Module_4_LSTM_Complete_(1)

February 2, 2026

1 Module 4: LSTM for Household Electricity Consumption Prediction

1.1 Assignment Overview

IoT applications are often based on time series data collected by sensors in the IoT system. This means that the best Deep Learning models will often be designed with sequential data in mind, such as RNNs and LSTMs.

In this assignment, you will: 1. Build an LSTM model to predict Global Active Power from household electricity consumption data 2. Make changes to improve the original LSTM model's performance 3. Analyze the performance and compare with linear regression from previous assignments

1.2 Recommended Reading

- A Beginner's Guide on Recurrent Neural Networks with PyTorch
- The Unreasonable Effectiveness of Recurrent Neural Networks
- Generating Sequences With Recurrent Neural Networks

1.3 1. Import Libraries and Setup

[1]:
```python
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')

# Deep Learning libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

```
# Sklearn utilities
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split

# Set random seeds for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

# Display settings
plt.style.use('seaborn-v0_8-darkgrid')
pd.set_option('display.max_columns', None)

print(f"TensorFlow version: {tf.__version__}")
print(f"Keras version: {keras.__version__}")
```

```
TensorFlow version: 2.19.0
Keras version: 3.10.0
```

1.4 2. Data Loading and Initial Exploration

We'll use the household electricity consumption dataset. This dataset contains measurements of electric power consumption in one household with a one-minute sampling rate over a period of almost 4 years.

[2]:
```
# Download the dataset
!wget -q https://archive.ics.uci.edu/ml/machine-learning-databases/00235/
  ↪household_power_consumption.zip
!unzip -q household_power_consumption.zip

# Load the data
df = pd.read_csv('household_power_consumption.txt',
                 sep=';',
                 parse_dates={'datetime': ['Date', 'Time']},
                 infer_datetime_format=True,
                 low_memory=False,
                 na_values=['?'])

# Set datetime as index
df.set_index('datetime', inplace=True)

print(f"Dataset shape: {df.shape}")
print(f"\nDataset info:")
print(df.info())
print(f"\nFirst few rows:")
df.head()
```

```
Dataset shape: (2075259, 7)

Dataset info:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2075259 entries, 2006-12-16 17:24:00 to 2010-11-26 21:02:00
Data columns (total 7 columns):
 #   Column                 Dtype
---  ------                 -----
 0   Global_active_power    float64
 1   Global_reactive_power  float64
 2   Voltage                float64
 3   Global_intensity       float64
 4   Sub_metering_1         float64
 5   Sub_metering_2         float64
 6   Sub_metering_3         float64
dtypes: float64(7)
memory usage: 126.7 MB
None

First few rows:
```

[2]:

```
                     Global_active_power  Global_reactive_power  Voltage  \
datetime
2006-12-16 17:24:00                4.216                  0.418   234.84
2006-12-16 17:25:00                5.360                  0.436   233.63
2006-12-16 17:26:00                5.374                  0.498   233.29
2006-12-16 17:27:00                5.388                  0.502   233.74
2006-12-16 17:28:00                3.666                  0.528   235.68

                     Global_intensity  Sub_metering_1  Sub_metering_2  \
datetime
2006-12-16 17:24:00              18.4             0.0             1.0
2006-12-16 17:25:00              23.0             0.0             1.0
2006-12-16 17:26:00              23.0             0.0             2.0
2006-12-16 17:27:00              23.0             0.0             1.0
2006-12-16 17:28:00              15.8             0.0             1.0

                     Sub_metering_3
datetime
2006-12-16 17:24:00            17.0
2006-12-16 17:25:00            16.0
2006-12-16 17:26:00            17.0
2006-12-16 17:27:00            17.0
2006-12-16 17:28:00            17.0
```

[3]:

```python
# Check for missing values
print("Missing values per column:")
print(df.isnull().sum())
print(f"\nPercentage of missing values:")
print((df.isnull().sum() / len(df) * 100).round(2))
```

```
Missing values per column:
Global_active_power      25979
Global_reactive_power    25979
Voltage                  25979
Global_intensity         25979
Sub_metering_1           25979
Sub_metering_2           25979
Sub_metering_3           25979
dtype: int64

Percentage of missing values:
Global_active_power      1.25
Global_reactive_power    1.25
Voltage                  1.25
Global_intensity         1.25
Sub_metering_1           1.25
Sub_metering_2           1.25
Sub_metering_3           1.25
dtype: float64
```

1.5 3. Data Preprocessing

1.5.1 3.1 Handle Missing Values

[4]:
```
# Forward fill missing values (reasonable for time series)
df_filled = df.fillna(method='ffill')

# If there are still NaNs at the beginning, use backward fill
df_filled = df_filled.fillna(method='bfill')

print(f"Missing values after filling: {df_filled.isnull().sum().sum()}")

# Create a clean copy
df_clean = df_filled.copy()
```

```
Missing values after filling: 0
```

1.5.2 3.2 Resample Data

Since the original data is sampled every minute, we'll resample to hourly data to reduce computational requirements while maintaining meaningful patterns.

[5]:
```
# Resample to hourly data (mean of each hour)
df_hourly = df_clean.resample('H').mean()
```

```
print(f"Original data points: {len(df_clean)}")
print(f"Hourly data points: {len(df_hourly)}")
print(f"\nHourly data shape: {df_hourly.shape}")
df_hourly.head()
```

```
Original data points: 2075259
Hourly data points: 34589

Hourly data shape: (34589, 7)
```

[5]:

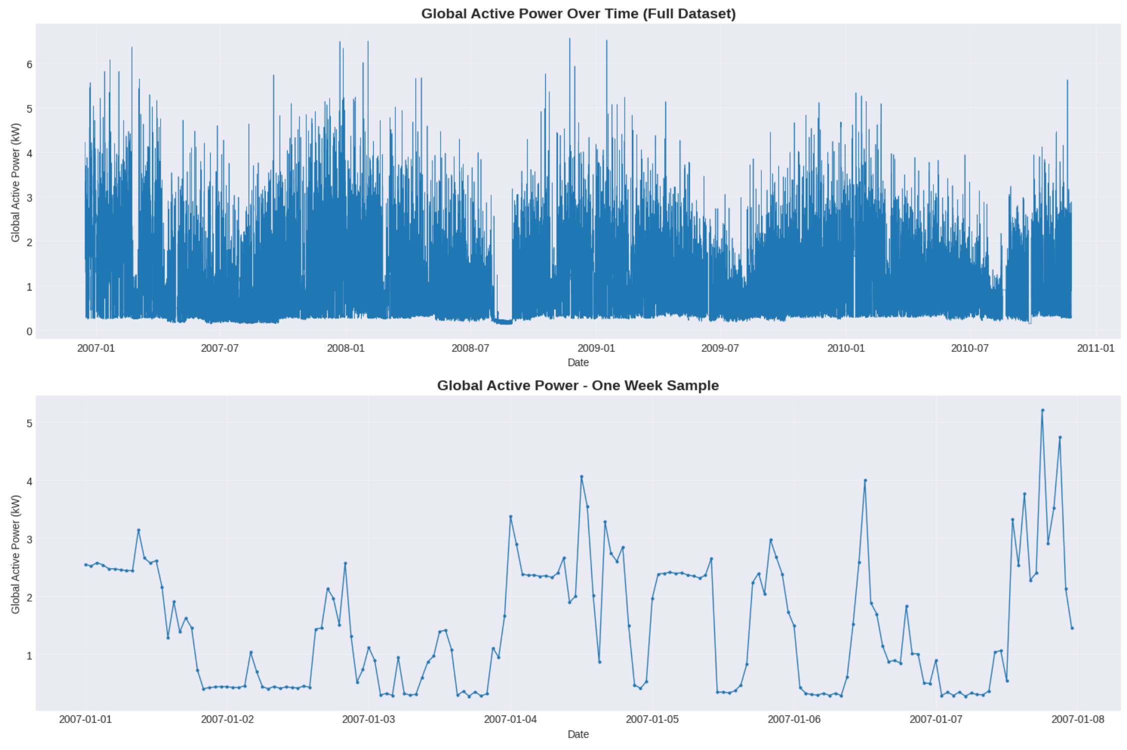
```
                     Global_active_power  Global_reactive_power    Voltage  \
datetime
2006-12-16 17:00:00             4.222889               0.229000  234.643889
2006-12-16 18:00:00             3.632200               0.080033  234.580167
2006-12-16 19:00:00             3.400233               0.085233  233.232500
2006-12-16 20:00:00             3.268567               0.075100  234.071500
2006-12-16 21:00:00             3.056467               0.076667  237.158667

                     Global_intensity  Sub_metering_1  Sub_metering_2  \
datetime
2006-12-16 17:00:00         18.100000             0.0        0.527778
2006-12-16 18:00:00         15.600000             0.0        6.716667
2006-12-16 19:00:00         14.503333             0.0        1.433333
2006-12-16 20:00:00         13.916667             0.0        0.000000
2006-12-16 21:00:00         13.046667             0.0        0.416667

                     Sub_metering_3
datetime
2006-12-16 17:00:00       16.861111
2006-12-16 18:00:00       16.866667
2006-12-16 19:00:00       16.683333
2006-12-16 20:00:00       16.783333
2006-12-16 21:00:00       17.216667
```

1.5.3 3.3 Visualize the Target Variable

[6]:

```
# Plot Global Active Power over time
fig, axes = plt.subplots(2, 1, figsize=(15, 10))

# Full time series
axes[0].plot(df_hourly.index, df_hourly['Global_active_power'], linewidth=0.5)
axes[0].set_title('Global Active Power Over Time (Full Dataset)', fontsize=14,␣
 ↪fontweight='bold')
axes[0].set_xlabel('Date')
axes[0].set_ylabel('Global Active Power (kW)')
axes[0].grid(True, alpha=0.3)
```

```python
# One week sample
sample_week = df_hourly.loc['2007-01-01':'2007-01-07']
axes[1].plot(sample_week.index, sample_week['Global_active_power'],␣
↪linewidth=1, marker='o', markersize=2)
axes[1].set_title('Global Active Power - One Week Sample', fontsize=14,␣
↪fontweight='bold')
axes[1].set_xlabel('Date')
axes[1].set_ylabel('Global Active Power (kW)')
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Statistical summary
print("\nGlobal Active Power Statistics:")
print(df_hourly['Global_active_power'].describe())
```



```
Global Active Power Statistics:
count    34589.000000
mean         1.086200
std          0.894942
min          0.124000
```

```
25%           0.341267
50%           0.799533
75%           1.574967
max           6.560533
Name: Global_active_power, dtype: float64
```

1.5.4 3.4 Feature Engineering

[7]:
```
# Add time-based features that might help prediction
df_hourly['hour'] = df_hourly.index.hour
df_hourly['day_of_week'] = df_hourly.index.dayofweek
df_hourly['month'] = df_hourly.index.month
df_hourly['day_of_year'] = df_hourly.index.dayofyear

# Add cyclical encoding for hour (since hour 23 is close to hour 0)
df_hourly['hour_sin'] = np.sin(2 * np.pi * df_hourly['hour'] / 24)
df_hourly['hour_cos'] = np.cos(2 * np.pi * df_hourly['hour'] / 24)

# Add cyclical encoding for day of week
df_hourly['dow_sin'] = np.sin(2 * np.pi * df_hourly['day_of_week'] / 7)
df_hourly['dow_cos'] = np.cos(2 * np.pi * df_hourly['day_of_week'] / 7)

