test_scaled)
lon_pred = lon_model.predict(X_test_scaled)

# Evaluate
lat_mae = mean_absolute_error(y_lat_test, lat_pred)
lon_mae = mean_absolute_error(y_lon_test, lon_pred)

# Distance error in nautical miles
distance_error = np.sqrt((lat_pred - y_lat_test)**2 + (lon_pred - y_lon_test)**2) * 60
avg_distance_error = np.mean(distance_error)

# Print results
print("\n📈 Model Performance:")
print(f"Latitude MAE:  {lat_mae:.4f}°")
print(f"Longitude MAE: {lon_mae:.4f}°")
print(f"🌊 Avg Distance Error: {avg_distance_error:.2f} nautical miles")
```

```

📈 Model Performance:
Latitude MAE:  0.8009°
Longitude MAE: 1.1310°
🌊 Avg Distance Error: 92.16 nautical miles
```

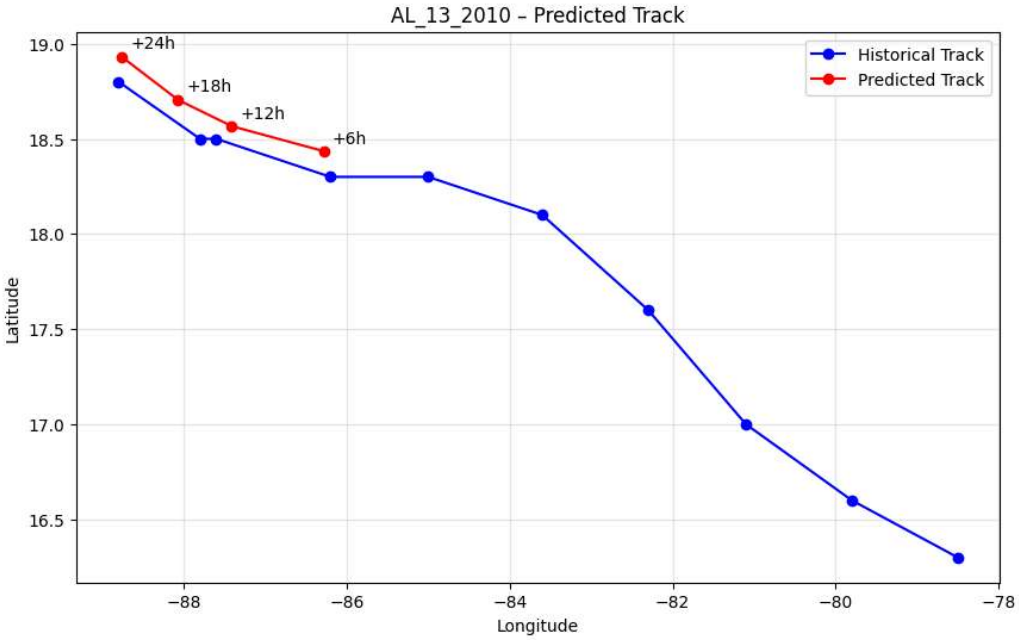
```
from datetime import timedelta

def predict_track(input_sequence, lat_model, lon_model, scaler, basin_encoder,
                  forecast_steps=4, max_sequence_length=6):
    predictions = []

    for step in range(1, forecast_steps + 1):
        features = extract_features(input_sequence, basin_encoder, max_sequence_length)
        features.append(step)  # Add step to features
        X_scaled = scaler.transform([features])

        # Predict
        pred_lat = lat_model.predict(X_scaled)[0]
        pred_lon = lon_model.predict(X_scaled)[0]

        # Predict time
        last_time = input_sequence.iloc[-1]['DateTime']
```

```
        pred_time = last_time + timedelta(hours=6 * step) if pd.notna(last_time) else None

        predictions.append({
            'step': step,
            'forecast_hour': step * 6,
            'predicted_lat': pred_lat,
            'predicted_lon': pred_lon,
            'predicted_time': pred_time
        })

    return predictions



def plot_track_prediction(historical_data, predictions, title="Hurricane Track Prediction"):
    plt.figure(figsize=(10, 6))

    # Historical path
    plt.plot(historical_data['Lon'], historical_data['Lat'], 'bo-', label='Historical Track')

    # Predicted path
    pred_lons = [p['predicted_lon'] for p in predictions]
    pred_lats = [p['predicted_lat'] for p in predictions]
    plt.plot(pred_lons, pred_lats, 'ro-', label='Predicted Track')

