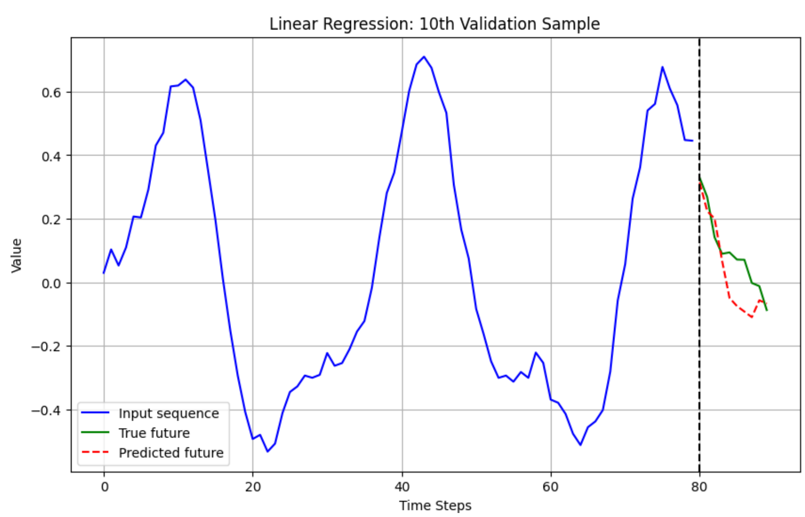
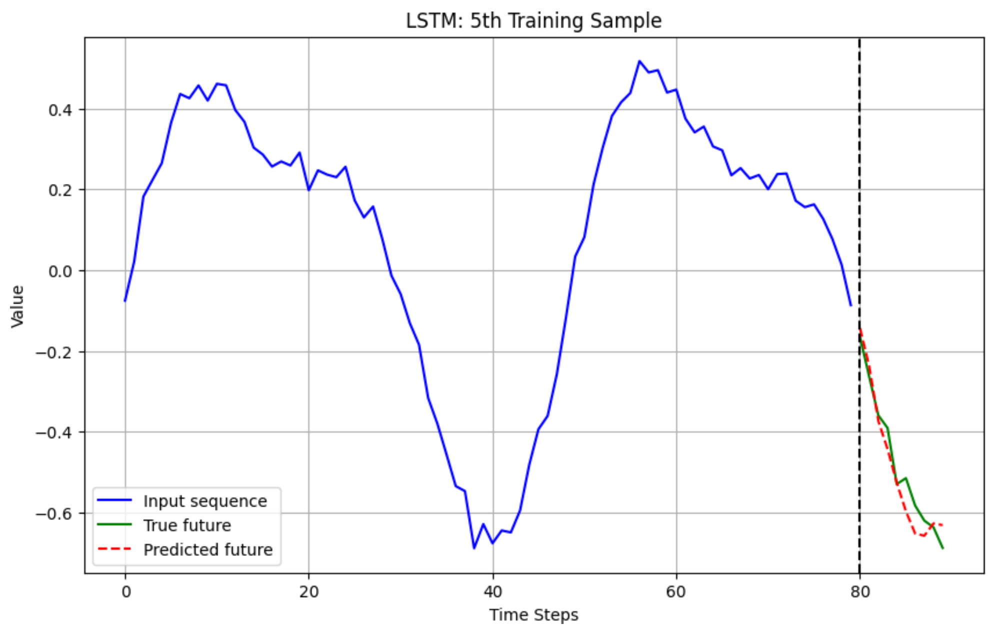
Classwork 07

