lstm-1

April 16, 2025

[ ]:
```
!curl https://www.bgc-jena.mpg.de/wetter/mpi_saale_2021b.zip -o mpi_saale_2021b.
 ↪zip
```

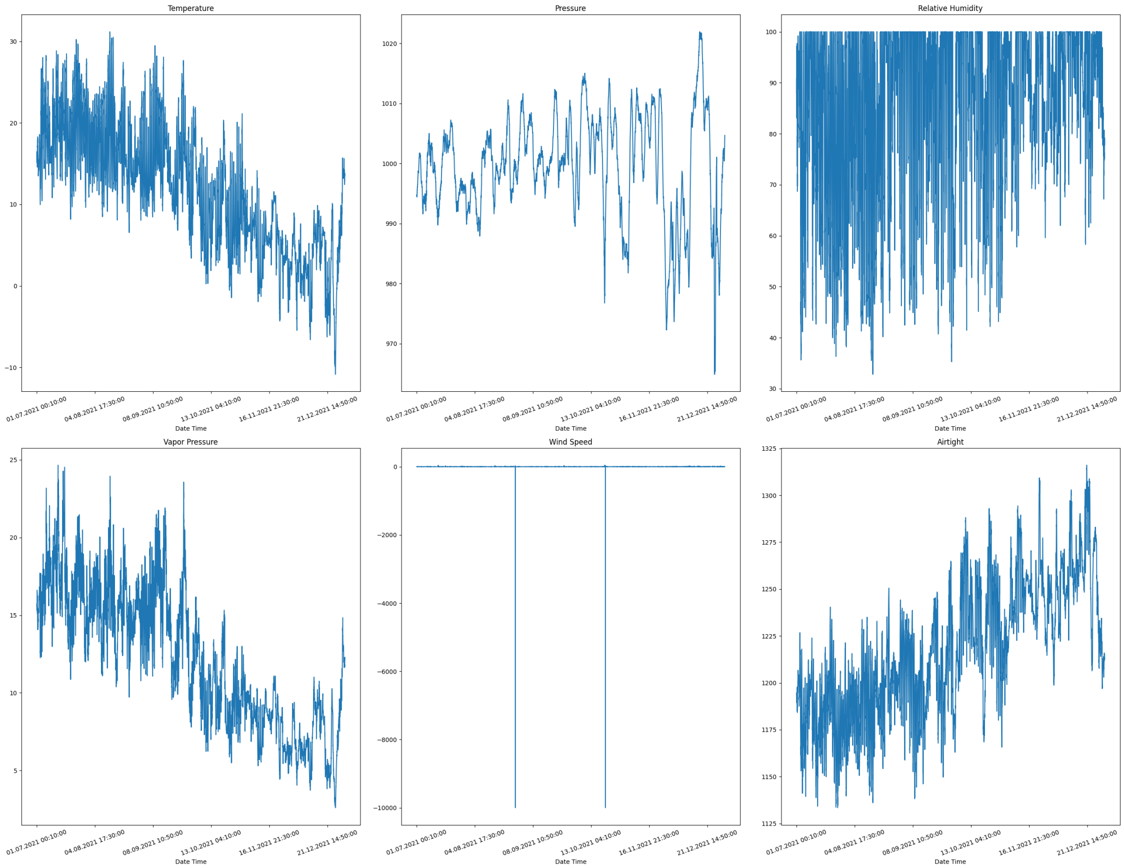
```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 1486k  100 1486k    0     0   481k      0  0:00:03  0:00:03 --:--:--  481k
```

[ ]:
```
import zipfile
import pandas

zip_file = zipfile.ZipFile("mpi_saale_2021b.zip")
zip_file.extractall()
csv_path = "mpi_saale_2021b.csv"
data_frame = pandas.read_csv(csv_path)
```

[ ]:
```
time = data_frame['Date Time']
temperature = data_frame['T (degC)']
pressure = data_frame['p (mbar)']
relative_humidity = data_frame['rh (%)']
vapor_pressure = data_frame['VPact (mbar)']
wind_speed = data_frame['wv (m/s)']
airtight = data_frame['rho (g/m**3)']
```

[ ]:
```
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

plt.subplots(nrows=2, ncols=3, figsize=(26, 20))

ax = plt.subplot(2, 3, 1)
temperature.index = time
temperature.head()
temperature.plot(rot=20)
plt.title('Temperature')

ax = plt.subplot(2, 3, 2)
pressure.index = time
pressure.head()
pressure.plot(rot=20)
```

```python
plt.title('Pressure')

ax = plt.subplot(2, 3, 3)
relative_humidity.index = time
relative_humidity.head()
relative_humidity.plot(rot=20)
plt.title('Relative Humidity')

ax = plt.subplot(2, 3, 4)
vapor_pressure.index = time
vapor_pressure.head()
vapor_pressure.plot(rot=20)
plt.title('Vapor Pressure')

ax = plt.subplot(2, 3, 5)
wind_speed.index = time
wind_speed.head()
wind_speed.plot(rot=20)
plt.title('Wind Speed')

ax = plt.subplot(2, 3, 6)
airtight.index = time
airtight.head()
airtight.plot(rot=20)
plt.title('Airtight')

plt.tight_layout()
plt.show()
```



[ ]:
```python
def normalize(data):
    data_mean = data.mean(axis=0)
    data_std = data.std(axis=0)
    return (data - data_mean) / data_std
```

[ ]:
```python
features = pandas.concat([temperature, pressure, relative_humidity,
 ↪vapor_pressure, wind_speed, airtight], axis=1)
features.index = time
features
```

[ ]:
```
                     T (degC)  p (mbar)  rh (%)  VPact (mbar)  wv (m/s)  \
Date Time
01.07.2021 00:10:00     15.30    994.66   88.00         15.32      0.52
01.07.2021 00:20:00     15.16    994.60   89.90         15.51      0.56
01.07.2021 00:30:00     15.18    994.56   90.90         15.71      1.09
01.07.2021 00:40:00     15.73    994.55   86.70         15.52      1.09
01.07.2021 00:50:00     16.18    994.58   84.30         15.53      1.28
…                         …        …    …               …         …
31.12.2021 23:20:00     13.53   1004.48   79.01         12.27      3.03
31.12.2021 23:30:00     13.49   1004.54   79.09         12.25      3.22
```

```
31.12.2021 23:40:00     13.52   1004.53   78.68          12.21       3.59
31.12.2021 23:50:00     13.55   1004.62   78.32          12.18       3.54
01.01.2022 00:00:00     13.52   1004.68   78.39          12.16       2.97

                     rho (g/m**3)
Date Time
01.07.2021 00:10:00       1194.25
01.07.2021 00:20:00       1194.67
01.07.2021 00:30:00       1194.45
01.07.2021 00:40:00       1192.25
01.07.2021 00:50:00       1190.43
…                           …
31.12.2021 23:20:00       1214.96
31.12.2021 23:30:00       1215.21
31.12.2021 23:40:00       1215.09
31.12.2021 23:50:00       1215.09
01.01.2022 00:00:00       1215.30

[26496 rows x 6 columns]
```

