# Training and Inference

In [104…

```
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

from tensorflow.keras import mixed_precision
```

In [105…

```
# Check if GPU is available
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
print("GPU devices:", tf.config.list_physical_devices('GPU'))

# Enable memory growth to prevent TensorFlow from allocating all GPU memory
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        print(f"GPU is available and will be used")
    except RuntimeError as e:
        print(e)
```

```
Num GPUs Available:  1
GPU devices: [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
GPU is available and will be used
```

In [106…

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
```

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

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

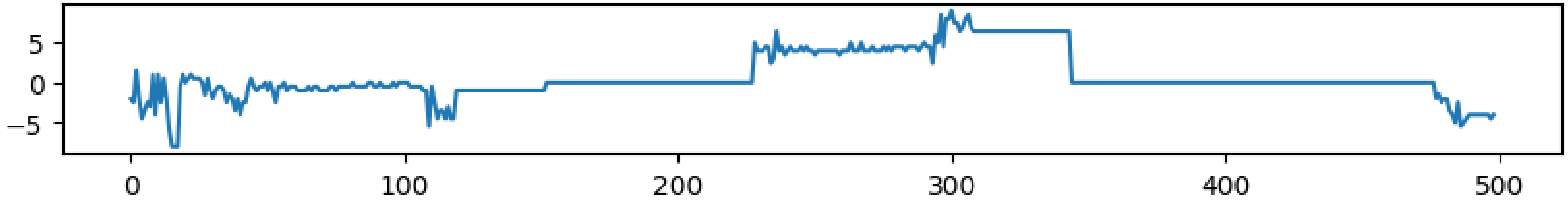
# Import data

In [107…

```
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 = 16 # it works for time series

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

In [108…

```
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 [109…

```
df = df_raw.copy()
df = df[1:] # full data
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 [110…

```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 [111…

```
# Create TensorFlow datasets for optimized GPU training
BATCH_SIZE = 1  # Keep the batch size as one for better forecasting accuracy.
                # A higher batch size results in lower accuracy.
BUFFER_SIZE = 10000  # For shuffling

# Training dataset with optimizations
train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train))
train_dataset = (train_dataset
                 .cache()  # Cache the dataset in memory
                 .shuffle(BUFFER_SIZE)  # Shuffle for better training
                 .batch(BATCH_SIZE)  # Batch the data
                 .prefetch(tf.data.AUTOTUNE))  # Prefetch for pipeline optimization

# Test dataset (no shuffling needed)
test_dataset = tf.data.Dataset.from_tensor_slices((X_test, y_test))
test_dataset = (test_dataset
                .batch(BATCH_SIZE)
                .cache()
                .prefetch(tf.data.AUTOTUNE))

# Verify dataset
print("\nDataset Information:")
print(f"Train batches: {tf.data.experimental.cardinality(train_dataset).numpy()}")
print(f"Test batches: {tf.data.experimental.cardinality(test_dataset).numpy()}")

# Get one batch to verify shape
for batch_x, batch_y in train_dataset.take(1):
    print(f"\nBatch shape - X: {batch_x.shape}, y: {batch_y.shape}")
```

```
Dataset Information:
Train batches: 1272
Test batches: 243

Batch shape - X: (1, 100, 1), y: (1,)
```

In [112…

```
# Split training data into train and validation
val_split = 0.2
val_size = int(len(X_train) * val_split)
train_size = len(X_train) - val_size

X_train_split = X_train[:train_size]
y_train_split = y_train[:train_size]
```

```python
X_val = X_train[train_size:]
y_val = y_train[train_size:]

# Create validation dataset
val_dataset = tf.data.Dataset.from_tensor_slices((X_val, y_val))
val_dataset = (val_dataset
               .batch(BATCH_SIZE)
               .cache()
               .prefetch(tf.data.AUTOTUNE))