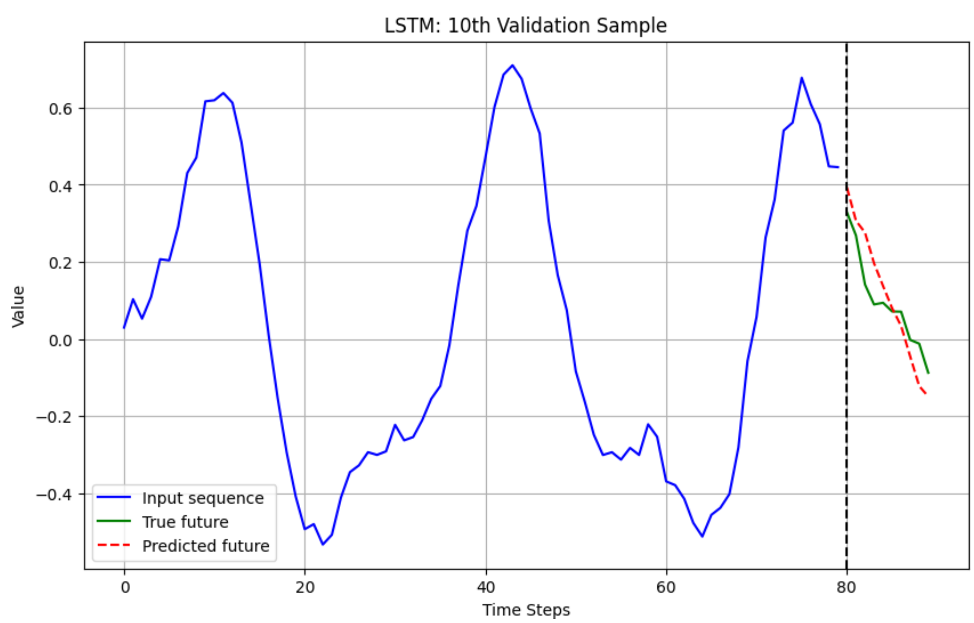
1. Generate 4000 time series with 80 time steps. The 80% of the generated data is used for training and the rest is used as validation data.

(a) Build a linear regression model to predict 10 values time steps ahead. Plot the waveform and its prediction in the 5^{th} generated time series in train set and the 10^{th} generated time series in validation set.