[ ]:
```
features = normalize(features.values)
features = pandas.DataFrame(features)
features
```

[ ]:
```
              0         1         2         3         4         5
0      0.503930 -0.586142  0.136120  0.737202  0.003771 -0.687664
1      0.485351 -0.593114  0.252726  0.780616  0.004097 -0.675877
2      0.488005 -0.597762  0.314097  0.826314  0.008410 -0.682051
3      0.560994 -0.598924  0.056337  0.782900  0.008410 -0.743794
4      0.620712 -0.595438 -0.090955  0.785185  0.009957 -0.794872
…           …         …         …         …         …         …
26491  0.269038  0.554893 -0.415611  0.040296  0.024199 -0.106443
26492  0.263729  0.561865 -0.410702  0.035726  0.025745 -0.099427
26493  0.267711  0.560703 -0.435864  0.026587  0.028756 -0.102795
26494  0.271692  0.571161 -0.457958  0.019732  0.028349 -0.102795
26495  0.267711  0.578132 -0.453662  0.015162  0.023711 -0.096901

[26496 rows x 6 columns]
```

[ ]:
```
training_size = int ( 0.8 * features.shape[0])
train_data = features.loc[0 : training_size - 1]
val_data = features.loc[training_size:]
```

[ ]:
```
start = 432 + 36
end = start + training_size

x_train = train_data.values
```

```
y_train = features.iloc[start:end][[0]]

sequence_length = int(432 / 6)
```

[ ]:
```
from tensorflow import keras

dataset_train = keras.preprocessing.timeseries_dataset_from_array(
    data=x_train,
    targets=y_train,
    sequence_length=sequence_length,
    sampling_rate=6,
    batch_size=64,
)
```

[ ]:
```
x_val_end = len(val_data) - start

label_start = training_size + start

x_val = val_data.iloc[:x_val_end][[i for i in range(6)]].values
y_val = features.iloc[label_start:][[0]]

dataset_val = keras.preprocessing.timeseries_dataset_from_array(
    x_val,
    y_val,
    sequence_length=sequence_length,
    sampling_rate=6,
    batch_size=64,
)
```

[ ]:
```
for batch in dataset_train.take(1):
    inputs, targets = batch

inputs = keras.layers.Input(shape=(inputs.shape[1], inputs.shape[2]))
lstm_out = keras.layers.LSTM(32)(inputs)
outputs = keras.layers.Dense(1)(lstm_out)

model = keras.Model(name="Weather_forcaster",inputs=inputs, outputs=outputs)
model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001), loss="mse")
model.summary()
```

```
Model: "Weather_forcaster"


 Layer (type)                       Output Shape                          ␣
↪Param #
```

```
 input_layer_1 (InputLayer)              (None, 72, 6)                              ␣
↪  0

 lstm_1 (LSTM)                           (None, 32)                               ␣
↪4,992

 dense_1 (Dense)                         (None, 1)                                  ␣
↪ 33
```

Total params: 5,025 (19.63 KB)

Trainable params: 5,025 (19.63 KB)

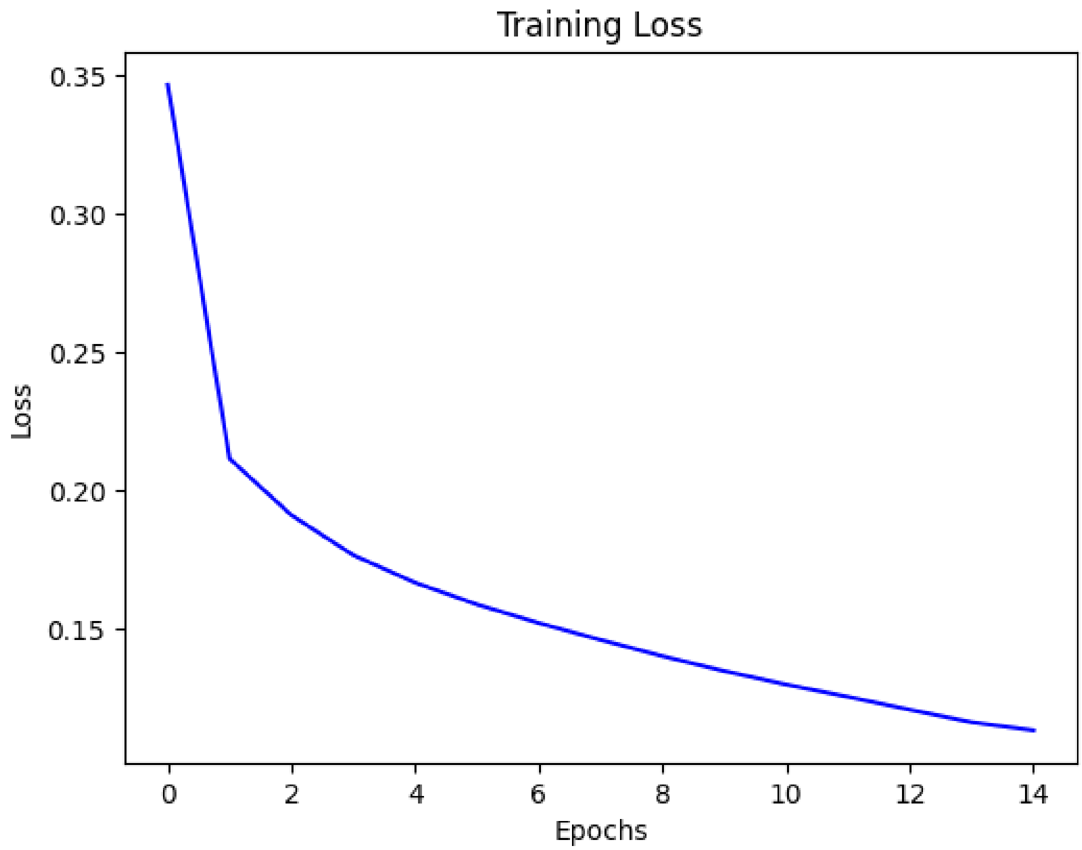
Non-trainable params: 0 (0.00 B)