    # Labels
    for p in predictions:
        plt.annotate(f"+{p['forecast_hour']}h", (p['predicted_lon'], p['predicted_lat']),
                     textcoords="offset points", xytext=(5, 5))

    plt.xlabel("Longitude")
    plt.ylabel("Latitude")
    plt.title(title)
    plt.grid(True, alpha=0.3)
    plt.legend()
    plt.show()



# Group processed data by storm
storm_groups = processed_data.groupby('Storm_ID')
train_storm_ids_set = set(train_storm_ids)

# Get training storms only
train_storm_groups = [(sid, group) for sid, group in storm_groups if sid in train_storm_ids_set]

# Pick 10 random storms
random.seed()
sample_storms = random.sample(train_storm_groups, min(10, len(train_storm_groups)))

# Loop through and predict
for idx, (storm_id, storm_data) in enumerate(sample_storms):
    storm_data = storm_data.sort_values('DateTime')
    input_data = storm_data.iloc[:min(6, len(storm_data))]  # Use up to 6 obs

    predictions = predict_track(
        input_sequence=input_data,
        lat_model=lat_model,
        lon_model=lon_model,
        scaler=scaler,
        basin_encoder=basin_encoder,
        forecast_steps=4,
        max_sequence_length=6
    )

    print("=" * 70)
    print(f"🌀 Storm {idx+1}: {storm_id}")
    print(f"Used {len(input_data)} observations to predict next 24 hours (4 x 6h)\n")

    for pred in predictions:
        print(f"  +{pred['forecast_hour']}h → Lat: {pred['predicted_lat']:.2f}, Lon: {pred['predicted_lon']:.2f}")

    plot_track_prediction(storm_data.iloc[:min(10, len(storm_data))], predictions,
                          title=f"{storm_id} – Predicted Track")
```

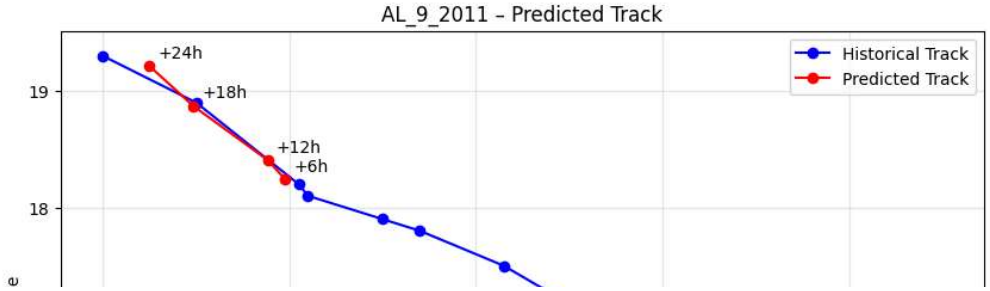
```
======================================================================
🌀 Storm 1: AL_1_2006
Used 6 observations to predict next 24 hours (4 x 6h)

  +6h → Lat: 24.29, Lon: -87.82
  +12h → Lat: 24.88, Lon: -87.93
  +18h → Lat: 25.81, Lon: -87.44
  +24h → Lat: 26.51, Lon: -87.02
```

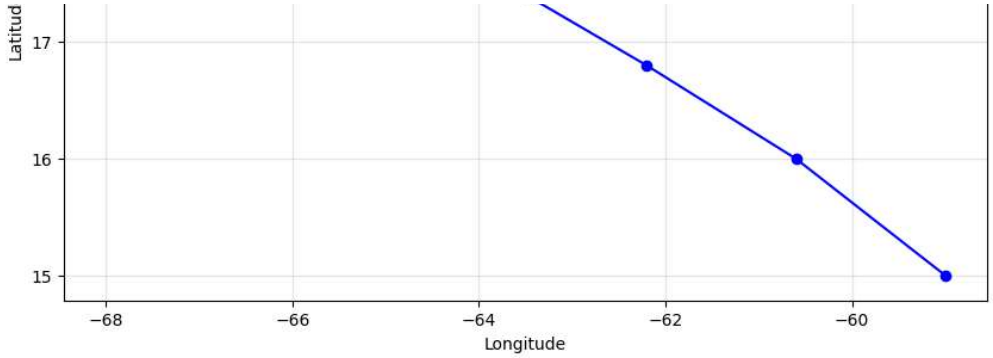