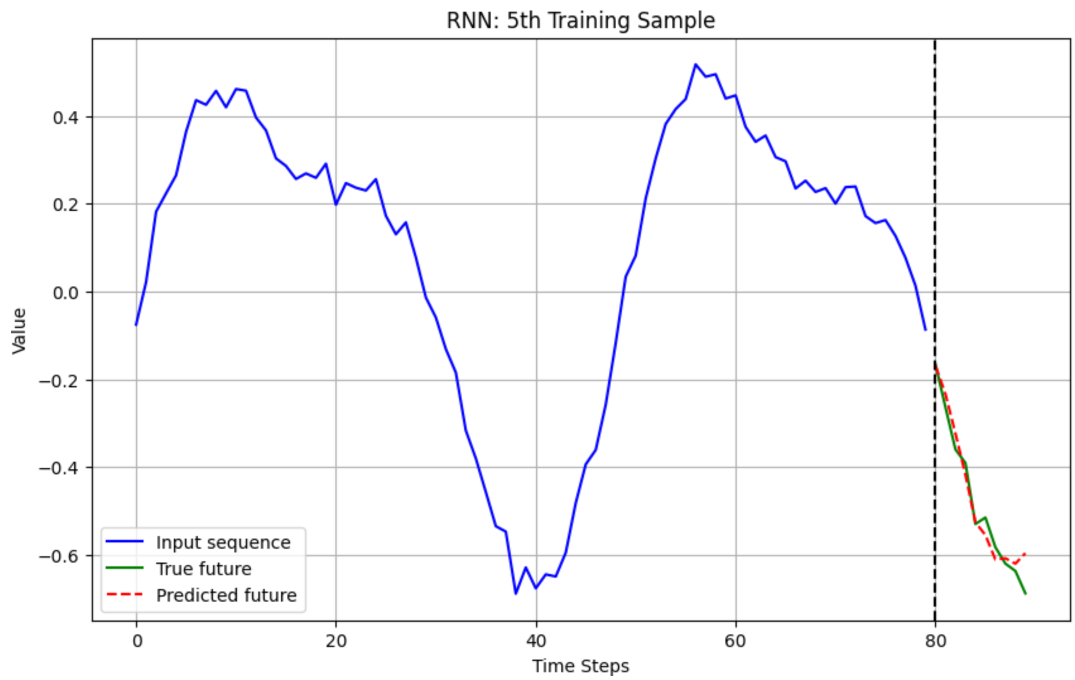


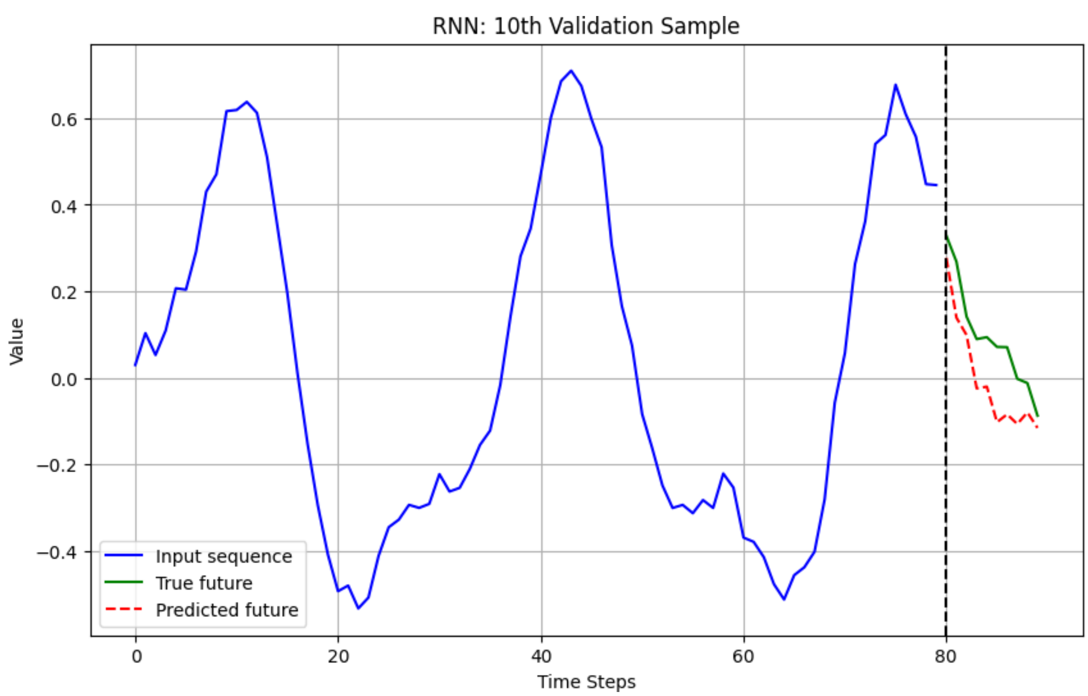
(b) Build a LSTM model to predict 10 values time steps ahead. The LSTM contains two layers with 10 and 10 units, respectively. Plot the waveform and its prediction in the 5th generated time series in train set and the 10th generated time series in validation set.





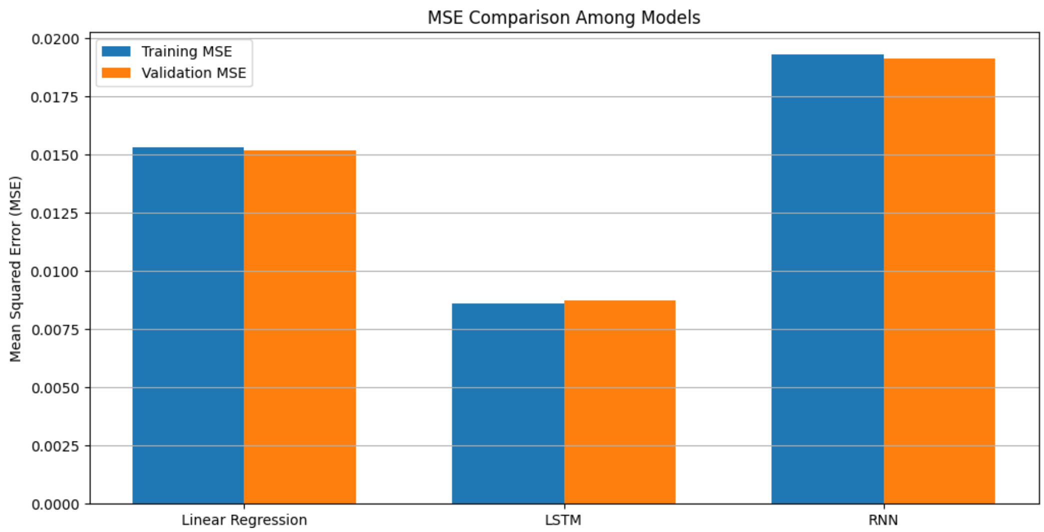
(c) Build a RNN model to predict 10 values time steps ahead. The RNN contains two layers with 10 and 10 units, respectively. Plot the waveform and its prediction in the 5th generated time series in train set and the 10th generated time series in validation set.