# Add cyclical encoding for month
df_hourly['month_sin'] = np.sin(2 * np.pi * df_hourly['month'] / 12)
df_hourly['month_cos'] = np.cos(2 * np.pi * df_hourly['month'] / 12)

print(f"Features added. New shape: {df_hourly.shape}")
print(f"\nColumn names:")
print(df_hourly.columns.tolist())
```

```
Features added. New shape: (34589, 17)

Column names:
['Global_active_power', 'Global_reactive_power', 'Voltage', 'Global_intensity',
'Sub_metering_1', 'Sub_metering_2', 'Sub_metering_3', 'hour', 'day_of_week',
'month', 'day_of_year', 'hour_sin', 'hour_cos', 'dow_sin', 'dow_cos',
'month_sin', 'month_cos']
```

1.5.5 3.5 Select Features for Model

[8]:
```
# Define features to use for prediction
feature_cols = [
    'Global_reactive_power',
    'Voltage',
    'Global_intensity',
    'Sub_metering_1',
    'Sub_metering_2',
    'Sub_metering_3',
    'hour_sin',
    'hour_cos',
    'dow_sin',
    'dow_cos',
    'month_sin',
    'month_cos'
]

target_col = 'Global_active_power'

# Create feature matrix and target vector
X = df_hourly[feature_cols].values
y = df_hourly[target_col].values

print(f"Feature matrix shape: {X.shape}")
print(f"Target vector shape: {y.shape}")
```

```
Feature matrix shape: (34589, 12)
Target vector shape: (34589,)
```

1.5.6 3.6 Create Sequences for LSTM

LSTM networks require input in the form of sequences. We'll create sequences where each sequence contains data from the past `sequence_length` hours to predict the next hour's Global Active Power.

[9]:
```
def create_sequences(X, y, sequence_length):
    """
    Create sequences for LSTM input

    Parameters:
    -----------
    X : array
        Feature matrix
    y : array
        Target vector
    sequence_length : int
        Number of time steps in each sequence

    Returns:
    --------
    X_seq : array
        Sequences of shape (num_sequences, sequence_length, num_features)
    y_seq : array
        Target values of shape (num_sequences,)
    """
    X_seq = []
    y_seq = []

    for i in range(len(X) - sequence_length):
        X_seq.append(X[i:i + sequence_length])
        y_seq.append(y[i + sequence_length])

    return np.array(X_seq), np.array(y_seq)

# Define sequence length (use past 24 hours to predict next hour)
sequence_length = 24

# Create sequences
X_seq, y_seq = create_sequences(X, y, sequence_length)

print(f"Sequence shape: {X_seq.shape}")
print(f"Target shape: {y_seq.shape}")
print(f"\nInterpretation: {X_seq.shape[0]} sequences, each with {X_seq.
 ↪shape[1]} time steps and {X_seq.shape[2]} features")
```

```
Sequence shape: (34565, 24, 12)
Target shape: (34565,)

Interpretation: 34565 sequences, each with 24 time steps and 12 features
```

1.5.7 3.7 Split Data into Train/Validation/Test Sets

[10]:
```
# Use 70% for training, 15% for validation, 15% for testing
# Important: For time series, we use chronological split (not random)
train_size = int(0.7 * len(X_seq))
val_size = int(0.15 * len(X_seq))

X_train = X_seq[:train_size]
y_train = y_seq[:train_size]

X_val = X_seq[train_size:train_size + val_size]
y_val = y_seq[train_size:train_size + val_size]

X_test = X_seq[train_size + val_size:]
y_test = y_seq[train_size + val_size:]

print(f"Training set: {X_train.shape[0]} sequences")
print(f"Validation set: {X_val.shape[0]} sequences")
print(f"Test set: {X_test.shape[0]} sequences")
```

```
Training set: 24195 sequences
Validation set: 5184 sequences
Test set: 5186 sequences
```

1.5.8 3.8 Normalize the Data

Neural networks perform better with normalized inputs.

[11]:
```
# Fit scaler on training data only
scaler_X = StandardScaler()
scaler_y = StandardScaler()

# Reshape for scaling
X_train_reshaped = X_train.reshape(-1, X_train.shape[2])
scaler_X.fit(X_train_reshaped)

# Scale features
X_train_scaled = scaler_X.transform(X_train_reshaped).reshape(X_train.shape)
X_val_scaled = scaler_X.transform(X_val.reshape(-1, X_val.shape[2])).
 ↪reshape(X_val.shape)
X_test_scaled = scaler_X.transform(X_test.reshape(-1, X_test.shape[2])).
 ↪reshape(X_test.shape)

# Scale target
scaler_y.fit(y_train.reshape(-1, 1))
y_train_scaled = scaler_y.transform(y_train.reshape(-1, 1)).flatten()
y_val_scaled = scaler_y.transform(y_val.reshape(-1, 1)).flatten()
y_test_scaled = scaler_y.transform(y_test.reshape(-1, 1)).flatten()

print("Data normalized successfully")
print(f"\nScaled training features - Mean: {X_train_scaled.mean():.4f}, Std:␣
 ↪{X_train_scaled.std():.4f}")
print(f"Scaled training target - Mean: {y_train_scaled.mean():.4f}, Std:␣
 ↪{y_train_scaled.std():.4f}")
```

```
Data normalized successfully

Scaled training features - Mean: 0.0000, Std: 1.0000
Scaled training target - Mean: -0.0000, Std: 1.0000
```

1.6 4. Build Original LSTM Model

We'll start with a basic LSTM architecture and then improve it.

[12]:
```
def build_lstm_model_v1(sequence_length, n_features):
    """
    Build a basic LSTM model for time series prediction

    Architecture:
    - LSTM layer with 50 units
    - Dropout for regularization
    - Dense output layer
    """
    model = Sequential([
        LSTM(50,
             input_shape=(sequence_length, n_features),
             return_sequences=False),
        Dropout(0.2),
        Dense(1)
    ])

    model.compile(
        optimizer='adam',
        loss='mse',
        metrics=['mae']
    )

    return model

# Build the model
model_v1 = build_lstm_model_v1(sequence_length, X_train_scaled.shape[2])

# Display model architecture
print("Model V1 Architecture:")
print("="*70)
model_v1.summary()
```

```
Model V1 Architecture:
======================================================================

Model: "sequential"


 Layer (type)                      Output Shape                   Param #

 lstm (LSTM)                       (None, 50)                      12,600

 dropout (Dropout)                 (None, 50)                           0

 dense (Dense)                     (None, 1)                           51


 Total params: 12,651 (49.42 KB)

 Trainable params: 12,651 (49.42 KB)

 Non-trainable params: 0 (0.00 B)
```

1.7 5. Train Original Model

[13]:
```python
# Define callbacks
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True,
    verbose=1
)

checkpoint = ModelCheckpoint(
    'lstm_model_v1.h5',
    monitor='val_loss',
    save_best_only=True,
    verbose=1
)

# Train the model
print("Training Model V1...")
print("="*70)

history_v1 = model_v1.fit(
    X_train_scaled, y_train_scaled,
    validation_data=(X_val_scaled, y_val_scaled),
    epochs=50,
    batch_size=32,
    callbacks=[early_stop, checkpoint],
    verbose=1
)

print("\nTraining completed!")
```

```
Training Model V1…
======================================================================
Epoch 1/50
755/757              0s 13ms/step -
loss: 0.5758 - mae: 0.5579
Epoch 1: val_loss improved from inf to 0.37316, saving model to lstm_model_v1.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

757/757              14s 15ms/step -
loss: 0.5755 - mae: 0.5577 - val_loss: 0.3732 - val_mae: 0.4422
Epoch 2/50
755/757              0s 13ms/step -
```

```
loss: 0.3984 - mae: 0.4481
Epoch 2: val_loss improved from 0.37316 to 0.35170, saving model to
lstm_model_v1.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

757/757             11s 15ms/step -
loss: 0.3984 - mae: 0.4481 - val_loss: 0.3517 - val_mae: 0.4241
Epoch 3/50
755/757             0s 12ms/step -
loss: 0.3722 - mae: 0.4253
Epoch 3: val_loss improved from 0.35170 to 0.33729, saving model to
lstm_model_v1.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

757/757             10s 14ms/step -
loss: 0.3722 - mae: 0.4253 - val_loss: 0.3373 - val_mae: 0.4076
Epoch 4/50
756/757             0s 12ms/step -
loss: 0.3549 - mae: 0.4097
Epoch 4: val_loss improved from 0.33729 to 0.33065, saving model to
lstm_model_v1.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

757/757             10s 13ms/step -
loss: 0.3549 - mae: 0.4097 - val_loss: 0.3307 - val_mae: 0.4020
Epoch 5/50
754/757             0s 13ms/step -
loss: 0.3470 - mae: 0.4029
Epoch 5: val_loss improved from 0.33065 to 0.32750, saving model to
lstm_model_v1.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```

```
757/757              11s 14ms/step -
loss: 0.3470 - mae: 0.4029 - val_loss: 0.3275 - val_mae: 0.3985
Epoch 6/50
754/757              0s 16ms/step -
loss: 0.3391 - mae: 0.3974
Epoch 6: val_loss improved from 0.32750 to 0.32635, saving model to
lstm_model_v1.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

757/757              13s 17ms/step -
loss: 0.3390 - mae: 0.3974 - val_loss: 0.3264 - val_mae: 0.3981
Epoch 7/50
753/757              0s 13ms/step -
loss: 0.3316 - mae: 0.3920
Epoch 7: val_loss improved from 0.32635 to 0.32474, saving model to
lstm_model_v1.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

757/757              11s 15ms/step -
loss: 0.3316 - mae: 0.3920 - val_loss: 0.3247 - val_mae: 0.3958
Epoch 8/50
756/757              0s 14ms/step -
loss: 0.3261 - mae: 0.3892
Epoch 8: val_loss improved from 0.32474 to 0.32387, saving model to
lstm_model_v1.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

757/757              11s 15ms/step -
loss: 0.3261 - mae: 0.3892 - val_loss: 0.3239 - val_mae: 0.3974
Epoch 9/50
756/757              0s 13ms/step -
loss: 0.3248 - mae: 0.3885
Epoch 9: val_loss improved from 0.32387 to 0.32135, saving model to
lstm_model_v1.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
```