# Update training dataset with split data
train_dataset_split = tf.data.Dataset.from_tensor_slices((X_train_split, y_train_split))
train_dataset_split = (train_dataset_split
                       .cache()
                       .shuffle(BUFFER_SIZE)
                       .batch(BATCH_SIZE)
                       .prefetch(tf.data.AUTOTUNE))

print(f"\nWith validation split:")
print(f"Training samples: {train_size}")
print(f"Validation samples: {val_size}")
print(f"Test samples: {len(X_test)}")
```

```
With validation split:
Training samples: 1018
Validation samples: 254
Test samples: 243
```

In [113…

```python
# Build 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, dtype='float32'))  # Output layer uses float32 for mixed precision

model.compile(optimizer='adam', loss='mean_squared_error')
```

In [114…

```python
model.summary()
```

**Model: "LSTM_model"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| lstm_14 (LSTM) | (None, 100, 100) | 40,800 |
| lstm_15 (LSTM) | (None, 50) | 30,200 |
| dense_14 (Dense) | (None, 25) | 1,275 |
| dense_15 (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)

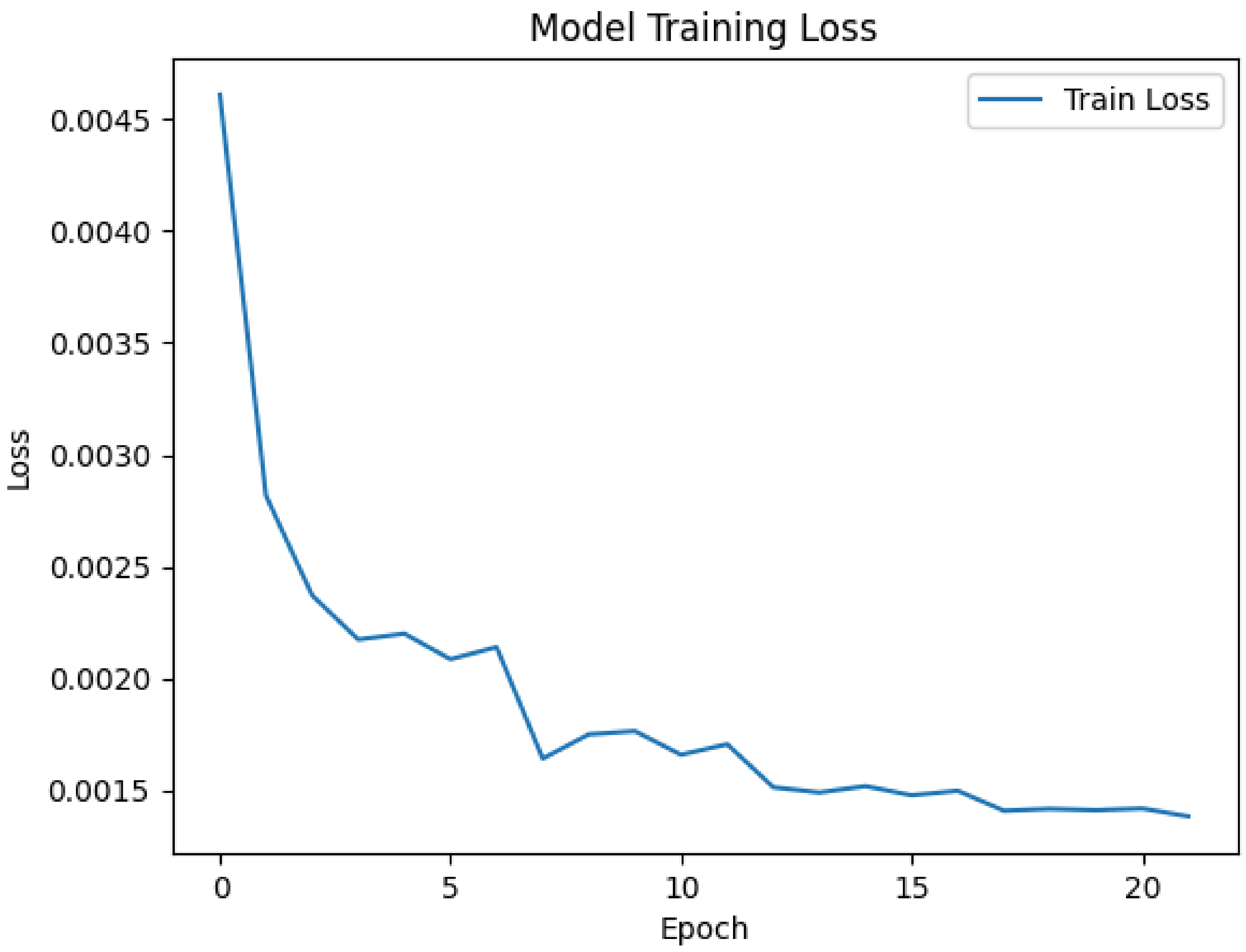
In [115…