[ ]:
```
history = model.fit(
    dataset_train,
    epochs=15,
    validation_data=dataset_val
)
```

```
Epoch 1/15
325/325                   16s 42ms/step -
loss: 0.4128 - val_loss: 0.2254
Epoch 2/15
325/325                   13s 41ms/step -
loss: 0.2287 - val_loss: 0.2514
Epoch 3/15
325/325                   14s 42ms/step -
loss: 0.2094 - val_loss: 0.2590
Epoch 4/15
325/325                   13s 40ms/step -
loss: 0.1874 - val_loss: 0.2558
Epoch 5/15
325/325                   13s 40ms/step -
loss: 0.1750 - val_loss: 0.2515
Epoch 6/15
325/325                   13s 41ms/step -
loss: 0.1658 - val_loss: 0.2482
Epoch 7/15
325/325                   13s 41ms/step -
loss: 0.1583 - val_loss: 0.2438
Epoch 8/15
325/325                   21s 42ms/step -
```

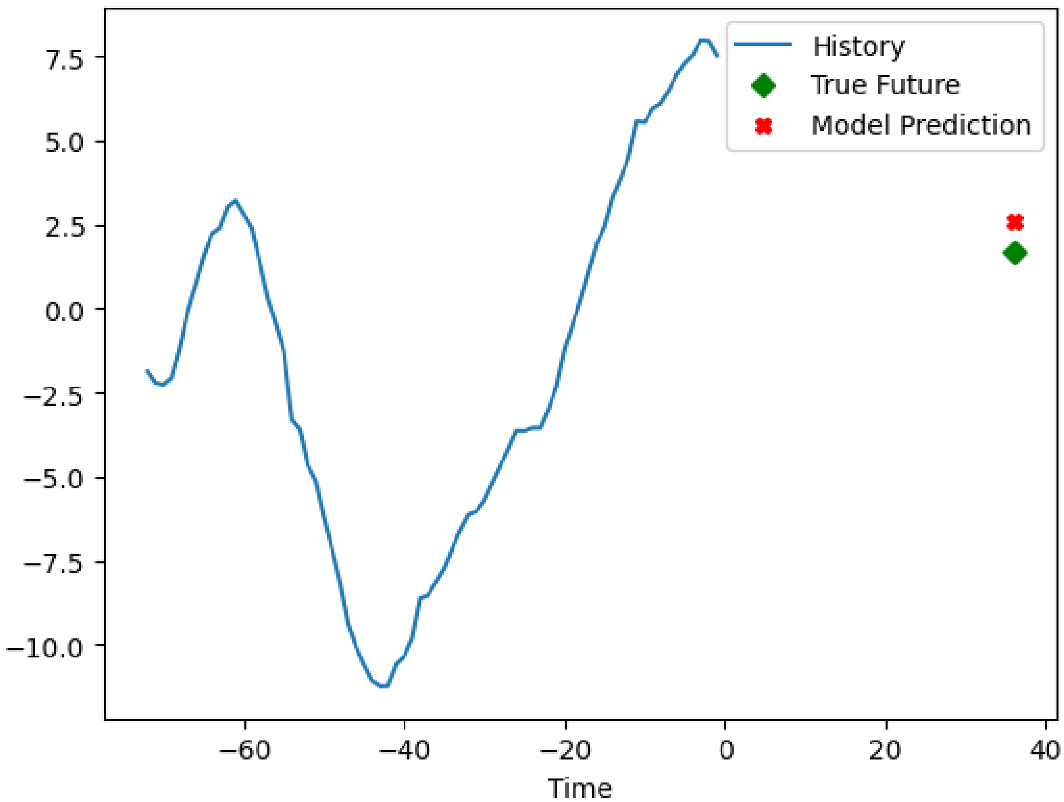
```
loss: 0.1514 - val_loss: 0.2404
Epoch 9/15
325/325            13s 40ms/step -
loss: 0.1447 - val_loss: 0.2398
Epoch 10/15
325/325            13s 40ms/step -
loss: 0.1385 - val_loss: 0.2441
Epoch 11/15
325/325            13s 39ms/step -
loss: 0.1334 - val_loss: 0.2546
Epoch 12/15
325/325            13s 40ms/step -
loss: 0.1296 - val_loss: 0.2554
Epoch 13/15
325/325            13s 39ms/step -
loss: 0.1267 - val_loss: 0.2445
Epoch 14/15
325/325            14s 42ms/step -
loss: 0.1254 - val_loss: 0.2764
Epoch 15/15
325/325            13s 41ms/step -
loss: 0.1255 - val_loss: 0.2507
```

[ ]:
```
loss = history.history["loss"]
epochs = range(len(loss))
plt.figure()
plt.plot(epochs, loss, "b", label="Training loss")
plt.title("Training Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```



[ ]:
```
temp_mean = temperature.mean(axis=0)
temp_std = temperature.std(axis=0)

for x, y in dataset_val.skip(12):
    history_data = x[0][:, 1].numpy() * temp_std + temp_mean
    true_value = y[0].numpy() * temp_std + temp_mean
    prediction = model.predict(x)[0] * temp_std + temp_mean
    time_steps = list(range(-(history_data.shape[0]), 0))
    plt.plot(time_steps, history_data)
    plt.plot(36, true_value, "gD")
    plt.plot(36, prediction, "rX")
    plt.legend(["History", "True Future", "Model Prediction"])
    plt.xlabel("Time")
    plt.show()
    break
```

```
2/2          0s 24ms/step
```



[ ]:
```python
import numpy as np

def lorenz96(x, F=8.0):
    N = len(x)
    dxdt = np.zeros(N)
    for i in range(N):
        dxdt[i] = (x[(i+1)%N] - x[i-2]) * x[i-1] - x[i] + F
    return dxdt

def rk4_step(x, dt, F=8.0):
    k1 = lorenz96(x, F)
    k2 = lorenz96(x + dt * k1 / 2.0, F)
    k3 = lorenz96(x + dt * k2 / 2.0, F)
    k4 = lorenz96(x + dt * k3, F)
    return x + dt / 6.0 * (k1 + 2*k2 + 2*k3 + k4)

# Generate data
N = 6  # 6 features like weather (temp, pressure, etc.)
steps = 10000
dt = 0.01
x = np.random.rand(N)
```

```
lorenz_data = []

for _ in range(steps):
    x = rk4_step(x, dt)
    lorenz_data.append(x.copy())

lorenz_data = np.array(lorenz_data)
```

[ ]:
```
import pandas as pd

lorenz_df = pd.DataFrame(lorenz_data, columns=['temp', 'press', 'hum', 'vp',␣
↪'wind', 'airtight'])