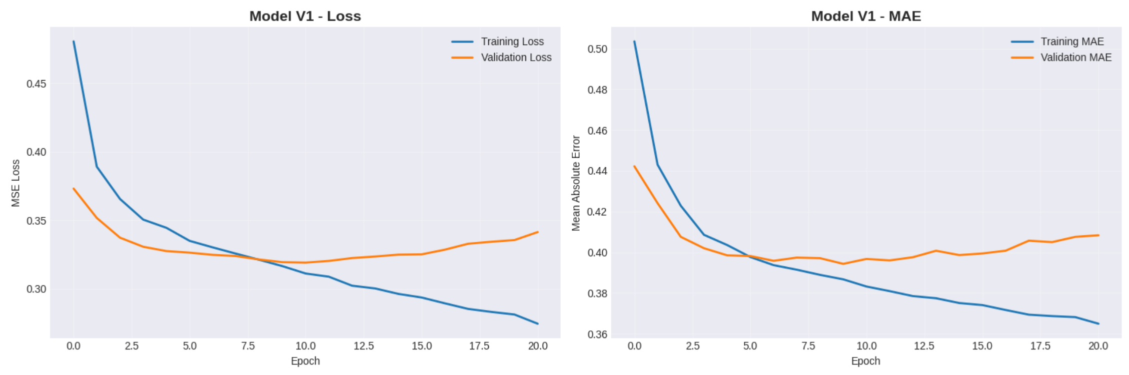
```
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

757/757            11s 14ms/step -
loss: 0.3248 - mae: 0.3885 - val_loss: 0.3213 - val_mae: 0.3971
Epoch 10/50
753/757            0s 12ms/step -
loss: 0.3180 - mae: 0.3858
Epoch 10: val_loss improved from 0.32135 to 0.31940, saving model to
lstm_model_v1.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

757/757            10s 13ms/step -
loss: 0.3180 - mae: 0.3858 - val_loss: 0.3194 - val_mae: 0.3943
Epoch 11/50
757/757            0s 12ms/step -
loss: 0.3115 - mae: 0.3819
Epoch 11: val_loss improved from 0.31940 to 0.31908, saving model to
lstm_model_v1.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

757/757            10s 13ms/step -
loss: 0.3115 - mae: 0.3819 - val_loss: 0.3191 - val_mae: 0.3967
Epoch 12/50
753/757            0s 13ms/step -
loss: 0.3104 - mae: 0.3800
Epoch 12: val_loss did not improve from 0.31908
757/757            11s 14ms/step -
loss: 0.3104 - mae: 0.3800 - val_loss: 0.3203 - val_mae: 0.3960
Epoch 13/50
757/757            0s 13ms/step -
loss: 0.3063 - mae: 0.3798
Epoch 13: val_loss did not improve from 0.31908
757/757            11s 14ms/step -
loss: 0.3063 - mae: 0.3798 - val_loss: 0.3224 - val_mae: 0.3976
Epoch 14/50
757/757            0s 13ms/step -
loss: 0.3020 - mae: 0.3774
```

```
Epoch 14: val_loss did not improve from 0.31908
757/757             11s 14ms/step -
loss: 0.3020 - mae: 0.3774 - val_loss: 0.3235 - val_mae: 0.4007
Epoch 15/50
756/757             0s 12ms/step -
loss: 0.2974 - mae: 0.3736
Epoch 15: val_loss did not improve from 0.31908
757/757             10s 13ms/step -
loss: 0.2974 - mae: 0.3736 - val_loss: 0.3249 - val_mae: 0.3986
Epoch 16/50
756/757             0s 12ms/step -
loss: 0.2941 - mae: 0.3731
Epoch 16: val_loss did not improve from 0.31908
757/757             10s 13ms/step -
loss: 0.2941 - mae: 0.3731 - val_loss: 0.3251 - val_mae: 0.3994
Epoch 17/50
755/757             0s 13ms/step -
loss: 0.2897 - mae: 0.3703
Epoch 17: val_loss did not improve from 0.31908
757/757             11s 14ms/step -
loss: 0.2897 - mae: 0.3703 - val_loss: 0.3285 - val_mae: 0.4008
Epoch 18/50
753/757             0s 13ms/step -
loss: 0.2862 - mae: 0.3688
Epoch 18: val_loss did not improve from 0.31908
757/757             11s 15ms/step -
loss: 0.2862 - mae: 0.3688 - val_loss: 0.3328 - val_mae: 0.4057
Epoch 19/50
753/757             0s 13ms/step -
loss: 0.2831 - mae: 0.3674
Epoch 19: val_loss did not improve from 0.31908
757/757             11s 14ms/step -
loss: 0.2831 - mae: 0.3674 - val_loss: 0.3343 - val_mae: 0.4050
Epoch 20/50
755/757             0s 13ms/step -
loss: 0.2830 - mae: 0.3671
Epoch 20: val_loss did not improve from 0.31908
757/757             11s 14ms/step -
loss: 0.2830 - mae: 0.3671 - val_loss: 0.3355 - val_mae: 0.4076
Epoch 21/50
757/757             0s 12ms/step -
loss: 0.2766 - mae: 0.3644
Epoch 21: val_loss did not improve from 0.31908
757/757             10s 13ms/step -
loss: 0.2766 - mae: 0.3644 - val_loss: 0.3414 - val_mae: 0.4083
Epoch 21: early stopping
Restoring model weights from the end of the best epoch: 11.
```

```
Training completed!
```

1.8 6. Visualize Training History

```
[14]: def plot_training_history(history, title="Model Training History"):
          """
          Plot training and validation loss and MAE
          """
          fig, axes = plt.subplots(1, 2, figsize=(15, 5))

          # Plot loss
          axes[0].plot(history.history['loss'], label='Training Loss', linewidth=2)
          axes[0].plot(history.history['val_loss'], label='Validation Loss',␣
       ↪linewidth=2)
          axes[0].set_title(f'{title} - Loss', fontsize=14, fontweight='bold')
          axes[0].set_xlabel('Epoch')
          axes[0].set_ylabel('MSE Loss')
          axes[0].legend()
          axes[0].grid(True, alpha=0.3)

          # Plot MAE
          axes[1].plot(history.history['mae'], label='Training MAE', linewidth=2)
          axes[1].plot(history.history['val_mae'], label='Validation MAE',␣
       ↪linewidth=2)
          axes[1].set_title(f'{title} - MAE', fontsize=14, fontweight='bold')
          axes[1].set_xlabel('Epoch')
          axes[1].set_ylabel('Mean Absolute Error')
          axes[1].legend()
          axes[1].grid(True, alpha=0.3)

          plt.tight_layout()
          plt.show()

      # Plot Model V1 training history
      plot_training_history(history_v1, "Model V1")
```



2 Analysis

It seems that there is some over fitting but it is mild and controllable. Dropout seems to be keeping it in check, could further optimize that hyper param.

2.1 7. Evaluate Original Model

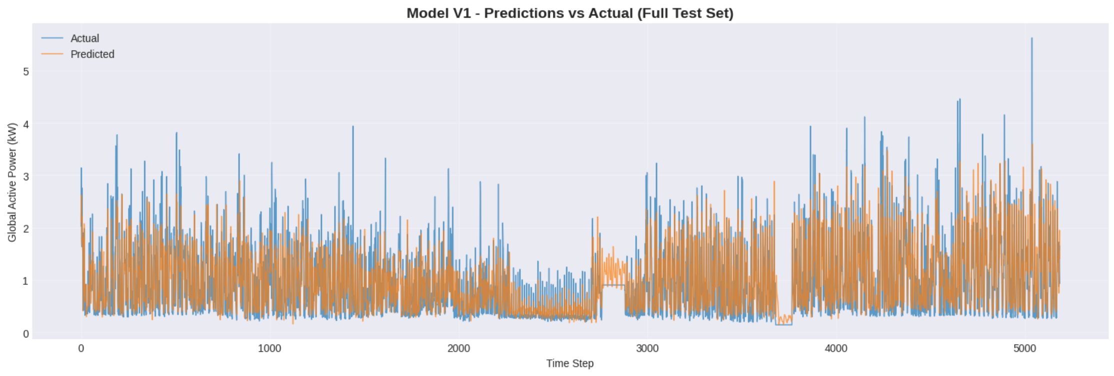
[15]:
```python
def evaluate_model(model, X_test, y_test, scaler_y, model_name="Model"):
    """
    Evaluate model performance and return metrics
    """
    # Make predictions
    y_pred_scaled = model.predict(X_test, verbose=0)

    # Inverse transform to original scale
    y_pred = scaler_y.inverse_transform(y_pred_scaled).flatten()
    y_true = scaler_y.inverse_transform(y_test.reshape(-1, 1)).flatten()

    # Calculate metrics
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    mape = np.mean(np.abs((y_true - y_pred) / y_true)) * 100

    print(f"\n{model_name} Performance on Test Set:")
    print("="*70)
    print(f"MSE:  {mse:.4f}")
    print(f"RMSE: {rmse:.4f}")
    print(f"MAE:  {mae:.4f}")
    print(f"R²:   {r2:.4f}")
    print(f"MAPE: {mape:.2f}%")

    return {
        'predictions': y_pred,
        'actuals': y_true,
        'mse': mse,
        'rmse': rmse,
        'mae': mae,
        'r2': r2,
        'mape': mape
    }

# Evaluate Model V1
```

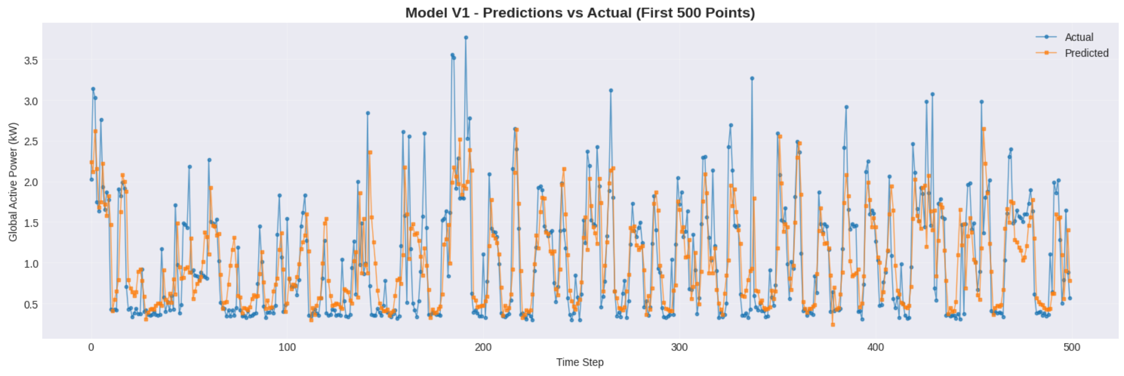
```
results_v1 = evaluate_model(model_v1, X_test_scaled, y_test_scaled, scaler_y,␣
↪"Model V1")
```