```
======================================================================
🌀 Storm 2: AL_13_2010
Used 6 observations to predict next 24 hours (4 x 6h)

  +6h → Lat: 18.44, Lon: -86.28
  +12h → Lat: 18.57, Lon: -87.41
  +18h → Lat: 18.70, Lon: -88.07
  +24h → Lat: 18.93, Lon: -88.76
```




```
======================================================================
🌀 Storm 3: AL_9_2011
Used 6 observations to predict next 24 hours (4 x 6h)

  +6h → Lat: 18.24, Lon: -66.05
  +12h → Lat: 18.41, Lon: -66.23
  +18h → Lat: 18.87, Lon: -67.03
  +24h → Lat: 19.21, Lon: -67.50
```



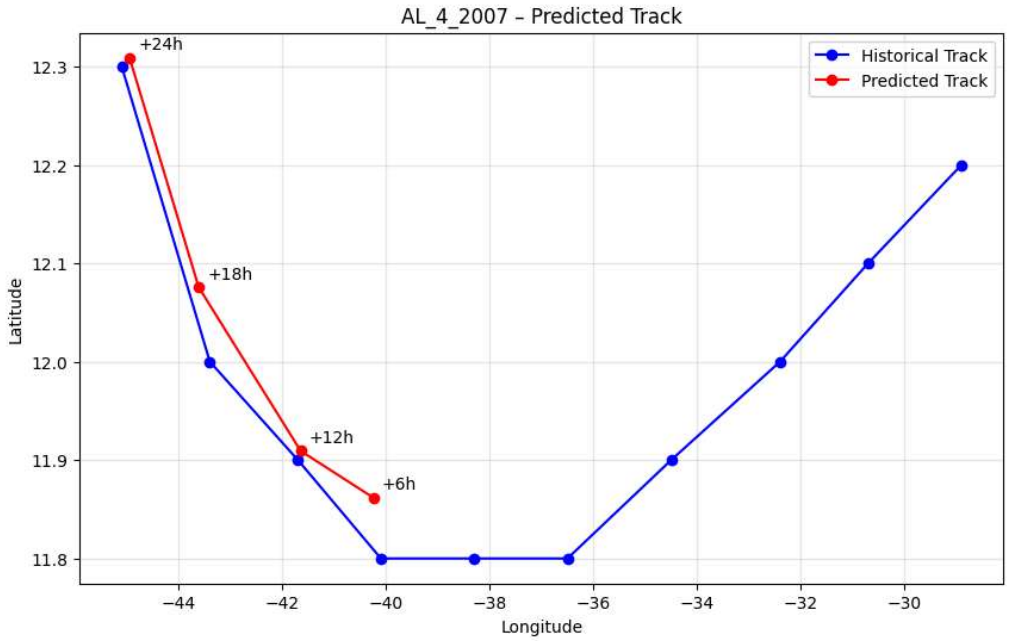


==========

🌀 Storm 4: AL_4_2007

Used 6 observations to predict next 24 hours (4 x 6h)