# Normalize like before
def normalize(data):
    data_mean = data.mean(axis=0)
    data_std = data.std(axis=0)
    return (data - data_mean) / data_std, data_mean, data_std

lorenz_norm, l_mean, l_std = normalize(lorenz_df)

train_len = int(0.8 * len(lorenz_norm))
lorenz_train = lorenz_norm[:train_len]
lorenz_val = lorenz_norm[train_len:]

# Create sequences
def create_seq(data, target_col=0, step=6, seq_len=72):
    x = []
    y = []
    #The original for loop iterated beyond the boundaries of the data
    #This was fixed to iterate through the acceptable boundaries
    for i in range(0, len(data) - seq_len * step - 1, step):
        x.append(data.iloc[i:i+seq_len*step:step].values) # Access data using .
↪iloc and .values
        y.append(data.iloc[i+seq_len*step, target_col]) # Access data using .
↪iloc
    return np.array(x), np.array(y)

x_train, y_train = create_seq(lorenz_train)
x_val, y_val = create_seq(lorenz_val)
```

[ ]:
```
from tensorflow import keras

model = keras.Sequential([
    keras.layers.Input(shape=(x_train.shape[1], x_train.shape[2])),
    keras.layers.LSTM(64, return_sequences=True),
    keras.layers.Dropout(0.2),
```

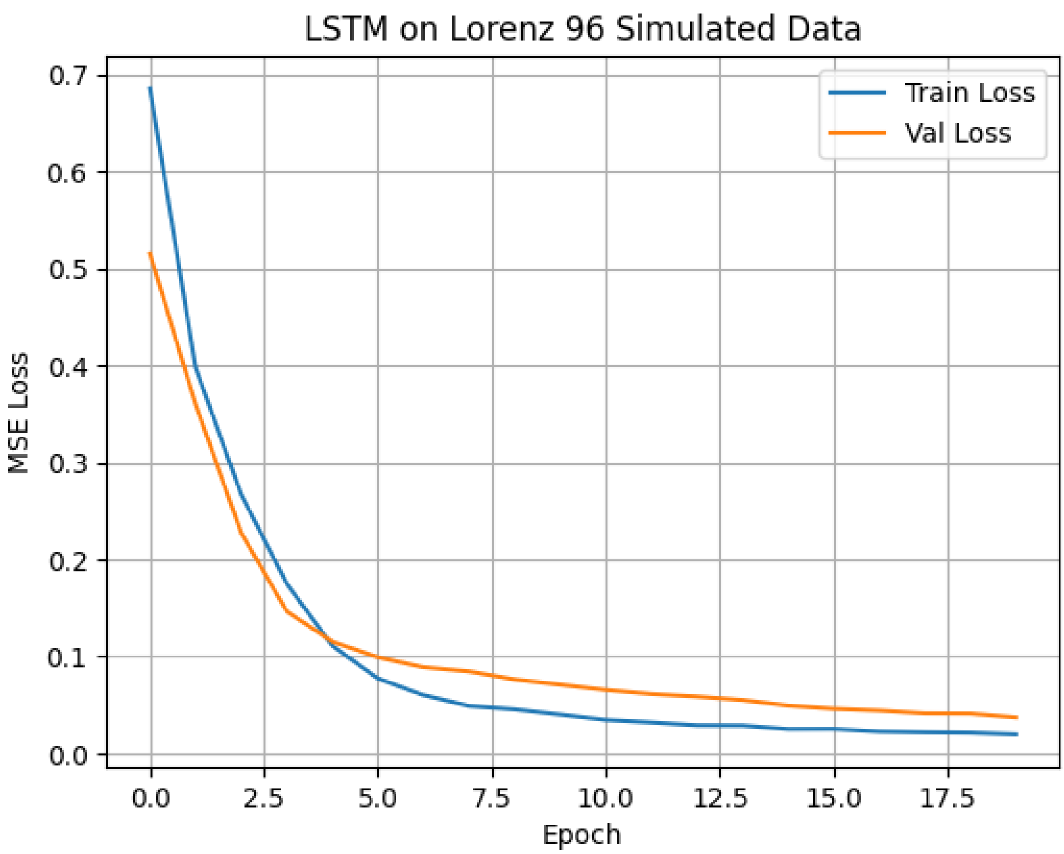
```
    keras.layers.LSTM(32),
    keras.layers.Dense(1)
])

model.compile(optimizer='adam', loss='mse')
model.summary()

history = model.fit(x_train, y_train, epochs=20, batch_size=64,␣
 ↪validation_data=(x_val, y_val))
```

```
Model: "sequential_1"


 Layer (type)                    Output Shape                  ␣
↪Param #

 lstm_2 (LSTM)                   (None, 72, 64)                 ␣
↪18,176

 dropout (Dropout)               (None, 72, 64)                    ␣
↪  0

 lstm_3 (LSTM)                   (None, 32)                     ␣
↪12,416

 dense_2 (Dense)                 (None, 1)                         ␣
↪ 33


 Total params: 30,625 (119.63 KB)

 Trainable params: 30,625 (119.63 KB)

 Non-trainable params: 0 (0.00 B)

Epoch 1/20
20/20                 6s 103ms/step -
loss: 0.8408 - val_loss: 0.5154
Epoch 2/20
20/20                 2s 81ms/step -
loss: 0.4377 - val_loss: 0.3601
Epoch 3/20
20/20                 3s 103ms/step -
loss: 0.2816 - val_loss: 0.2277
```

```
Epoch 4/20
20/20            2s 80ms/step -
loss: 0.1726 - val_loss: 0.1467
Epoch 5/20
20/20            4s 129ms/step -
loss: 0.1321 - val_loss: 0.1154
Epoch 6/20
20/20            2s 101ms/step -
loss: 0.0840 - val_loss: 0.0993
Epoch 7/20
20/20            2s 82ms/step -
loss: 0.0603 - val_loss: 0.0891
Epoch 8/20
20/20            3s 88ms/step -
loss: 0.0434 - val_loss: 0.0847
Epoch 9/20
20/20            3s 108ms/step -
loss: 0.0464 - val_loss: 0.0764
Epoch 10/20
20/20            3s 104ms/step -
loss: 0.0465 - val_loss: 0.0711
Epoch 11/20
20/20            2s 81ms/step -
loss: 0.0386 - val_loss: 0.0657
Epoch 12/20
20/20            3s 80ms/step -
loss: 0.0339 - val_loss: 0.0615
Epoch 13/20
20/20            2s 82ms/step -
loss: 0.0293 - val_loss: 0.0587
Epoch 14/20
20/20            2s 80ms/step -
loss: 0.0292 - val_loss: 0.0551
Epoch 15/20
20/20            3s 114ms/step -
loss: 0.0271 - val_loss: 0.0494
Epoch 16/20
20/20            2s 104ms/step -
loss: 0.0234 - val_loss: 0.0462
Epoch 17/20
20/20            2s 79ms/step -
loss: 0.0236 - val_loss: 0.0444
Epoch 18/20
20/20            3s 80ms/step -
loss: 0.0218 - val_loss: 0.0415
Epoch 19/20
20/20            2s 81ms/step -
loss: 0.0212 - val_loss: 0.0413
```