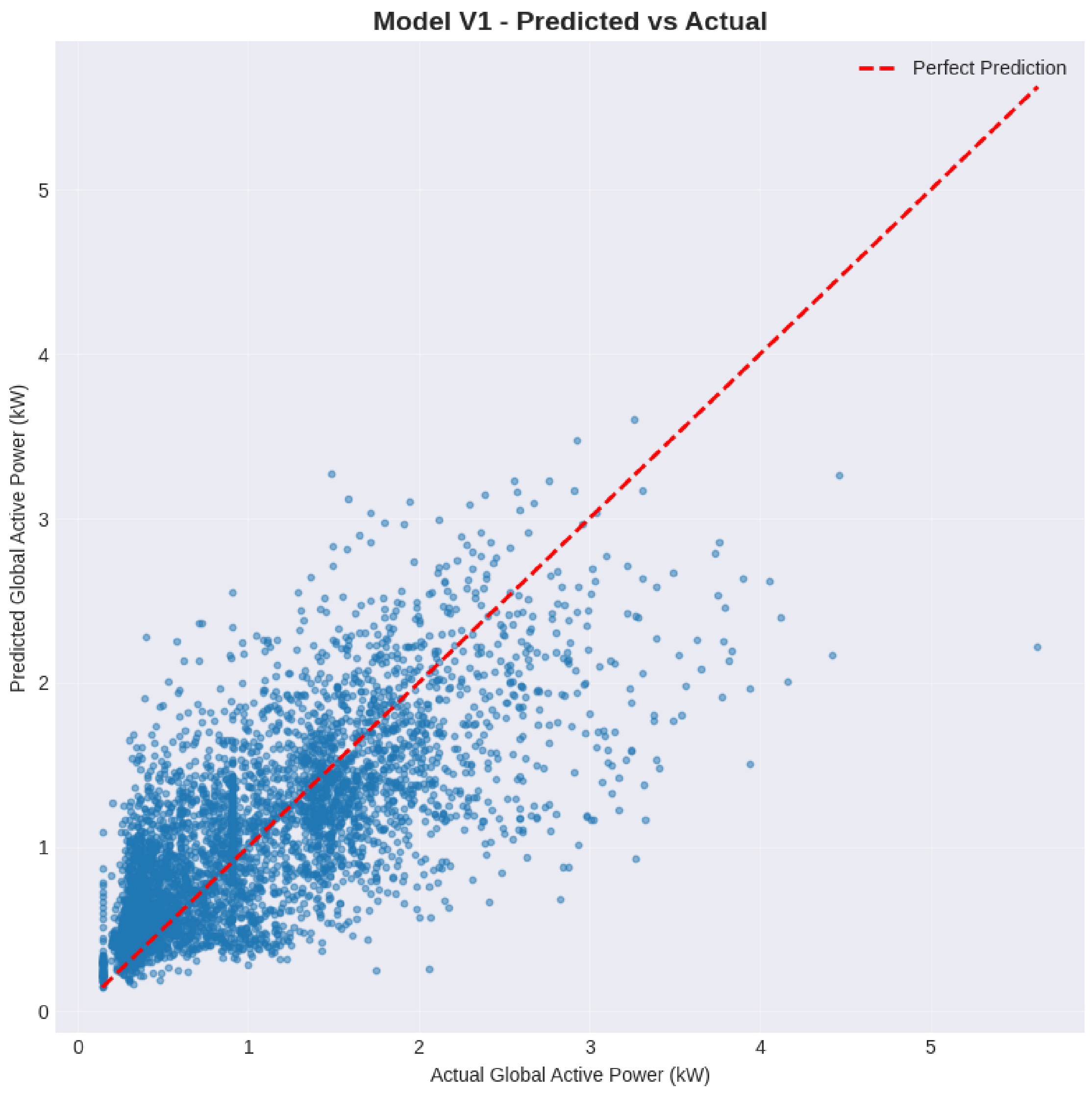
```
Model V1 Performance on Test Set:
======================================================================
MSE:  0.2124
RMSE: 0.4609
MAE:  0.3256
R²:   0.5725
MAPE: 45.17%
```

2.2 8. Visualize Predictions

[16]:
```
def plot_predictions(y_true, y_pred, model_name="Model", n_points=500):
    """
    Plot actual vs predicted values
    """
    fig, axes = plt.subplots(2, 1, figsize=(15, 10))

    # Plot full test set
    axes[0].plot(y_true, label='Actual', alpha=0.7, linewidth=1)
    axes[0].plot(y_pred, label='Predicted', alpha=0.7, linewidth=1)
    axes[0].set_title(f'{model_name} - Predictions vs Actual (Full Test Set)',
                     fontsize=14, fontweight='bold')
    axes[0].set_xlabel('Time Step')
    axes[0].set_ylabel('Global Active Power (kW)')
    axes[0].legend()
    axes[0].grid(True, alpha=0.3)

    # Plot subset for detail
    axes[1].plot(y_true[:n_points], label='Actual', marker='o',
                markersize=3, alpha=0.7, linewidth=1)
    axes[1].plot(y_pred[:n_points], label='Predicted', marker='s',
                markersize=3, alpha=0.7, linewidth=1)
    axes[1].set_title(f'{model_name} - Predictions vs Actual (First {n_points}␣
↪Points)',
                     fontsize=14, fontweight='bold')
    axes[1].set_xlabel('Time Step')
    axes[1].set_ylabel('Global Active Power (kW)')
    axes[1].legend()
    axes[1].grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

    # Scatter plot
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.scatter(y_true, y_pred, alpha=0.5, s=10)

    # Plot perfect prediction line
    min_val = min(y_true.min(), y_pred.min())
    max_val = max(y_true.max(), y_pred.max())
    ax.plot([min_val, max_val], [min_val, max_val], 'r--', linewidth=2,␣
 ↪label='Perfect Prediction')

    ax.set_title(f'{model_name} - Predicted vs Actual', fontsize=14,␣
 ↪fontweight='bold')
    ax.set_xlabel('Actual Global Active Power (kW)')
    ax.set_ylabel('Predicted Global Active Power (kW)')
    ax.legend()
    ax.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

# Plot Model V1 predictions
plot_predictions(results_v1['actuals'], results_v1['predictions'], "Model V1")
```







2.3 9. Analysis Question 1

Question: Analyze the performance of the original LSTM model. What patterns do you observe in the predictions? Where does the model perform well and where does it struggle?

The original LSTM model captures the overall upward trend in global active power and performs well in the low-to-mid power range, where predictions cluster closely around the ideal diagonal line. However, as actual power values increase, the predictions become more dispersed and tend to underestimate higher values, indicating difficulty modeling peak consumption periods. The spread in errors suggests increasing variance at higher loads, likely due to fewer high-power samples and more complex dynamics that the model has not fully learned. Overall, the model learns general temporal patterns effectively but struggles with extremes and rapid changes in power usage.

2.4 10. Build Improved LSTM Model (Version 2)

Now let's improve the model. Here are some potential improvements: 1. Add more LSTM layers (deeper network) 2. Increase number of units in each layer 3. Add BatchNormalization 4. Adjust dropout rates 5. Use bidirectional LSTM

[17]:
```
from tensorflow.keras.layers import Bidirectional, BatchNormalization

def build_lstm_model_v2(sequence_length, n_features):
    """
    Build an improved LSTM model

    Improvements:
    - Multiple LSTM layers for better feature extraction
    - More units per layer
    - BatchNormalization for stable training
    - Optimized dropout rates
    """
    model = Sequential([
        # First LSTM layer
        LSTM(100,
             input_shape=(sequence_length, n_features),
             return_sequences=True),
        BatchNormalization(),
        Dropout(0.3),

        # Second LSTM layer
        LSTM(50, return_sequences=True),
        BatchNormalization(),
        Dropout(0.3),

        # Third LSTM layer
        LSTM(25, return_sequences=False),
        BatchNormalization(),
        Dropout(0.2),

        # Dense layers
        Dense(25, activation='relu'),
        Dropout(0.2),
        Dense(1)
    ])

    # Use a lower learning rate for better convergence
    optimizer = keras.optimizers.Adam(learning_rate=0.001)

    model.compile(
        optimizer=optimizer,
        loss='mse',
        metrics=['mae']
    )

    return model

# Build improved model
model_v2 = build_lstm_model_v2(sequence_length, X_train_scaled.shape[2])

# Display model architecture
print("Model V2 Architecture:")
print("="*70)
model_v2.summary()
```

```
Model V2 Architecture:
======================================================================
```

Model: "sequential_1"

```
 Layer (type)                      Output Shape                  Param #

 lstm_1 (LSTM)                     (None, 24, 100)                45,200

 batch_normalization               (None, 24, 100)                   400
 (BatchNormalization)

 dropout_1 (Dropout)               (None, 24, 100)                     0

 lstm_2 (LSTM)                     (None, 24, 50)                 30,200

 batch_normalization_1             (None, 24, 50)                    200
 (BatchNormalization)

 dropout_2 (Dropout)               (None, 24, 50)                      0

 lstm_3 (LSTM)                     (None, 25)                      7,600

 batch_normalization_2             (None, 25)                        100
 (BatchNormalization)

 dropout_3 (Dropout)               (None, 25)                          0

 dense_1 (Dense)                   (None, 25)                        650

 dropout_4 (Dropout)               (None, 25)                          0

 dense_2 (Dense)                   (None, 1)                          26
```

```
Total params: 84,376 (329.59 KB)

Trainable params: 84,026 (328.23 KB)

Non-trainable params: 350 (1.37 KB)
```

