Training and Inference

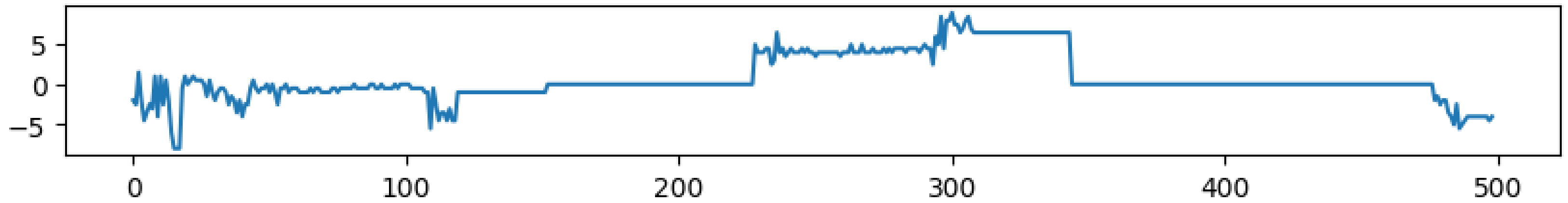
In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt

import math
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler

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

In [2]:
```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

import logging
logging.getLogger('tensorflow').setLevel(logging.ERROR)

import warnings
warnings.filterwarnings('ignore')
```

Import data

In [3]:
```
TRAIN_EPOCHS = 100
NUM_FEATURES = 1 # univariate time series, SINR or PHR or dlBler
TIME_STEP = 100  # Number of past time steps to use
BATCH_SIZE = 1 # it works for time series

INPUT_WIDTH = 64
LABEL_WIDTH = 64
PREDICTION_STEPS = 64 # prediction steps into the future
```

In [4]:
```
df_raw = pd.read_csv('data_one_feat_sinr_clean.csv', header=None)
plt.figure(figsize=(10, 1))
plot_features = df_raw[0][1:500]
plt.plot(range(len(plot_features)), plot_features)
plt.show()
```



In [5]:
```
df = df_raw.copy()
df = df[1:] # full data
```

```python
print(df.shape)
print(df.head())
```

```
(1717, 1)
     0
1 -2.0
2 -2.5
3  1.5
4 -2.0
5 -4.5
```

In [6]:
```python
# Scale the data to the range [0, 1]
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(df.values)

# Define the function to create the dataset
def create_dataset(dataset, time_step=1):
    dataX, dataY = [], []
    for i in range(len(dataset) - time_step - 1):
        a = dataset[i:(i + time_step), 0]
        dataX.append(a)
        dataY.append(dataset[i + time_step, 0])
    return np.array(dataX), np.array(dataY)

# Split the data into training and testing sets
training_size = int(len(scaled_data) * 0.80)
test_size = len(scaled_data) - training_size
train_data, test_data = scaled_data[0:training_size, :], scaled_data[training_size:len(scaled_data), :]

# Create the datasets for training and testing
X_train, y_train = create_dataset(train_data, TIME_STEP)
X_test, y_test = create_dataset(test_data, TIME_STEP)

print(X_train.shape, y_train.shape)

# Reshape the input to be [samples, time steps, features] which is required for LSTM
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

print(X_train.shape, y_train.shape)
print(scaled_data.shape)
```

```
(1272, 100) (1272,)
(1272, 100, 1) (1272,)
(1717, 1)
```

In [7]:
```python
# Build the LSTM model
model = Sequential(name='LSTM_model')
model.add(LSTM(100, return_sequences=True, input_shape=(TIME_STEP, NUM_FEATURES)))
model.add(LSTM(50, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam',
              loss='mean_squared_error')

model.summary()
```

Model: "LSTM_model"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 100, 100)	40,800
lstm_1 (LSTM)	(None, 50)	30,200
dense (Dense)	(None, 25)	1,275
dense_1 (Dense)	(None, 1)	26