```
Epoch 20/20
20/20              3s 81ms/step -
loss: 0.0203 - val_loss: 0.0372
```

[ ]:
```python
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('LSTM on Lorenz 96 Simulated Data')
plt.xlabel('Epoch')
plt.ylabel('MSE Loss')
plt.legend()
plt.grid(True)
plt.show()
```



[ ]:
```python
# Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
```

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# 1. Lorenz 96 Dynamics and RK4 Solver
def lorenz96(x, F=8.0):
    N = len(x)
    dxdt = np.zeros(N)
    for i in range(N):
        dxdt[i] = (x[(i+1)%N] - x[i-2]) * x[i-1] - x[i] + F
    return dxdt

def rk4_step(x, dt, F=8.0):
    k1 = lorenz96(x, F)
    k2 = lorenz96(x + dt * k1 / 2.0, F)
    k3 = lorenz96(x + dt * k2 / 2.0, F)
    k4 = lorenz96(x + dt * k3, F)
    return x + dt / 6.0 * (k1 + 2*k2 + 2*k3 + k4)

# 2. Simulate Lorenz 96 data
N = 6
steps = 10000
x = np.random.rand(N)
lorenz_data = [x.copy()]
for _ in range(steps - 1):
    x = rk4_step(x, 0.01)
    lorenz_data.append(x.copy())
lorenz_df = pd.DataFrame(lorenz_data, columns=['temp', 'press', 'hum', 'vp', 'wind', 'airtight'])

# 3. Create synthetic real weather data
np.random.seed(0)
real_df = lorenz_df + np.random.normal(0, 0.5, lorenz_df.shape)

# 4. Normalize data
def normalize(data):
    mean = data.mean()
    std = data.std()
    return (data - mean) / std, mean, std

def create_seq(data, target_col, step=6, seq_len=72):
    x, y = [], []
    for i in range(len(data) - seq_len * step):
        x.append(data[i:i+seq_len*step:step].values)
        y.append(data.iloc[i+seq_len*step, target_col])
    return np.array(x), np.array(y)

# 5. LSTM Model Builder
```

```python
def build_lstm(input_shape):
    model = Sequential([
        LSTM(64, input_shape=input_shape),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse')
    return model

# 6. Train and Evaluate for temp, press, wind
results = {}
features = ['temp', 'press', 'wind']

for feature in features:
    feature_idx = lorenz_df.columns.get_loc(feature)

    # Normalize
    lorenz_norm, lorenz_mean, lorenz_std = normalize(lorenz_df)
    real_norm, real_mean, real_std = normalize(real_df)

    # Create sequences
    x_lorenz, y_lorenz = create_seq(lorenz_norm, feature_idx)
    x_real, y_real = create_seq(real_norm, feature_idx)

    # Train-test split
    split = int(0.8 * len(x_lorenz))
    x_l_train, y_l_train = x_lorenz[:split], y_lorenz[:split]
    x_l_test, y_l_test = x_lorenz[split:], y_lorenz[split:]
    x_r_train, y_r_train = x_real[:split], y_real[:split]
    x_r_test, y_r_test = x_real[split:], y_real[split:]

    # Model training for Lorenz
    model_l = build_lstm((x_l_train.shape[1], x_l_train.shape[2]))
    hist_l = model_l.fit(x_l_train, y_l_train, epochs=10, batch_size=64,␣
 ↪verbose=0)
    pred_l = model_l.predict(x_l_test)
    rmse_l = np.sqrt(mean_squared_error(y_l_test, pred_l))

    # Model training for Real
    model_r = build_lstm((x_r_train.shape[1], x_r_train.shape[2]))
    hist_r = model_r.fit(x_r_train, y_r_train, epochs=10, batch_size=64,␣
 ↪verbose=0)
    pred_r = model_r.predict(x_r_test)
    rmse_r = np.sqrt(mean_squared_error(y_r_test, pred_r))

    # Store results
    results[feature] = {
        'rmse_lorenz': rmse_l,
```

```
        'rmse_real': rmse_r,
        'y_true': y_r_test,
        'pred_real': pred_r.flatten(),
        'pred_lorenz': pred_l.flatten(),
        'loss_l': hist_l.history['loss'],
        'loss_r': hist_r.history['loss']
    }

# 7. Plotting Results
for feature in features:
    res = results[feature]
    plt.figure(figsize=(14, 5))
    plt.plot(res['y_true'], label='True')
    plt.plot(res['pred_real'], label='Real LSTM')
    plt.plot(res['pred_lorenz'], label='Lorenz LSTM')
    plt.title(f"{feature.upper()} Prediction (RMSE Real: {res['rmse_real']:.
↪4f}, Lorenz: {res['rmse_lorenz']:.4f})")
    plt.xlabel('Time step')
    plt.ylabel(feature)
    plt.legend()
    plt.grid(True)
    plt.show()

    plt.figure()
    plt.plot(res['loss_r'], label='Real Train Loss')
    plt.plot(res['loss_l'], label='Lorenz Train Loss')
    plt.title(f"{feature.upper()} - Training Loss")
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.show()


from mpl_toolkits.mplot3d import Axes3D

# Pick feature indices
temp_idx = lorenz_df.columns.get_loc('temp')
press_idx = lorenz_df.columns.get_loc('press')
wind_idx = lorenz_df.columns.get_loc('wind')

# Get true and predicted values from the results
true_temp = results['temp']['y_true']
true_press = results['press']['y_true']
true_wind = results['wind']['y_true']

pred_real_temp = results['temp']['pred_real']
```

```
pred_real_press = results['press']['pred_real']
pred_real_wind = results['wind']['pred_real']

pred_lorenz_temp = results['temp']['pred_lorenz']
pred_lorenz_press = results['press']['pred_lorenz']
pred_lorenz_wind = results['wind']['pred_lorenz']

# Plot all in a single 3D graph
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot true values - RED
ax.plot(true_temp, true_press, true_wind, color='red', label='True (Real)',␣
↪linewidth=1.2)

# Plot LSTM on real data - GREEN
ax.plot(pred_real_temp, pred_real_press, pred_real_wind, color='green',␣
↪label='LSTM (Real Data)', linewidth=1.0)

# Plot LSTM on Lorenz data - BLACK
ax.plot(pred_lorenz_temp, pred_lorenz_press, pred_lorenz_wind, color='black',␣
↪label='LSTM (Lorenz Data)', linewidth=1.0)

# Labels and legend
ax.set_xlabel("Temperature")
ax.set_ylabel("Pressure")
ax.set_zlabel("Wind")
ax.set_title("3D Trajectory: True vs Predicted (All Features)")
ax.legend()
plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200:
UserWarning: 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)
```