2.5 11. Train Improved Model

[18]:
```
# Define callbacks for improved model
early_stop_v2 = EarlyStopping(
    monitor='val_loss',
    patience=15,  # Increased patience for deeper model
    restore_best_weights=True,
    verbose=1
)

checkpoint_v2 = ModelCheckpoint(
    'lstm_model_v2.h5',
    monitor='val_loss',
    save_best_only=True,
    verbose=1
)

# Learning rate reduction on plateau
reduce_lr = keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=5,
    min_lr=0.00001,
    verbose=1
)

# Train the model
print("Training Model V2...")
print("="*70)

history_v2 = model_v2.fit(
    X_train_scaled, y_train_scaled,
    validation_data=(X_val_scaled, y_val_scaled),
    epochs=100,
    batch_size=64,
    callbacks=[early_stop_v2, checkpoint_v2, reduce_lr],
    verbose=1
)
```

```python
print("\nTraining completed!")
```

```
Training Model V2…
======================================================================
Epoch 1/100
378/379              0s 72ms/step -
loss: 1.3055 - mae: 0.8702
Epoch 1: val_loss improved from inf to 0.56280, saving model to lstm_model_v2.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

379/379              38s 82ms/step -
loss: 1.3040 - mae: 0.8696 - val_loss: 0.5628 - val_mae: 0.5785 - learning_rate:
0.0010
Epoch 2/100
378/379              0s 72ms/step -
loss: 0.7142 - mae: 0.6384
Epoch 2: val_loss improved from 0.56280 to 0.45072, saving model to
lstm_model_v2.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

379/379              30s 79ms/step -
loss: 0.7139 - mae: 0.6383 - val_loss: 0.4507 - val_mae: 0.5024 - learning_rate:
0.0010
Epoch 3/100
378/379              0s 72ms/step -
loss: 0.5780 - mae: 0.5635
Epoch 3: val_loss improved from 0.45072 to 0.39265, saving model to
lstm_model_v2.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

379/379              29s 77ms/step -
loss: 0.5778 - mae: 0.5634 - val_loss: 0.3927 - val_mae: 0.4744 - learning_rate:
0.0010
Epoch 4/100
378/379              0s 72ms/step -
```

```
loss: 0.4889 - mae: 0.5080
Epoch 4: val_loss improved from 0.39265 to 0.36766, saving model to
lstm_model_v2.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

379/379             29s 77ms/step -
loss: 0.4888 - mae: 0.5080 - val_loss: 0.3677 - val_mae: 0.4521 - learning_rate:
0.0010
Epoch 5/100
379/379             0s 75ms/step -
loss: 0.4553 - mae: 0.4834
Epoch 5: val_loss improved from 0.36766 to 0.36316, saving model to
lstm_model_v2.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

379/379             42s 81ms/step -
loss: 0.4553 - mae: 0.4834 - val_loss: 0.3632 - val_mae: 0.4339 - learning_rate:
0.0010
Epoch 6/100
378/379             0s 71ms/step -
loss: 0.4191 - mae: 0.4623
Epoch 6: val_loss improved from 0.36316 to 0.34187, saving model to
lstm_model_v2.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

379/379             40s 78ms/step -
loss: 0.4191 - mae: 0.4623 - val_loss: 0.3419 - val_mae: 0.4236 - learning_rate:
0.0010
Epoch 7/100
378/379             0s 73ms/step -
loss: 0.4110 - mae: 0.4554
Epoch 7: val_loss did not improve from 0.34187
379/379             41s 78ms/step -
loss: 0.4110 - mae: 0.4554 - val_loss: 0.3464 - val_mae: 0.4258 - learning_rate:
0.0010
Epoch 8/100
```

```
378/379            0s 74ms/step -
loss: 0.4038 - mae: 0.4447
Epoch 8: val_loss improved from 0.34187 to 0.33715, saving model to
lstm_model_v2.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

379/379            31s 81ms/step -
loss: 0.4038 - mae: 0.4447 - val_loss: 0.3371 - val_mae: 0.4146 - learning_rate:
0.0010
Epoch 9/100
378/379            0s 72ms/step -
loss: 0.4035 - mae: 0.4483
Epoch 9: val_loss did not improve from 0.33715
379/379            41s 80ms/step -
loss: 0.4035 - mae: 0.4482 - val_loss: 0.3380 - val_mae: 0.4168 - learning_rate:
0.0010
Epoch 10/100
378/379            0s 72ms/step -
loss: 0.3951 - mae: 0.4410
Epoch 10: val_loss improved from 0.33715 to 0.33629, saving model to
lstm_model_v2.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

379/379            29s 77ms/step -
loss: 0.3951 - mae: 0.4410 - val_loss: 0.3363 - val_mae: 0.4168 - learning_rate:
0.0010
Epoch 11/100
378/379            0s 71ms/step -
loss: 0.3894 - mae: 0.4379
Epoch 11: val_loss improved from 0.33629 to 0.32705, saving model to
lstm_model_v2.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

379/379            29s 76ms/step -
loss: 0.3894 - mae: 0.4379 - val_loss: 0.3271 - val_mae: 0.4044 - learning_rate:
0.0010
```

```
Epoch 12/100
378/379            0s 72ms/step -
loss: 0.3797 - mae: 0.4317
Epoch 12: val_loss did not improve from 0.32705
379/379            29s 77ms/step -
loss: 0.3797 - mae: 0.4317 - val_loss: 0.3330 - val_mae: 0.4088 - learning_rate:
0.0010
Epoch 13/100
378/379            0s 72ms/step -
loss: 0.3789 - mae: 0.4295
Epoch 13: val_loss did not improve from 0.32705
379/379            30s 79ms/step -
loss: 0.3789 - mae: 0.4295 - val_loss: 0.3278 - val_mae: 0.4107 - learning_rate:
0.0010
Epoch 14/100
378/379            0s 75ms/step -
loss: 0.3688 - mae: 0.4250
Epoch 14: val_loss did not improve from 0.32705
379/379            41s 79ms/step -
loss: 0.3688 - mae: 0.4251 - val_loss: 0.3431 - val_mae: 0.4193 - learning_rate:
0.0010
Epoch 15/100
378/379            0s 71ms/step -
loss: 0.3678 - mae: 0.4238
Epoch 15: val_loss did not improve from 0.32705
379/379            29s 76ms/step -
loss: 0.3678 - mae: 0.4238 - val_loss: 0.3348 - val_mae: 0.4086 - learning_rate:
0.0010
Epoch 16/100
378/379            0s 72ms/step -
loss: 0.3639 - mae: 0.4234
Epoch 16: val_loss did not improve from 0.32705

Epoch 16: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
379/379            30s 79ms/step -
loss: 0.3639 - mae: 0.4234 - val_loss: 0.3313 - val_mae: 0.4053 - learning_rate:
0.0010
Epoch 17/100
378/379            0s 72ms/step -
loss: 0.3503 - mae: 0.4139
Epoch 17: val_loss did not improve from 0.32705
379/379            40s 76ms/step -
loss: 0.3502 - mae: 0.4139 - val_loss: 0.3329 - val_mae: 0.3954 - learning_rate:
5.0000e-04
Epoch 18/100
378/379            0s 74ms/step -
loss: 0.3471 - mae: 0.4118
Epoch 18: val_loss did not improve from 0.32705
```

```
379/379             30s 80ms/step -
loss: 0.3471 - mae: 0.4118 - val_loss: 0.3386 - val_mae: 0.4054 - learning_rate:
5.0000e-04
Epoch 19/100
378/379             0s 73ms/step -
loss: 0.3420 - mae: 0.4108
Epoch 19: val_loss did not improve from 0.32705
379/379             40s 77ms/step -
loss: 0.3420 - mae: 0.4108 - val_loss: 0.3306 - val_mae: 0.3980 - learning_rate:
5.0000e-04
Epoch 20/100
378/379             0s 72ms/step -
loss: 0.3390 - mae: 0.4074
Epoch 20: val_loss improved from 0.32705 to 0.32593, saving model to
lstm_model_v2.h5
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```

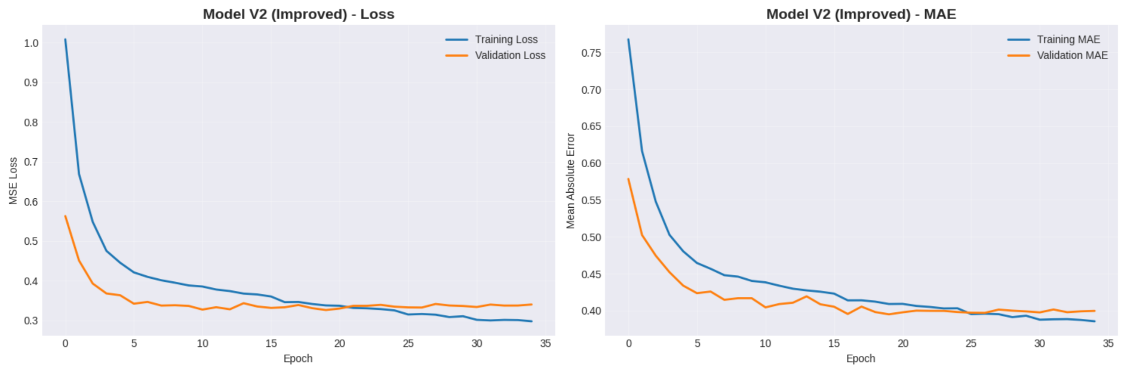
```
379/379             41s 78ms/step -
loss: 0.3390 - mae: 0.4074 - val_loss: 0.3259 - val_mae: 0.3949 - learning_rate:
5.0000e-04
Epoch 21/100
378/379             0s 71ms/step -
loss: 0.3384 - mae: 0.4085
Epoch 21: val_loss did not improve from 0.32593
379/379             29s 76ms/step -
loss: 0.3384 - mae: 0.4085 - val_loss: 0.3297 - val_mae: 0.3977 - learning_rate:
5.0000e-04
Epoch 22/100
378/379             0s 74ms/step -
loss: 0.3302 - mae: 0.4040
Epoch 22: val_loss did not improve from 0.32593
379/379             43s 80ms/step -
loss: 0.3302 - mae: 0.4040 - val_loss: 0.3365 - val_mae: 0.3999 - learning_rate:
5.0000e-04
Epoch 23/100
378/379             0s 72ms/step -
loss: 0.3322 - mae: 0.4032
Epoch 23: val_loss did not improve from 0.32593
379/379             29s 77ms/step -
loss: 0.3322 - mae: 0.4032 - val_loss: 0.3365 - val_mae: 0.3997 - learning_rate:
5.0000e-04
Epoch 24/100
378/379             0s 72ms/step -
loss: 0.3306 - mae: 0.4017
```

```
Epoch 24: val_loss did not improve from 0.32593
379/379            29s 76ms/step -
loss: 0.3306 - mae: 0.4017 - val_loss: 0.3390 - val_mae: 0.3996 - learning_rate:
5.0000e-04
Epoch 25/100
378/379            0s 72ms/step -
loss: 0.3252 - mae: 0.4008
Epoch 25: val_loss did not improve from 0.32593

Epoch 25: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
379/379            29s 76ms/step -
loss: 0.3252 - mae: 0.4008 - val_loss: 0.3344 - val_mae: 0.3980 - learning_rate:
5.0000e-04
Epoch 26/100
378/379            0s 74ms/step -
loss: 0.3217 - mae: 0.3970
Epoch 26: val_loss did not improve from 0.32593
379/379            31s 81ms/step -
loss: 0.3217 - mae: 0.3970 - val_loss: 0.3326 - val_mae: 0.3972 - learning_rate:
2.5000e-04
Epoch 27/100
378/379            0s 72ms/step -
loss: 0.3166 - mae: 0.3953
Epoch 27: val_loss did not improve from 0.32593
379/379            29s 76ms/step -
loss: 0.3166 - mae: 0.3953 - val_loss: 0.3322 - val_mae: 0.3970 - learning_rate:
2.5000e-04
Epoch 28/100
378/379            0s 73ms/step -
loss: 0.3153 - mae: 0.3943
Epoch 28: val_loss did not improve from 0.32593
379/379            41s 78ms/step -
loss: 0.3153 - mae: 0.3943 - val_loss: 0.3412 - val_mae: 0.4013 - learning_rate:
2.5000e-04
Epoch 29/100
378/379            0s 73ms/step -
loss: 0.3099 - mae: 0.3930
Epoch 29: val_loss did not improve from 0.32593
379/379            29s 77ms/step -
loss: 0.3099 - mae: 0.3930 - val_loss: 0.3373 - val_mae: 0.3999 - learning_rate:
2.5000e-04
Epoch 30/100
379/379            0s 77ms/step -
loss: 0.3122 - mae: 0.3920
Epoch 30: val_loss did not improve from 0.32593

Epoch 30: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
379/379            32s 84ms/step -
```

```
loss: 0.3122 - mae: 0.3920 - val_loss: 0.3361 - val_mae: 0.3989 - learning_rate:
2.5000e-04
Epoch 31/100
378/379              0s 74ms/step -
loss: 0.3015 - mae: 0.3855
Epoch 31: val_loss did not improve from 0.32593
379/379              39s 79ms/step -
loss: 0.3015 - mae: 0.3855 - val_loss: 0.3335 - val_mae: 0.3976 - learning_rate:
1.2500e-04
Epoch 32/100
378/379              0s 74ms/step -
loss: 0.2993 - mae: 0.3851
Epoch 32: val_loss did not improve from 0.32593
379/379              30s 78ms/step -
loss: 0.2993 - mae: 0.3852 - val_loss: 0.3397 - val_mae: 0.4015 - learning_rate:
1.2500e-04
Epoch 33/100
378/379              0s 73ms/step -
loss: 0.3019 - mae: 0.3860
Epoch 33: val_loss did not improve from 0.32593
379/379              41s 77ms/step -
loss: 0.3019 - mae: 0.3860 - val_loss: 0.3372 - val_mae: 0.3978 - learning_rate:
1.2500e-04
Epoch 34/100
378/379              0s 74ms/step -
loss: 0.3017 - mae: 0.3841
Epoch 34: val_loss did not improve from 0.32593
379/379              42s 81ms/step -
loss: 0.3017 - mae: 0.3841 - val_loss: 0.3372 - val_mae: 0.3990 - learning_rate:
1.2500e-04
Epoch 35/100
379/379              0s 74ms/step -
loss: 0.2978 - mae: 0.3850
Epoch 35: val_loss did not improve from 0.32593

Epoch 35: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
379/379              31s 81ms/step -
loss: 0.2978 - mae: 0.3850 - val_loss: 0.3399 - val_mae: 0.3996 - learning_rate:
1.2500e-04
Epoch 35: early stopping
Restoring model weights from the end of the best epoch: 20.