+6h → Lat: 11.86, Lon: -40.24
+12h → Lat: 11.91, Lon: -41.65
+18h → Lat: 12.08, Lon: -43.61
+24h → Lat: 12.31, Lon: -44.95

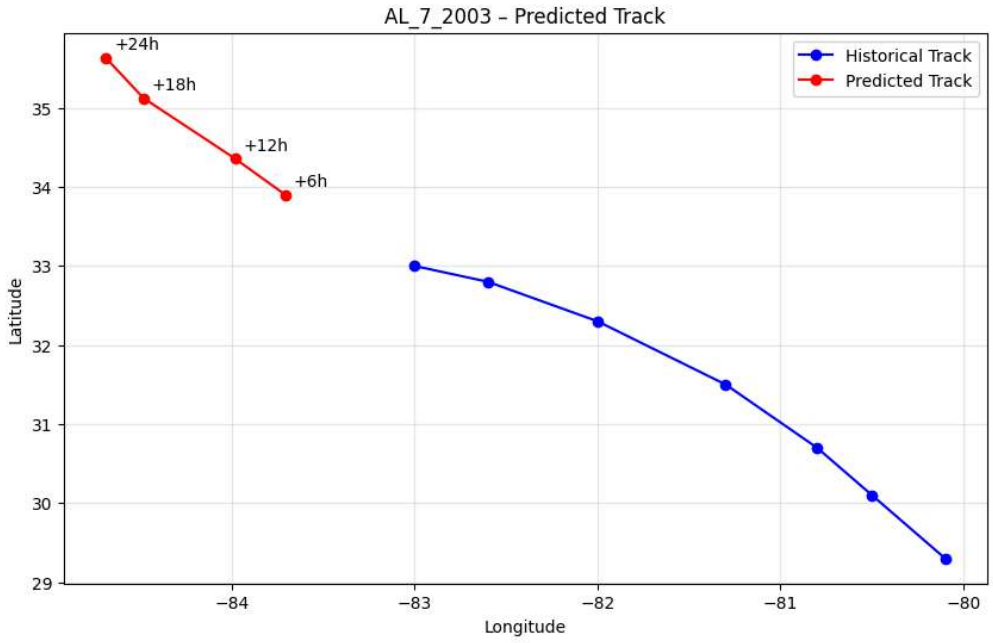


==========

🌀 Storm 5: AL_7_2003

Used 6 observations to predict next 24 hours (4 x 6h)

+6h → Lat: 33.90, Lon: -83.71
+12h → Lat: 34.36, Lon: -83.98
+18h → Lat: 35.12, Lon: -84.48
+24h → Lat: 35.63, Lon: -84.69



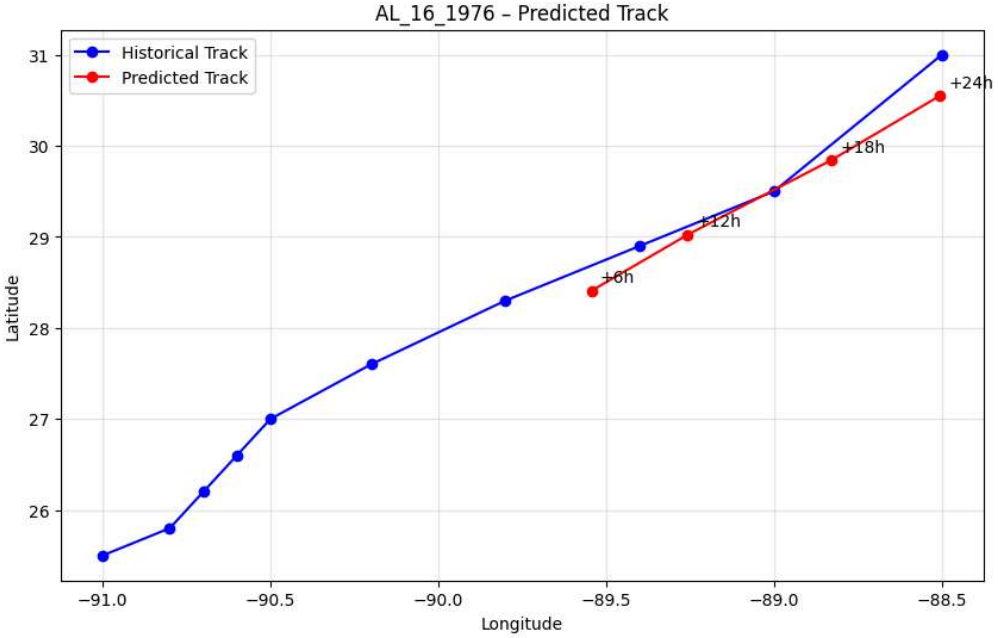
==========

🌀 Storm 6: AL_16_1976

Used 6 observations to predict next 24 hours (4 x 6h)

+6h → Lat: 28.41, Lon: -89.54

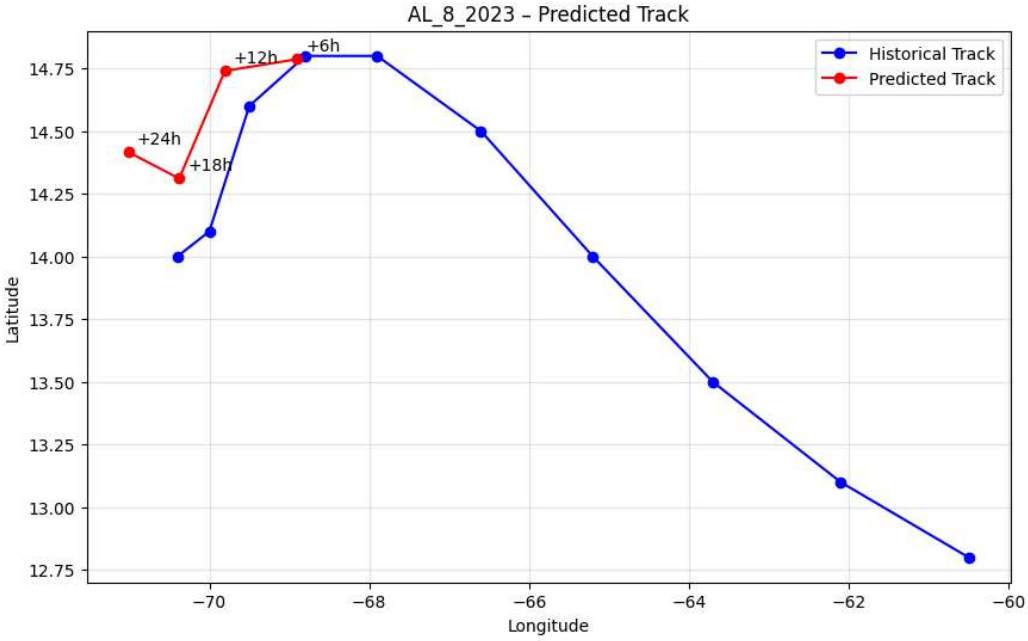
+12h → Lat: 29.02, Lon: -89.26
+18h → Lat: 29.84, Lon: -88.83
+24h → Lat: 30.55, Lon: -88.51



==========