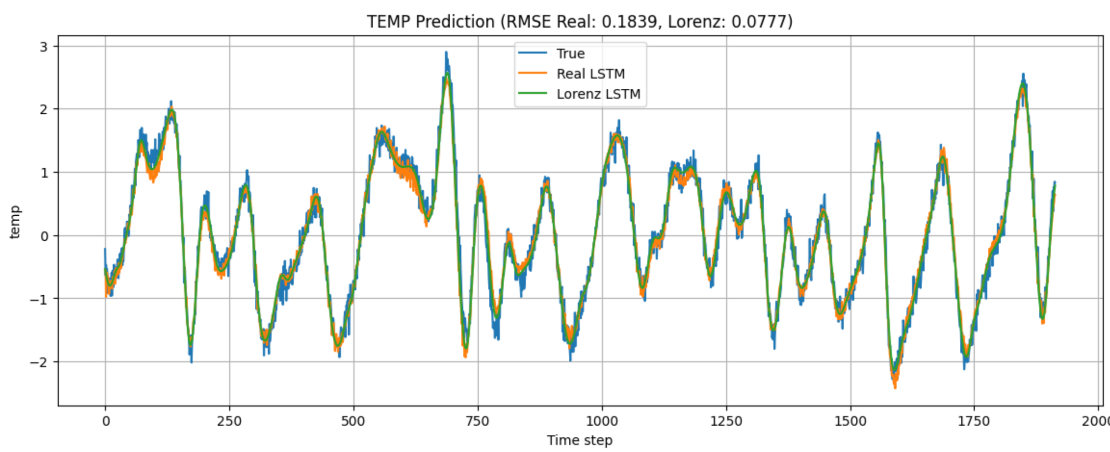
```
60/60              1s 13ms/step
```

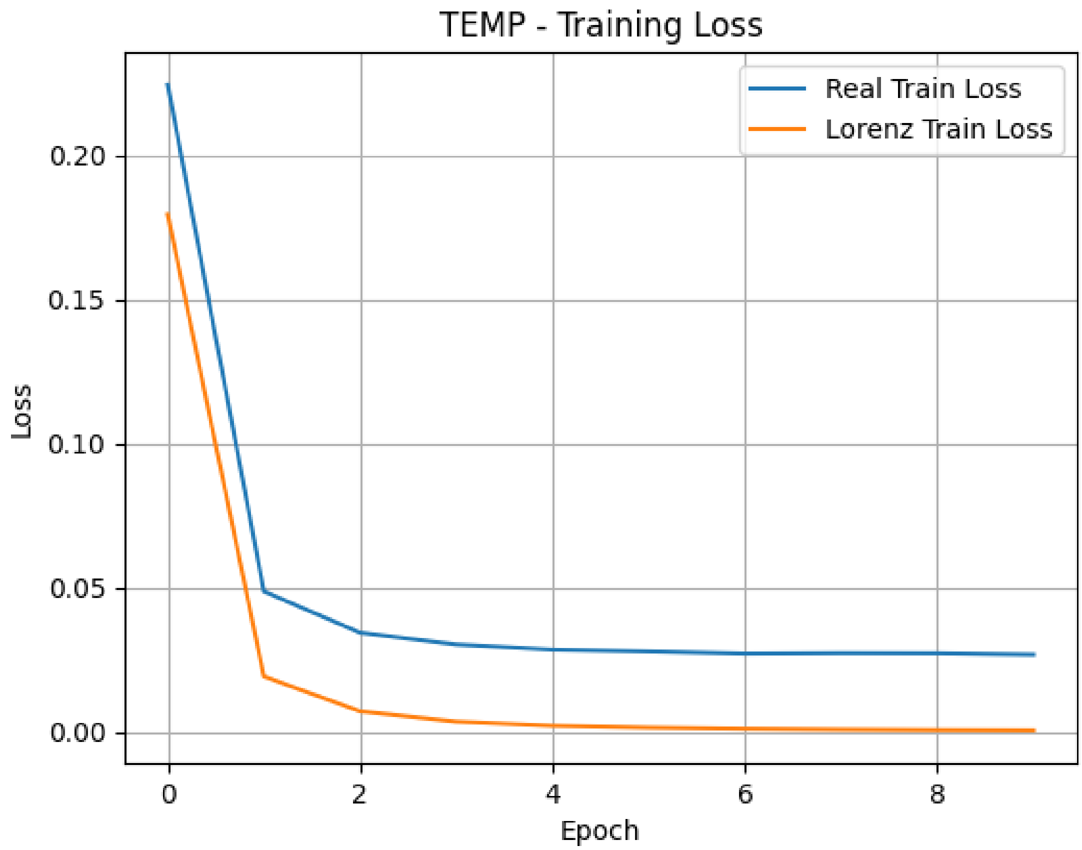
```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(**kwargs)
```