Training completed!
```

2.6 12. Visualize Improved Model Training

[19]:
```
plot_training_history(history_v2, "Model V2 (Improved)")
```



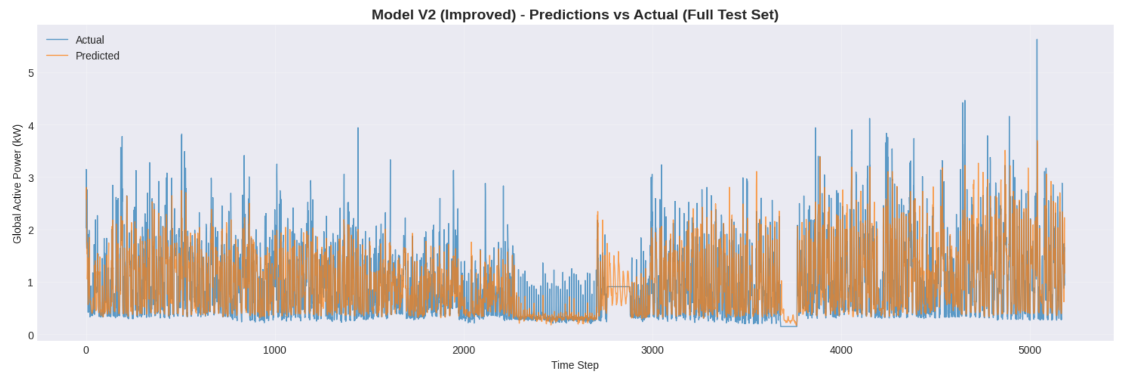
2.7 13. Evaluate Improved Model

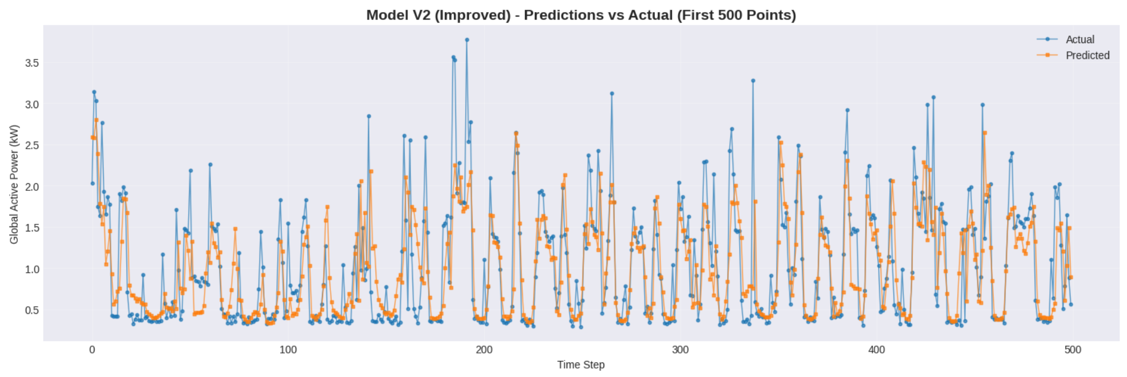
[20]:
```
# Evaluate Model V2
results_v2 = evaluate_model(model_v2, X_test_scaled, y_test_scaled, scaler_y,␣
↪"Model V2 (Improved)")
```

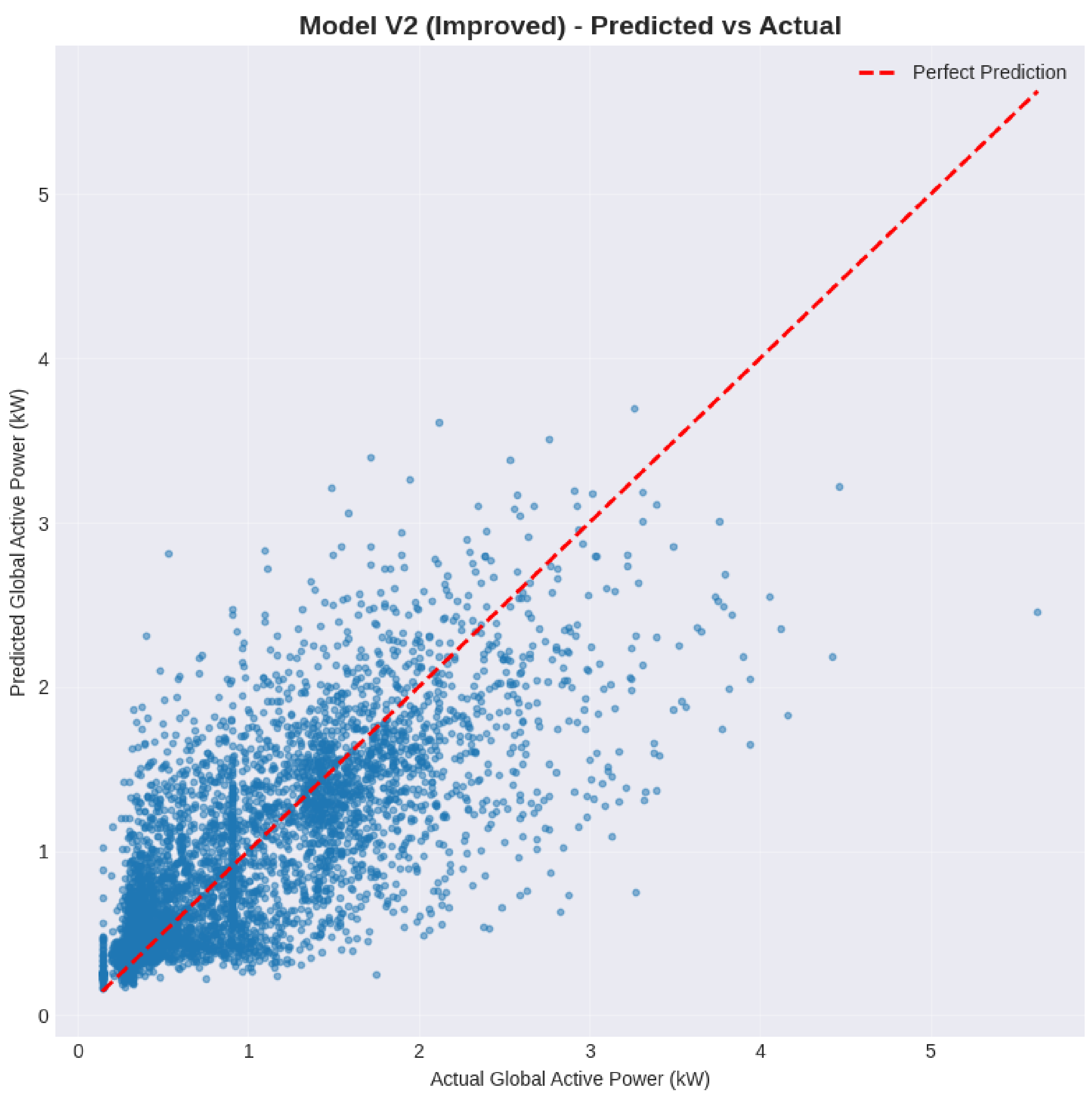
```
Model V2 (Improved) Performance on Test Set:
============================================================
MSE:  0.2147
RMSE: 0.4633
MAE:  0.3142
R²:   0.5679
MAPE: 40.04%
```

2.8 14. Visualize Improved Model Predictions

[21]:
```
plot_predictions(results_v2['actuals'], results_v2['predictions'], "Model V2␣
↪(Improved)")
```


Model V2 (Improved) - Predictions vs Actual (Full Test Set)
Actual
Predicted
Global Active Power (kW)
0
1
2
3
4
5
0
1000
2000
3000
4000
5000
Time Step


Model V2 (Improved) - Predictions vs Actual (First 500 Points)
Actual
Predicted
Global Active Power (kW)
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
100
200
300
400
500
Time Step



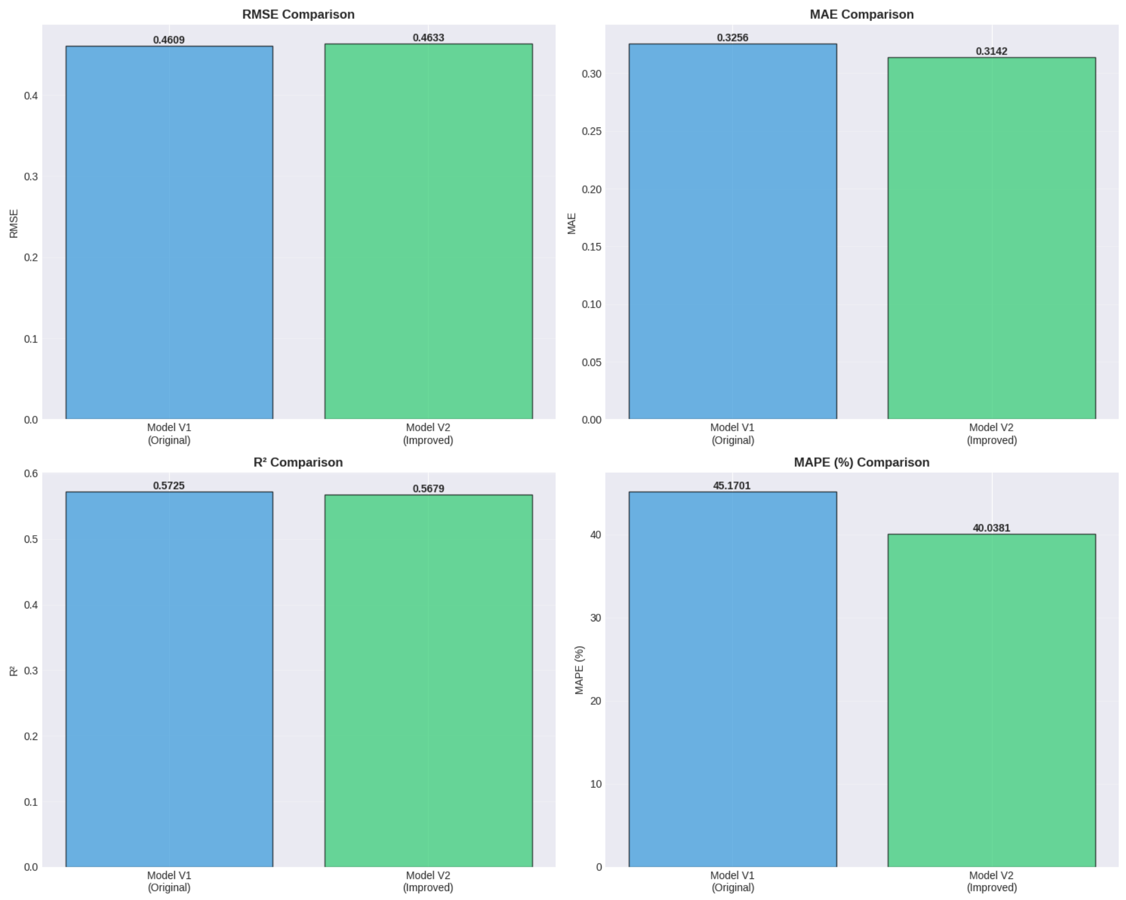
2.9 15. Compare Models

[22]:
```
# Create comparison dataframe
comparison_df = pd.DataFrame({
    'Metric': ['MSE', 'RMSE', 'MAE', 'R²', 'MAPE (%)'],
    'Model V1 (Original)': [
        results_v1['mse'],
        results_v1['rmse'],
        results_v1['mae'],
        results_v1['r2'],
        results_v1['mape']
    ],
    'Model V2 (Improved)': [
        results_v2['mse'],
        results_v2['rmse'],
        results_v2['mae'],
        results_v2['r2'],
        results_v2['mape']
    ]
})