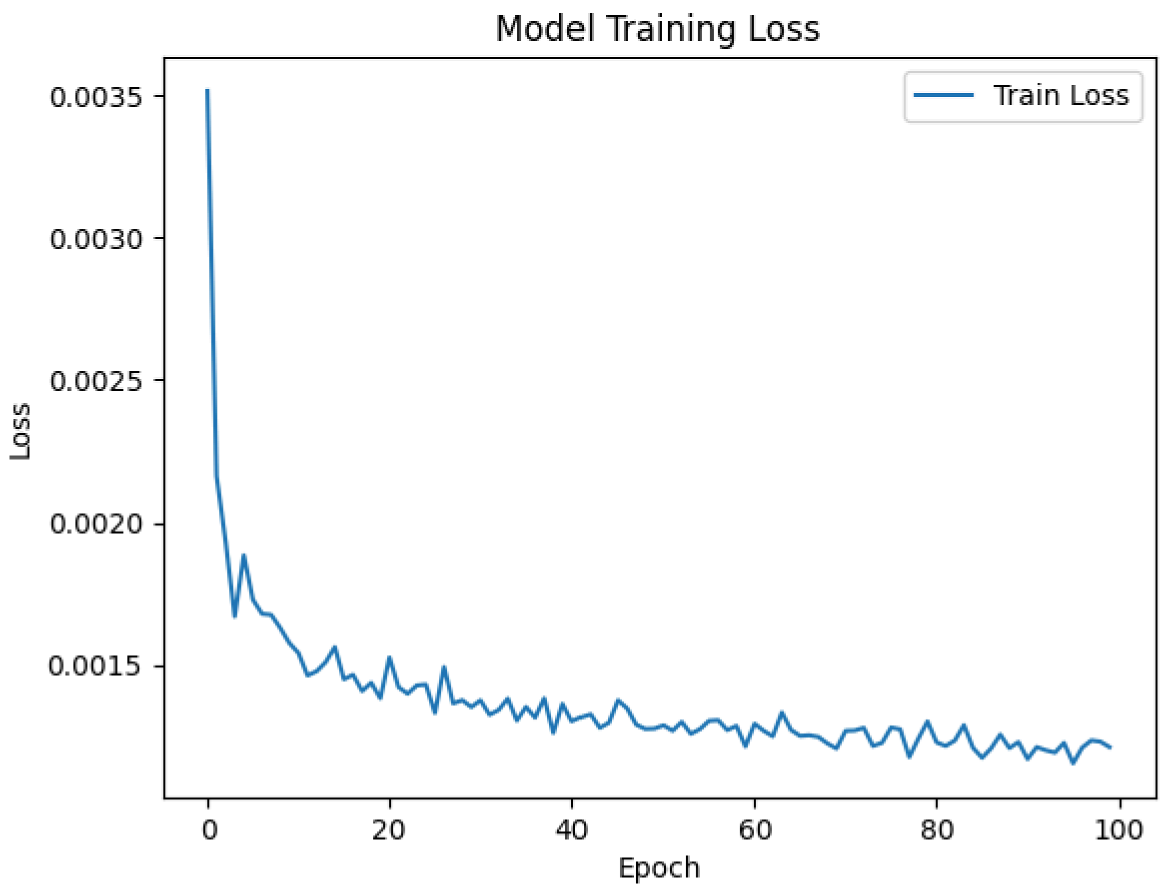
Total params: 72,301 (282.43 KB)

Trainable params: 72,301 (282.43 KB)

Non-trainable params: 0 (0.00 B)

In [8]:

```
# Train the model and save the history
history = model.fit(X_train, y_train,
                    batch_size=BATCH_SIZE,
                    epochs=TRAIN_EPOCHS,
                    verbose=0)

# Plot the training loss
plt.plot(history.history['loss'], label='Train Loss')
plt.title('Model Training Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



In [9]:

```
# Convert history.history (dict) to DataFrame
history_df = pd.DataFrame(history.history)
```