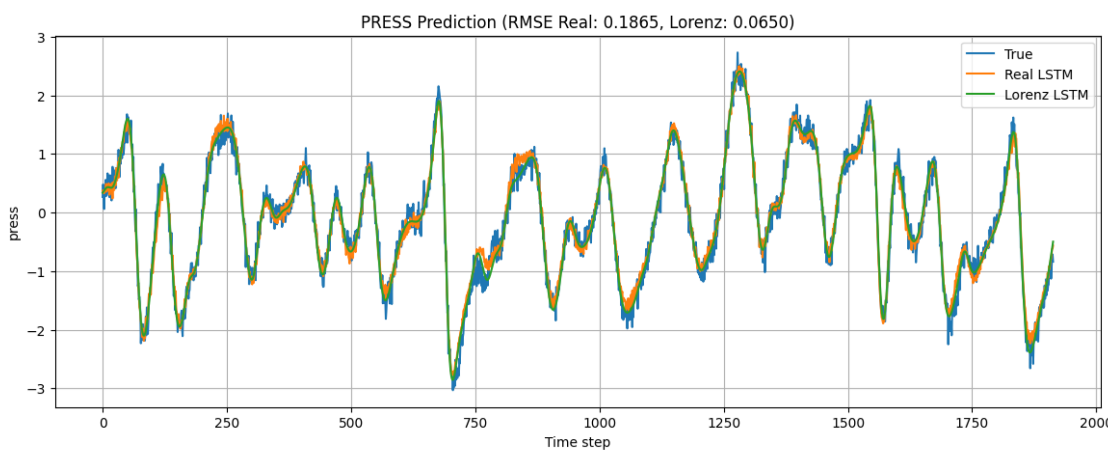
```
60/60              1s 18ms/step
```

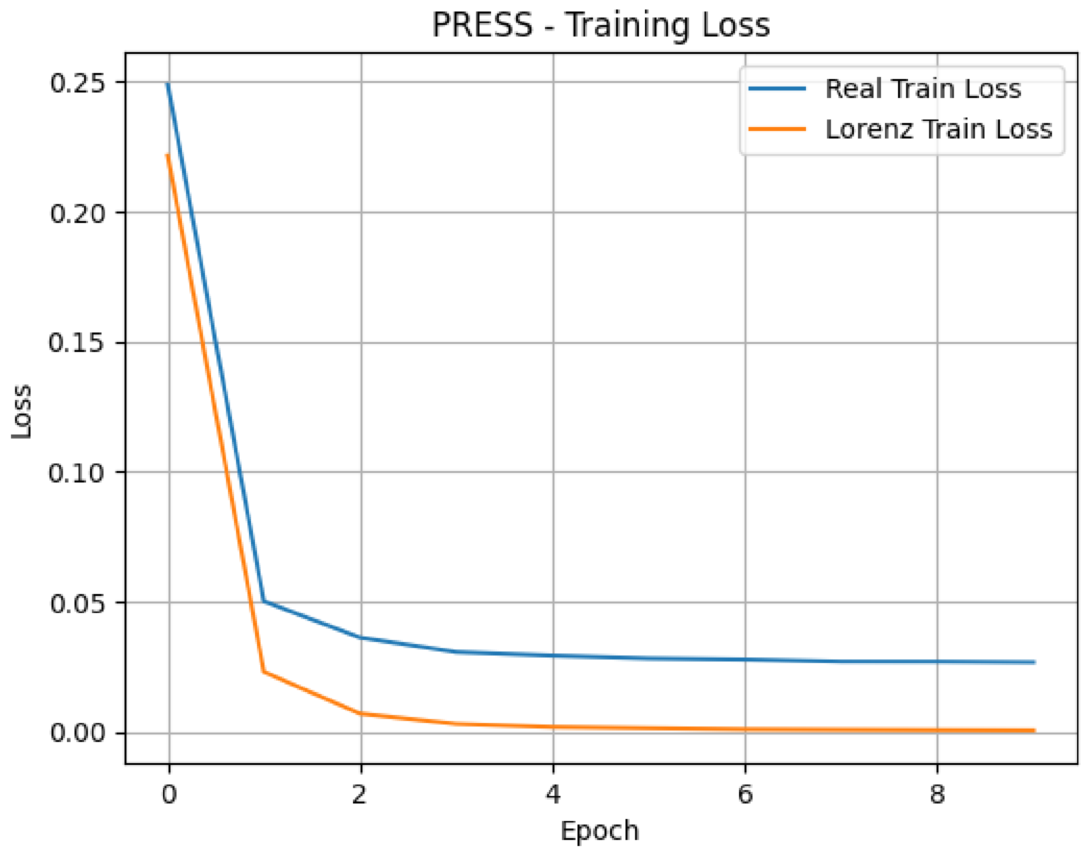
```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200:
```

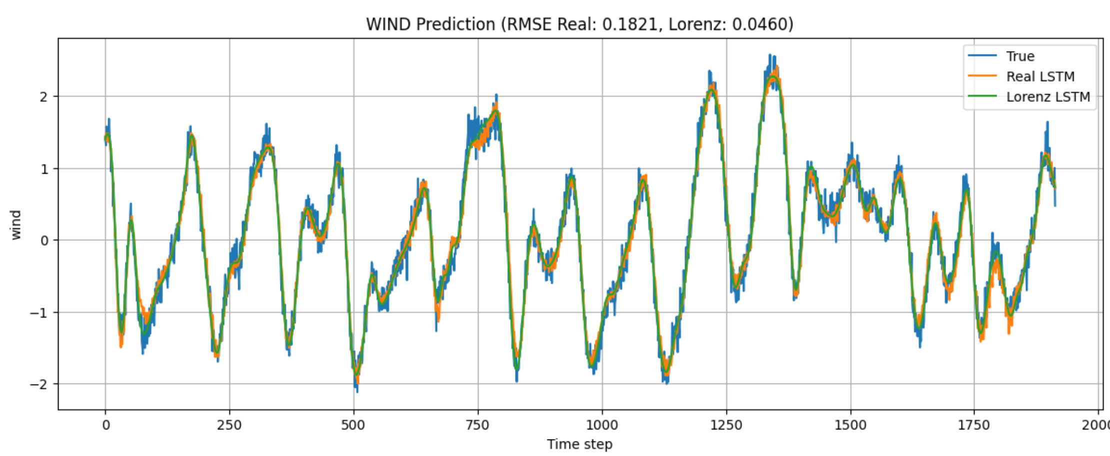
```
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(**kwargs)

60/60              1s 12ms/step

/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200:
UserWarning: 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)

60/60              1s 13ms/step

/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200:
UserWarning: 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)

60/60              1s 13ms/step

/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200:
UserWarning: 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)

60/60              1s 13ms/step
```