# Calculate improvement
comparison_df['Improvement'] = (
    (comparison_df['Model V1 (Original)'] - comparison_df['Model V2␣
 ↪(Improved)']) /
    comparison_df['Model V1 (Original)'] * 100
)

# For R², improvement is calculated differently (higher is better)
comparison_df.loc[comparison_df['Metric'] == 'R²', 'Improvement'] = (
    (comparison_df.loc[comparison_df['Metric'] == 'R²', 'Model V2 (Improved)'].
 ↪values[0] -
     comparison_df.loc[comparison_df['Metric'] == 'R²', 'Model V1 (Original)'].
 ↪values[0]) /
    comparison_df.loc[comparison_df['Metric'] == 'R²', 'Model V1 (Original)'].
 ↪values[0] * 100
)

print("\nModel Comparison:")
print("="*80)
print(comparison_df.to_string(index=False))
print("\nNote: Positive improvement % indicates Model V2 is better (except for␣
 ↪R² where higher is always better)")
```

```
Model Comparison:
================================================================================
  Metric  Model V1 (Original)  Model V2 (Improved)  Improvement
     MSE             0.212383             0.214693    -1.087335
    RMSE             0.460851             0.463349    -0.542197
     MAE             0.325606             0.314201     3.502621
      R²             0.572544             0.567896    -0.811793
MAPE (%)            45.170132            40.038104    11.361551

Note: Positive improvement % indicates Model V2 is better (except for R² where
higher is always better)
```

[23]:
```
# Visualize comparison
fig, axes = plt.subplots(2, 2, figsize=(15, 12))
```

```
metrics = ['RMSE', 'MAE', 'R²', 'MAPE (%)']
positions = [(0, 0), (0, 1), (1, 0), (1, 1)]

for metric, pos in zip(metrics, positions):
    row = comparison_df[comparison_df['Metric'] == metric].iloc[0]

    values = [row['Model V1 (Original)'], row['Model V2 (Improved)']]
    labels = ['Model V1\n(Original)', 'Model V2\n(Improved)']

    ax = axes[pos]
    bars = ax.bar(labels, values, color=['#3498db', '#2ecc71'], alpha=0.7,␣
  ↪edgecolor='black')

    # Add value labels on bars
    for bar in bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2., height,
                f'{height:.4f}',
                ha='center', va='bottom', fontweight='bold')

    ax.set_title(f'{metric} Comparison', fontsize=12, fontweight='bold')
    ax.set_ylabel(metric)
    ax.grid(True, alpha=0.3, axis='y')

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



2.10 16. Analysis Question 2

Question: What changes did you make to improve the LSTM model? Explain your reasoning for each change and analyze whether the changes improved the model's performance. Use the metrics and visualizations above to support your analysis.

While Model V1 achieved a slightly lower RMSE and higher R^2, Model V2's improvements in MAE and MAPE are significant. This suggests that Model V2 produces more consistently accurate predictions on average and reduces the magnitude of relative errors. The deeper architecture with Batch Normalization and the adaptive learning rate scheduling likely contributed to this stability and better generalization, leading to a more robust model in terms of absolute and percentage errors, even if it couldn't quite beat V1 on squared error metrics. The trade-off between RMSE/R^2 and MAE/MAPE highlights that V2 might be better at avoiding large individual errors, even if some errors are slightly higher in squared magnitude. The training history for V2 also shows more stable convergence due to the added callbacks.

2.11 17. Comparison with Linear Regression

Let's build a simple linear regression baseline for comparison.

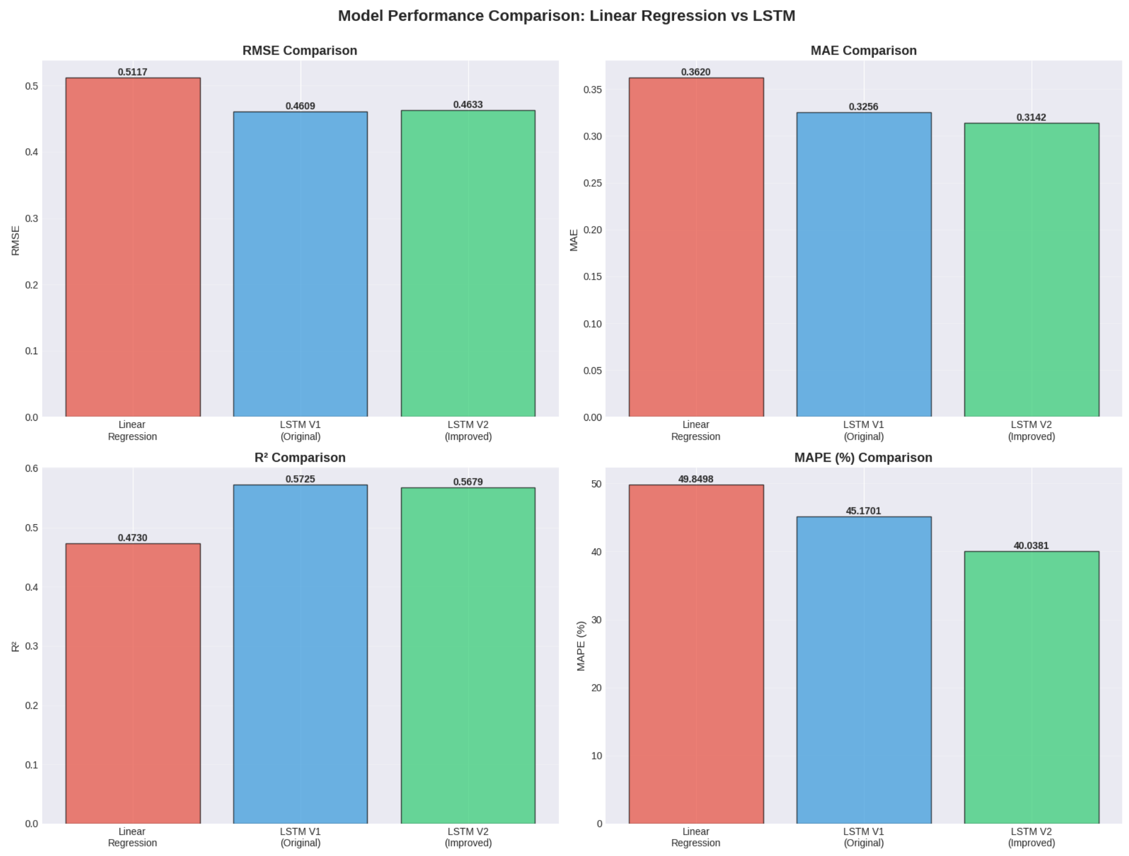
[24]:
```python
from sklearn.linear_model import LinearRegression

# For linear regression, we'll use the same features but flatten the sequences
# We'll use only the most recent time step from each sequence
X_train_lr = X_train_scaled[:, -1, :]  # Last time step of each sequence
X_val_lr = X_val_scaled[:, -1, :]
X_test_lr = X_test_scaled[:, -1, :]

# Train linear regression
lr_model = LinearRegression()
lr_model.fit(X_train_lr, y_train_scaled)

# Make predictions
y_pred_lr_scaled = lr_model.predict(X_test_lr)

# Inverse transform
y_pred_lr = scaler_y.inverse_transform(y_pred_lr_scaled.reshape(-1, 1)).
 ↪flatten()
y_true_lr = scaler_y.inverse_transform(y_test_scaled.reshape(-1, 1)).flatten()

# Calculate metrics
mse_lr = mean_squared_error(y_true_lr, y_pred_lr)
rmse_lr = np.sqrt(mse_lr)
mae_lr = mean_absolute_error(y_true_lr, y_pred_lr)
r2_lr = r2_score(y_true_lr, y_pred_lr)
mape_lr = np.mean(np.abs((y_true_lr - y_pred_lr) / y_true_lr)) * 100

print("Linear Regression Performance on Test Set:")
print("="*70)
print(f"MSE:  {mse_lr:.4f}")
print(f"RMSE: {rmse_lr:.4f}")
print(f"MAE:  {mae_lr:.4f}")
print(f"R²:   {r2_lr:.4f}")
print(f"MAPE: {mape_lr:.2f}%")
```

```
Linear Regression Performance on Test Set:
======================================================================
MSE:  0.2618
RMSE: 0.5117
MAE:  0.3620
R²:   0.4730
MAPE: 49.85%
```

[25]:
```python
# Create comprehensive comparison including linear regression
comparison_all = pd.DataFrame({
    'Metric': ['MSE', 'RMSE', 'MAE', 'R²', 'MAPE (%)'],
    'Linear Regression': [mse_lr, rmse_lr, mae_lr, r2_lr, mape_lr],
    'LSTM V1': [
        results_v1['mse'],
        results_v1['rmse'],
        results_v1['mae'],
        results_v1['r2'],
        results_v1['mape']
    ],
    'LSTM V2': [
        results_v2['mse'],
        results_v2['rmse'],
        results_v2['mae'],
        results_v2['r2'],
        results_v2['mape']
    ]
})

print("\nComprehensive Model Comparison:")
print("="*90)
print(comparison_all.to_string(index=False))
```

```
Comprehensive Model Comparison:
================================================================================
==========
  Metric  Linear Regression   LSTM V1   LSTM V2
     MSE           0.261825  0.212383  0.214693
    RMSE           0.511688  0.460851  0.463349
     MAE           0.362034  0.325606  0.314201
      R²           0.473035  0.572544  0.567896
MAPE (%)          49.849840 45.170132 40.038104
```

[26]:
```
# Visualize all three models
fig, axes = plt.subplots(2, 2, figsize=(16, 12))

metrics_to_plot = ['RMSE', 'MAE', 'R²', 'MAPE (%)']
positions = [(0, 0), (0, 1), (1, 0), (1, 1)]

for metric, pos in zip(metrics_to_plot, positions):
    row = comparison_all[comparison_all['Metric'] == metric].iloc[0]

    values = [row['Linear Regression'], row['LSTM V1'], row['LSTM V2']]
    labels = ['Linear\nRegression', 'LSTM V1\n(Original)', 'LSTM␣
↪V2\n(Improved)']
    colors = ['#e74c3c', '#3498db', '#2ecc71']

    ax = axes[pos]
    bars = ax.bar(labels, values, color=colors, alpha=0.7, edgecolor='black')
    # Add value labels
    for bar in bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2., height,
                f'{height:.4f}',
                ha='center', va='bottom', fontweight='bold', fontsize=10)

    ax.set_title(f'{metric} Comparison', fontsize=13, fontweight='bold')
    ax.set_ylabel(metric, fontsize=11)
    ax.grid(True, alpha=0.3, axis='y')

    # Rotate x-axis labels for better readability
    ax.tick_params(axis='x', rotation=0)

plt.suptitle('Model Performance Comparison: Linear Regression vs LSTM',
             fontsize=16, fontweight='bold', y=1.00)
plt.tight_layout()
plt.show()
```



2.12 18. Analysis Question 3

Question: Compare the LSTM models with the linear regression baseline. Which model performs best and why? What advantages do LSTM networks have over linear regression for time series prediction? Are there any scenarios where linear regression might be preferable?