🌀 Storm 7: AL_8_2023
Used 6 observations to predict next 24 hours (4 x 6h)

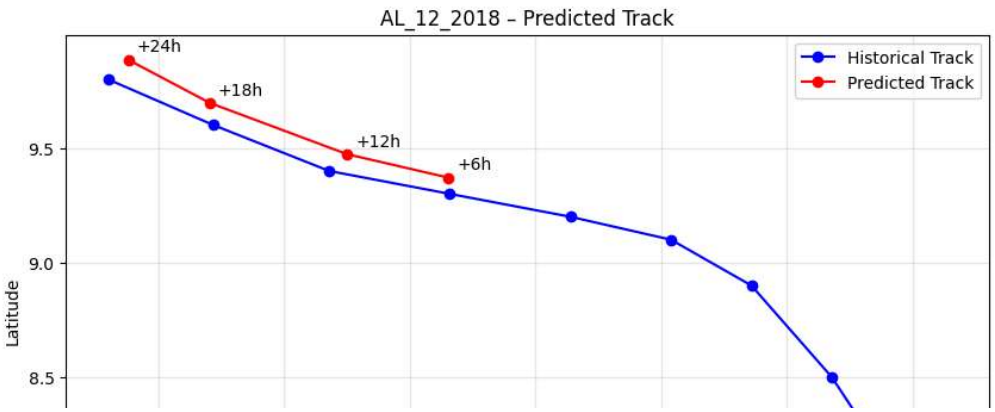
+6h → Lat: 14.79, Lon: -68.90
+12h → Lat: 14.74, Lon: -69.80
+18h → Lat: 14.31, Lon: -70.38
+24h → Lat: 14.42, Lon: -71.01

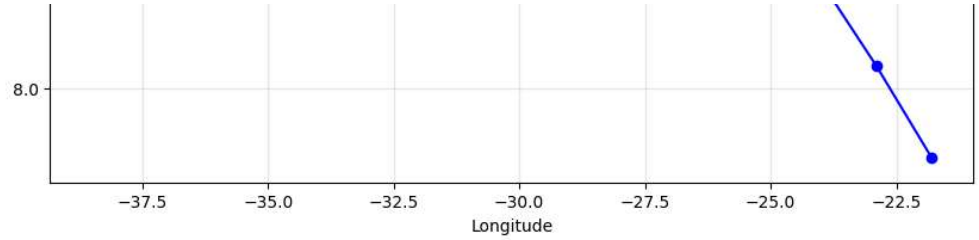


==========

🌀 Storm 8: AL_12_2018
Used 6 observations to predict next 24 hours (4 x 6h)

+6h → Lat: 9.37, Lon: -31.73
+12h → Lat: 9.47, Lon: -33.75
+18h → Lat: 9.70, Lon: -36.47
+24h → Lat: 9.88, Lon: -38.09





==========

🌀 Storm 9: AL_3_1925

Used 6 observations to predict next 24 hours (4 x 6h)

+6h → Lat: 26.69, Lon: -99.42
+12h → Lat: 27.45, Lon: -100.13
+18h → Lat: 27.89, Lon: -100.93
+24h → Lat: 28.35, Lon: -101.38