```
# Now train the model with the optimized dataset
history = model.fit(
    train_dataset_split,
    validation_data=val_dataset,
    epochs=TRAIN_EPOCHS,
    verbose=1,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            patience=10,
            restore_best_weights=True
        ),
        tf.keras.callbacks.ReduceLROnPlateau(
            monitor='val_loss',
            factor=0.5,
            patience=5,
            min_lr=1e-7
        )
    ]
)
```

```
Epoch 1/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 14s 11ms/step - loss: 0.0095 - val_loss: 0.0011 - learning_rate: 0.0010
Epoch 2/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 0.0029 - val_loss: 8.7941e-06 - learning_rate: 0.0010
Epoch 3/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 0.0019 - val_loss: 5.6772e-04 - learning_rate: 0.0010
Epoch 4/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 12s 11ms/step - loss: 0.0031 - val_loss: 4.9511e-05 - learning_rate: 0.0010
Epoch 5/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 0.0015 - val_loss: 1.0977e-04 - learning_rate: 0.0010
Epoch 6/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 0.0024 - val_loss: 3.1464e-04 - learning_rate: 0.0010
Epoch 7/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 0.0021 - val_loss: 0.0027 - learning_rate: 0.0010
Epoch 8/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 0.0014 - val_loss: 1.4327e-04 - learning_rate: 5.0000e-04
Epoch 9/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 0.0016 - val_loss: 1.0030e-04 - learning_rate: 5.0000e-04
Epoch 10/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 0.0015 - val_loss: 3.0681e-05 - learning_rate: 5.0000e-04
Epoch 11/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 0.0013 - val_loss: 7.5110e-05 - learning_rate: 5.0000e-04
Epoch 12/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 0.0012 - val_loss: 1.6991e-07 - learning_rate: 5.0000e-04
Epoch 13/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 0.0012 - val_loss: 1.7247e-04 - learning_rate: 2.5000e-04
Epoch 14/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 0.0015 - val_loss: 3.6352e-05 - learning_rate: 2.5000e-04
Epoch 15/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 0.0014 - val_loss: 1.4025e-06 - learning_rate: 2.5000e-04
Epoch 16/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 0.0012 - val_loss: 1.6583e-05 - learning_rate: 2.5000e-04
Epoch 17/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 9.5499e-04 - val_loss: 8.1182e-06 - learning_rate: 2.5000e-04
Epoch 18/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 0.0015 - val_loss: 9.8699e-06 - learning_rate: 1.2500e-04
Epoch 19/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 0.0012 - val_loss: 1.7129e-06 - learning_rate: 1.2500e-04
Epoch 20/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 0.0015 - val_loss: 1.3862e-05 - learning_rate: 1.2500e-04
Epoch 21/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 0.0014 - val_loss: 7.8471e-06 - learning_rate: 1.2500e-04
```

```
Epoch 22/100
1018/1018 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 0.0011 - val_loss: 4.6818e-05 - learning_rate: 1.2500e-04
```

In [116…
```python
# 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 [117…
```python
# Convert history.history (dict) to DataFrame
history_df = pd.DataFrame(history.history)
```

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

In [118…

```python
# Reload history
loaded_history = pd.read_csv("training_history.csv")
```