```
# Save to CSV
history_df.to_csv("training_history.csv", index=False)
```

In [10]:
```
# Reload history
loaded_history = pd.read_csv("training_history.csv")
```

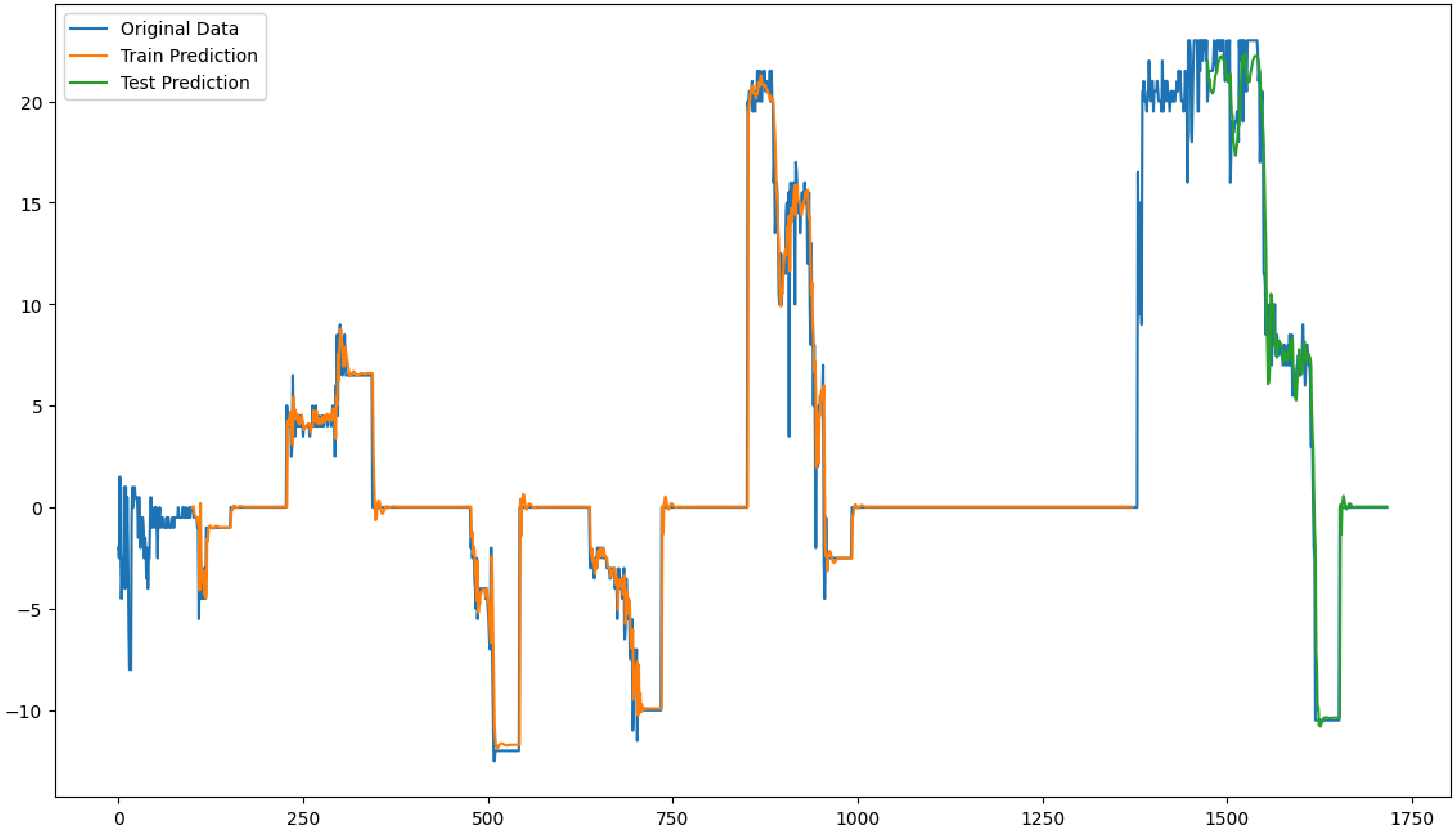
In [11]:
```
model.save("lstm_trained.keras")
```

Inference

In [12]:
```
model = tf.keras.models.load_model('./lstm_trained.keras')
```

In [13]:
```
# Make predictions
train_predict = model.predict(X_train, verbose=0)
test_predict = model.predict(X_test, verbose=0)

# Transform back to original form (if your data was scaled)
train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
y_train_inv = scaler.inverse_transform(y_train.reshape(-1, 1))
y_test_inv = scaler.inverse_transform(y_test.reshape(-1, 1))

# Calculate RMSE
train_rmse = math.sqrt(mean_squared_error(y_train_inv, train_predict))
test_rmse = math.sqrt(mean_squared_error(y_test_inv, test_predict))

print(f'Train RMSE: {train_rmse}')
print(f'Test RMSE: {test_rmse}')

# Plot the results
# Shift train predictions for plotting
train_predict_plot = np.empty_like(scaled_data)
train_predict_plot[:, :] = np.nan
train_predict_plot[TIME_STEP:len(train_predict) + TIME_STEP, :] = train_predict

# Shift test predictions for plotting
test_predict_plot = np.empty_like(scaled_data)
test_predict_plot[:, :] = np.nan
test_predict_plot[len(train_predict) + (TIME_STEP * 2) + 1:len(scaled_data) - 1, :] = test_predict

# Plot baseline and predictions
plt.figure(figsize=(14, 8))
plt.plot(scaler.inverse_transform(scaled_data), label='Original Data')
plt.plot(train_predict_plot, label='Train Prediction')
plt.plot(test_predict_plot, label='Test Prediction')
plt.legend()
plt.show()
```

```
Train RMSE: 1.130489515820426
Test RMSE: 1.5848351963949638
```



Original Data
Train Prediction
Test Prediction
20
15
10
5
0
−5
−10
0
250
500
750
1000
1250
1500
1750