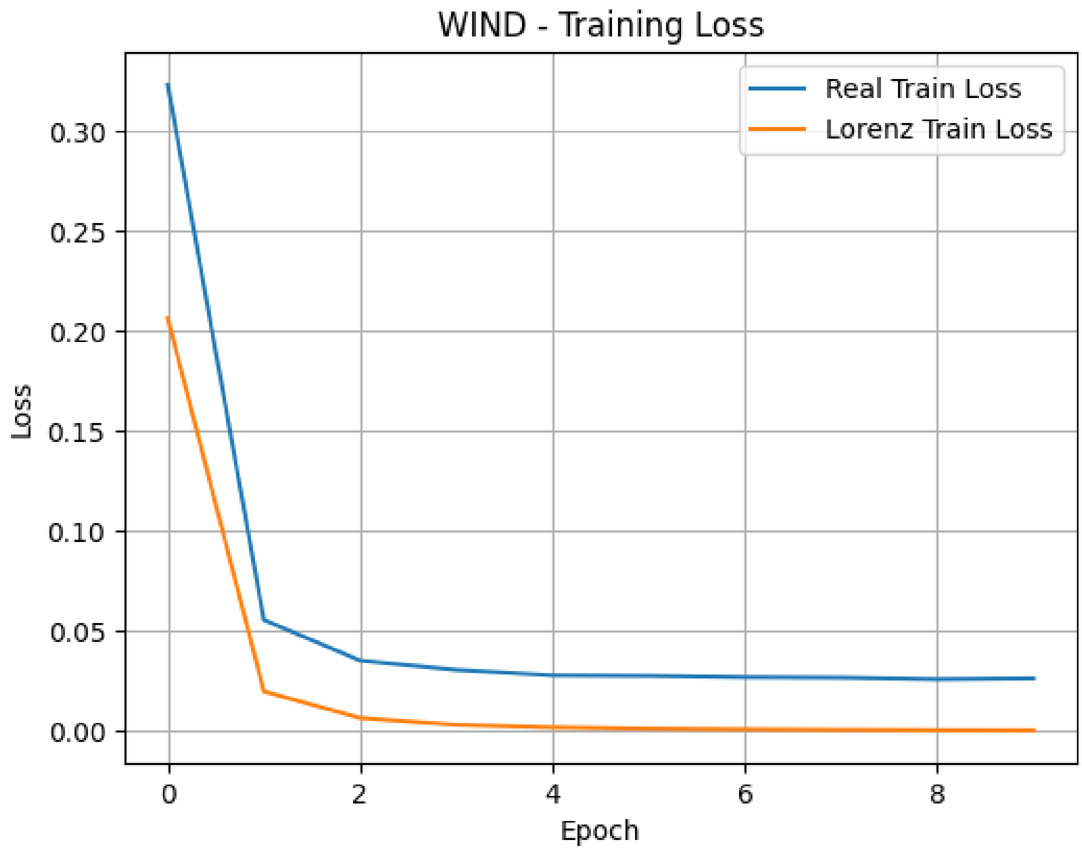


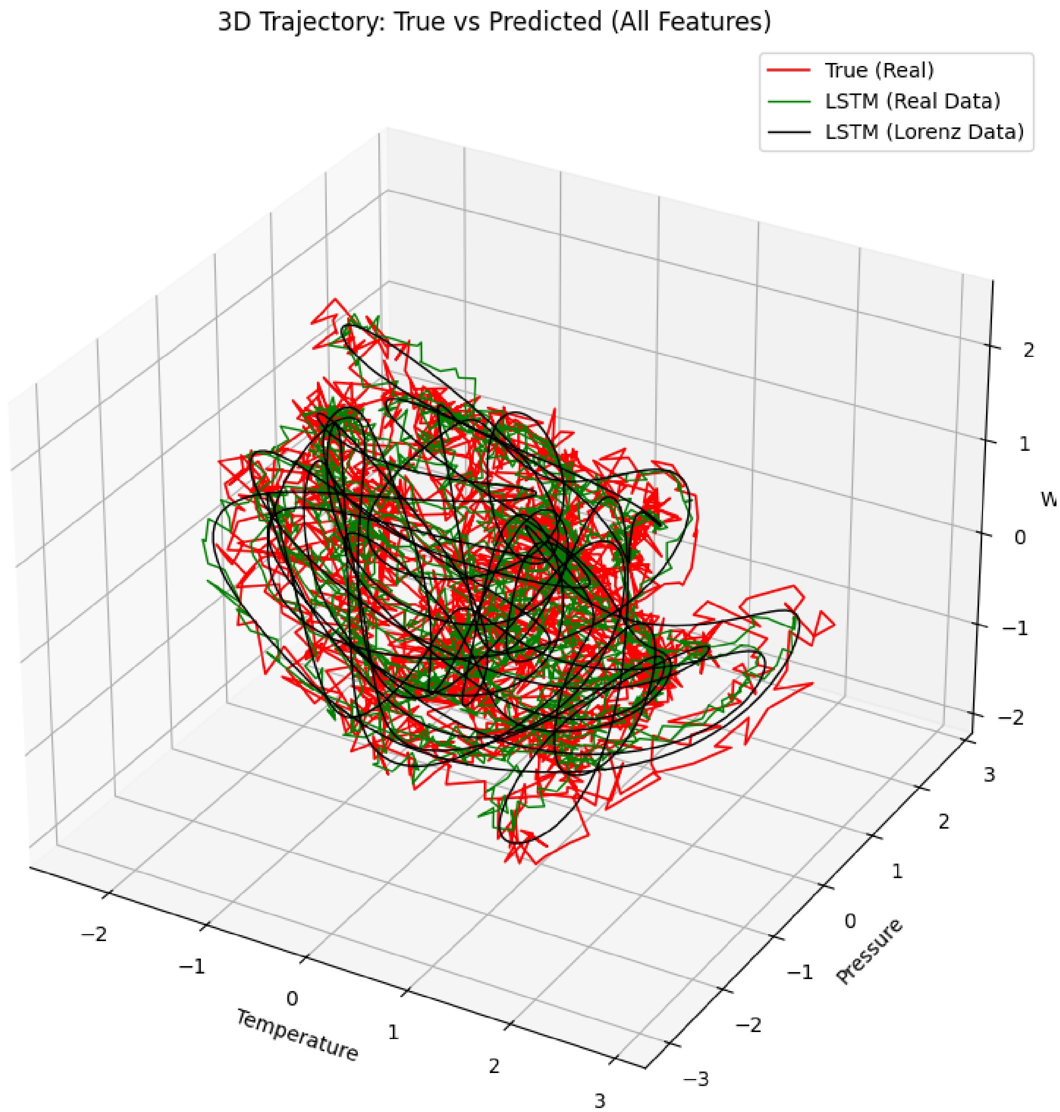

TEMP - Training Loss
Real Train Loss
Lorenz Train Loss
Loss
Epoch
0.00
0.05
0.10
0.15
0.20
0
2
4
6
8


PRESS Prediction (RMSE Real: 0.1865, Lorenz: 0.0650)
True
Real LSTM
Lorenz LSTM
press
Time step
0
250
500
750
1000
1250
1500
1750
2000


PRESS - Training Loss
Real Train Loss
Lorenz Train Loss
Loss
Epoch
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8


WIND Prediction (RMSE Real: 0.1821, Lorenz: 0.0460)
True
Real LSTM
Lorenz LSTM
wind
Time step
2
1
0
−1
−2
0
250
500
750
1000
1250
1500
1750
2000


WIND - Training Loss
Real Train Loss
Lorenz Train Loss
Loss
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8
Epoch


3D Trajectory: True vs Predicted (All Features)
True (Real)
LSTM (Real Data)
LSTM (Lorenz Data)
Temperature
Pressure
W
−2
−1
0
1
2
3
−3
−2
−1
0
1
2
3
−2
−1
0
1
2