In [119…

```python
model.save("lstm_trained.keras")
```

# Inference

In [120…

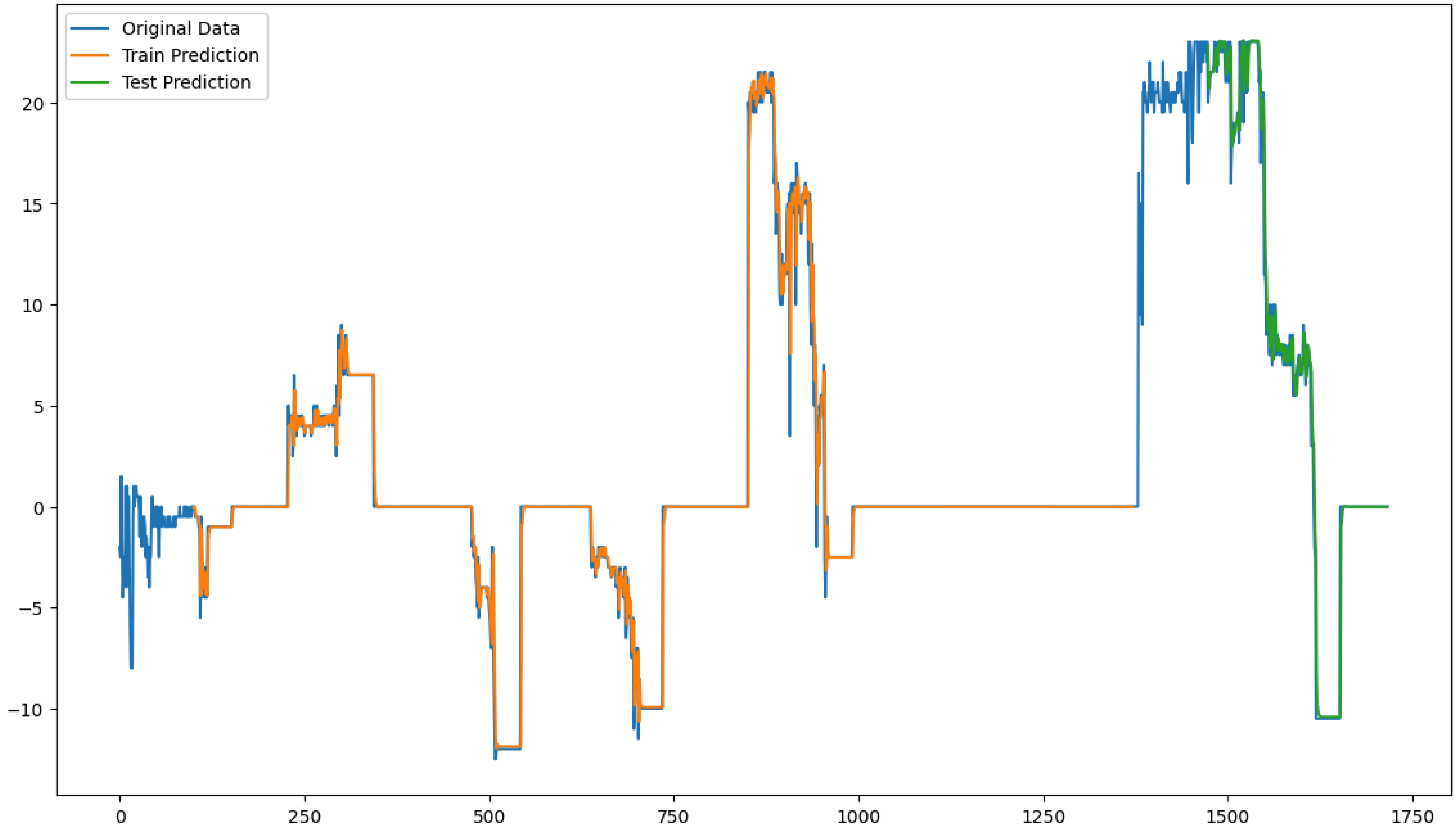
```python
model = tf.keras.models.load_model('./lstm_trained.keras')
```

In [121…

```python
# 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
```

```python
# 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: 1.1777889014455691
Test RMSE: 1.4419201534508579
```


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