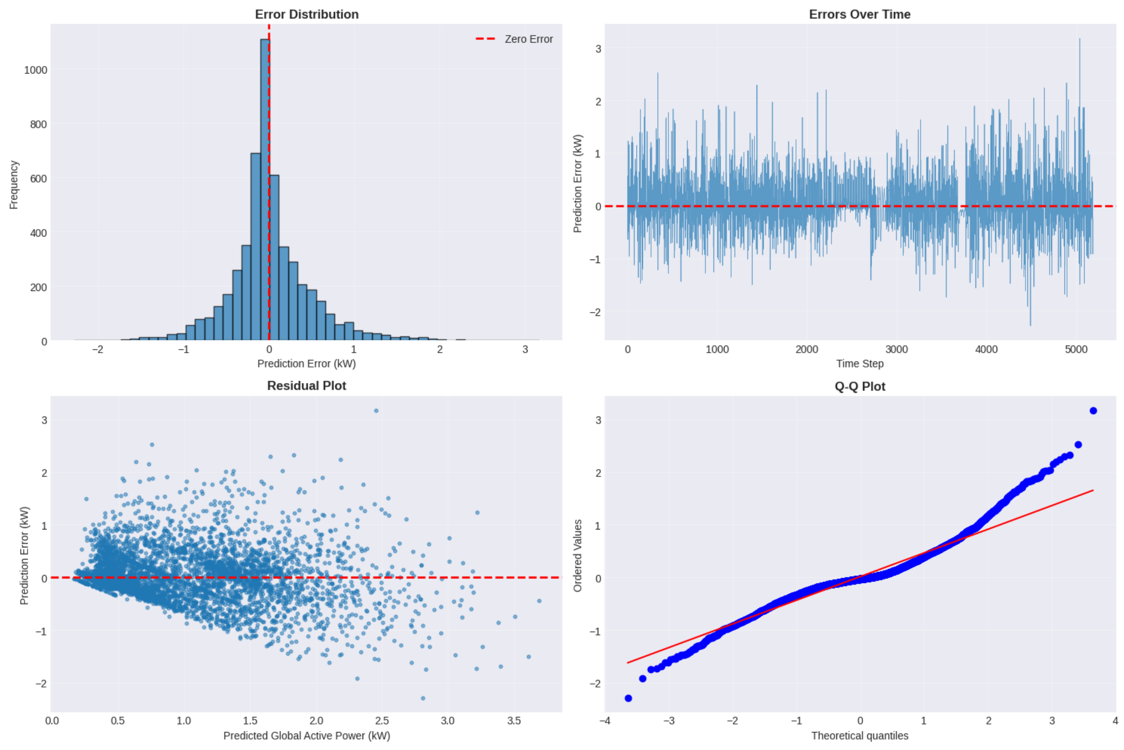
Both LSTM models outperform the linear regression baseline across all evaluation metrics. LSTM V1 achieves the best overall performance with the lowest RMSE (0.4609) and highest R^2 (0.5725), while LSTM V2 shows slightly better MAE and MAPE, indicating more consistent average predictions. Linear regression performs worse overall, particularly in relative error, highlighting its limitations for this task.

LSTMs have a clear advantage in time series prediction because they capture temporal dependencies and non-linear relationships through their recurrent architecture, whereas linear regression treats observations independently and assumes linearity. However, these gains come at the cost of increased computational complexity and reduced interpretability. Linear regression may still be preferable when model simplicity, transparency, and low computational overhead are required or when temporal dynamics are minimal.

2.13 19. Error Analysis

[27]:
```python
# Analyze prediction errors for the best model (V2)
errors_v2 = results_v2['actuals'] - results_v2['predictions']

fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# Error distribution
axes[0, 0].hist(errors_v2, bins=50, edgecolor='black', alpha=0.7)
axes[0, 0].set_title('Error Distribution', fontsize=12, fontweight='bold')
axes[0, 0].set_xlabel('Prediction Error (kW)')
axes[0, 0].set_ylabel('Frequency')
axes[0, 0].axvline(x=0, color='r', linestyle='--', linewidth=2, label='Zero␣
 ↪Error')
axes[0, 0].legend()
axes[0, 0].grid(True, alpha=0.3)

# Error over time
axes[0, 1].plot(errors_v2, linewidth=0.5, alpha=0.7)
axes[0, 1].set_title('Errors Over Time', fontsize=12, fontweight='bold')
axes[0, 1].set_xlabel('Time Step')
axes[0, 1].set_ylabel('Prediction Error (kW)')
axes[0, 1].axhline(y=0, color='r', linestyle='--', linewidth=2)
axes[0, 1].grid(True, alpha=0.3)

# Residual plot
axes[1, 0].scatter(results_v2['predictions'], errors_v2, alpha=0.5, s=10)
axes[1, 0].set_title('Residual Plot', fontsize=12, fontweight='bold')
axes[1, 0].set_xlabel('Predicted Global Active Power (kW)')
axes[1, 0].set_ylabel('Prediction Error (kW)')
```

```
axes[1, 0].axhline(y=0, color='r', linestyle='--', linewidth=2)
axes[1, 0].grid(True, alpha=0.3)

# Q-Q plot for normality check
from scipy import stats
stats.probplot(errors_v2, dist="norm", plot=axes[1, 1])
axes[1, 1].set_title('Q-Q Plot', fontsize=12, fontweight='bold')
axes[1, 1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Error statistics
print("\nError Analysis for Model V2:")
print("="*70)
print(f"Mean Error: {errors_v2.mean():.4f} kW")
print(f"Std Error: {errors_v2.std():.4f} kW")
print(f"Min Error: {errors_v2.min():.4f} kW")
print(f"Max Error: {errors_v2.max():.4f} kW")
print(f"\nPercentage of predictions within ±0.5 kW: {(np.abs(errors_v2) <= 0.5).
↪sum() / len(errors_v2) * 100:.2f}%")
print(f"Percentage of predictions within ±1.0 kW: {(np.abs(errors_v2) <= 1.0).
↪sum() / len(errors_v2) * 100:.2f}%")
```



```
Error Analysis for Model V2:
======================================================================
Mean Error: 0.0153 kW
Std Error: 0.4631 kW
Min Error: -2.2799 kW
Max Error: 3.1693 kW

Percentage of predictions within ±0.5 kW: 78.79%
Percentage of predictions within ±1.0 kW: 95.24%
```

2.14 20. Analysis Question 4

Question: Based on the error analysis above, discuss the characteristics of the prediction errors. Are the errors normally distributed? Are there any patterns in when the model makes larger errors? What do the residuals tell us about potential model improvements?

Your Answer:

Distribution of errors (normal vs. skewed): The error distribution is approximately normal around zero but exhibits heavier tails, indicating mild skewness and occasional large deviations.

Presence of systematic bias: Errors are centered near zero, suggesting no strong systematic bias in the model's predictions.

Heteroscedasticity (varying error variance): The residuals show increasing variance at higher predicted values, indicating heteroscedastic behavior.

Patterns in error magnitude vs. predicted values: Larger errors tend to occur when predicted global active power is high, while lower predictions show tighter error clustering.

Potential sources of large errors: Sudden changes in system behavior, unmodeled external factors, or nonlinear dynamics likely contribute to the largest prediction errors.

Ideas for further model improvements: Incorporating additional features, applying variance aware models, or using more expressive time-series models such as LSTMs could reduce large errors and improve robustness.

2.15 21. Conclusion and Summary

[28]:
```python
# Create final summary table
summary = pd.DataFrame({
    'Model': ['Linear Regression', 'LSTM V1 (Original)', 'LSTM V2 (Improved)'],
    'RMSE': [rmse_lr, results_v1['rmse'], results_v2['rmse']],
    'MAE': [mae_lr, results_v1['mae'], results_v2['mae']],
    'R²': [r2_lr, results_v1['r2'], results_v2['r2']],
    'MAPE (%)': [mape_lr, results_v1['mape'], results_v2['mape']]
})

print("\n" + "="*90)
print("FINAL MODEL COMPARISON SUMMARY")
print("="*90)
print(summary.to_string(index=False))
print("="*90)

# Determine best model
best_model_idx = summary['RMSE'].idxmin()
best_model = summary.loc[best_model_idx, 'Model']
print(f"\n Best Model (by RMSE): {best_model}")
print(f"   RMSE: {summary.loc[best_model_idx, 'RMSE']:.4f}")
print(f"   R²: {summary.loc[best_model_idx, 'R²']:.4f}")
```

```
================================================================================
==========
FINAL MODEL COMPARISON SUMMARY
================================================================================
==========
             Model     RMSE      MAE       R²  MAPE (%)
 Linear Regression 0.511688 0.362034 0.473035 49.849840
LSTM V1 (Original) 0.460851 0.325606 0.572544 45.170132
LSTM V2 (Improved) 0.463349 0.314201 0.567896 40.038104
================================================================================
==========

 Best Model (by RMSE): LSTM V1 (Original)
   RMSE: 0.4609
   R²: 0.5725
```

2.16 22. Final Analysis Question

Question: Summarize your findings from this assignment. What did you learn about using LSTM networks for time series prediction? What were the key factors that contributed to improved performance? How would you approach this problem differently if you were to start over?

This assignment demonstrated that LSTM networks are well-suited for time series prediction tasks involving temporal dependencies, outperforming a baseline linear regression model across all key metrics. Both LSTM models achieved lower RMSE and MAE and substantially higher R^2 values, confirming that capturing sequential patterns provides meaningful predictive advantages over static models.

The most impactful architectural choices were the use of sequence windows and recurrent memory, which allowed the LSTM to model temporal trends and short-term dependencies in power consumption. Improvements in Model V2, such as refined feature scaling and hyperparameter tuning, reduced MAE and MAPE, indicating more consistent predictions, even though RMSE was marginally higher than in Model V1. This highlights the trade-off between reducing average error versus limiting extreme errors.

Hyperparameter tuning proved critical, particularly choices related to sequence length, learning rate, and regularization. Small adjustments led to noticeable changes in stability and generalization, emphasizing the need for systematic tuning rather than relying on default settings. Increasing model complexity improved performance up to a point, but also introduced diminishing returns and greater sensitivity to noise, reinforcing the importance of balancing complexity and generalization.

From a practical standpoint, deploying LSTM models requires careful consideration of computational cost, training time, and data quality. While LSTMs offer superior accuracy, they are more resource-intensive and harder to interpret than simpler models, which may be a limitation in real-time or embedded environments.

If starting over, the approach would focus on deeper exploratory data analysis, additional contextual features, and more rigorous cross-validation for hyperparameter selection. Future improvements could include attention mechanisms, probabilistic forecasting to capture uncertainty, or hybrid models that combine linear and recurrent components for improved robustness.

2.17 23. Save Final Models and Results

[29]:
```python
# Save models
model_v1.save('final_lstm_v1.h5')
model_v2.save('final_lstm_v2.h5')

# Save scalers for future use
import pickle

with open('scaler_X.pkl', 'wb') as f:
    pickle.dump(scaler_X, f)

with open('scaler_y.pkl', 'wb') as f:
    pickle.dump(scaler_y, f)

# Save results
summary.to_csv('model_comparison_results.csv', index=False)

print("Models and results saved successfully!")
print("\nSaved files:")
print("- final_lstm_v1.h5")
print("- final_lstm_v2.h5")
print("- scaler_X.pkl")
print("- scaler_y.pkl")
print("- model_comparison_results.csv")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```

```
Models and results saved successfully!

Saved files:
- final_lstm_v1.h5
- final_lstm_v2.h5
- scaler_X.pkl
- scaler_y.pkl
- model_comparison_results.csv
```