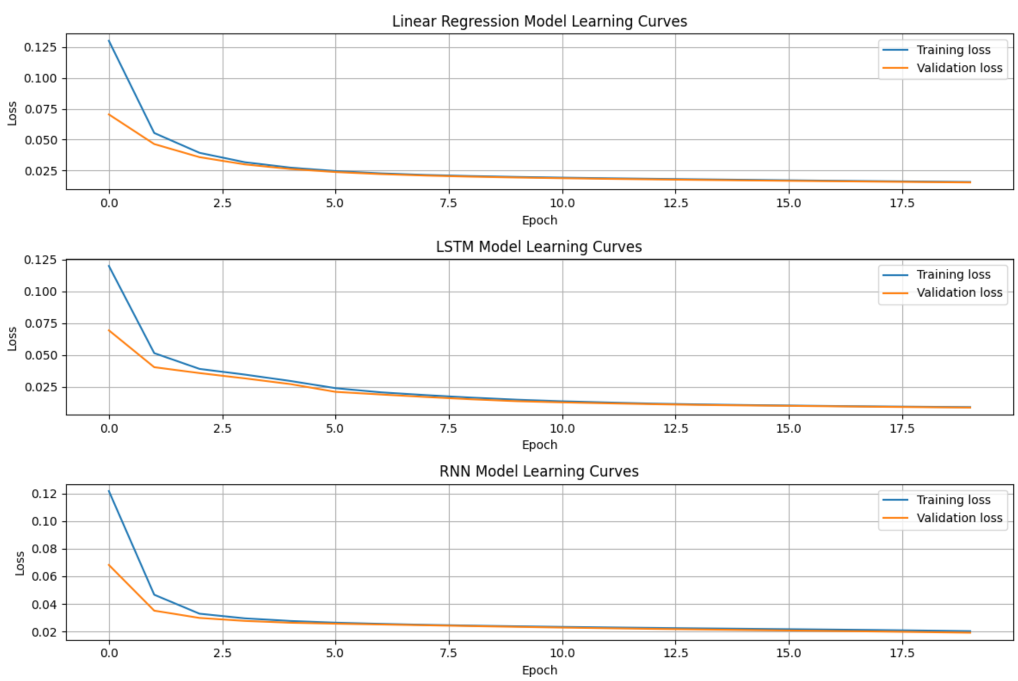




(d) Compare the generated waveform and MSE among the aforementioned models.

Model	**Linear Regression**	**LSTM Model**	**RNN Model**
Training MSE	**0.015324**	**0.008623**	**0.01G286**
Validation MSE	**0.015172**	**0.008735**	**0.01G135**





It is evident that LSTM has the least MSE.

Waveform Analysis

The generated time series consist of two sine waves with different frequencies plus noise. This creates a pattern that:

- Has clear periodic components
- Contains some complexity due to the combination of waves
- Includes randomness from the noise component

For such data:

- Linear regression can capture basic relationships but struggles with the non-linear periodic patterns
- LSTM can model the temporal dependencies and periodic components well
- RNN falls somewhere in between, handling some periodicity but potentially struggling with longer dependencies

The optimal model depends on the complexity of the underlying patterns in the time series. For simple patterns, even the linear model might perform adequately, while more complex patterns would benefit from the LSTM's sophisticated architecture.

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras

# Make sure TensorFlow doesn't use previously cached state
tf.keras.backend.clear_session()

# Generate 4000 time series with 80 time steps each
def generate_time_series(batch_size, n_steps):
    """Generate synthetic time series data."""
    np.random.seed(42)  # For reproducibility
    freq1, freq2, offsets1, offsets2 = np.random.rand(4, batch_size,
1)
    time = np.linspace(0, 1, n_steps)
    series = 0.5 * np.sin((time - offsets1) * (freq1 * 10 + 10))  #
wave 1
    series += 0.2 * np.sin((time - offsets2) * (freq2 * 20 + 20))  #
wave 2
    series += 0.1 * (np.random.rand(batch_size, n_steps) - 0.5)  #
noise
    return series[..., np.newaxis].astype(np.float32)

# Generate 4000 time series with 80+10 time steps
n_steps = 80
n_future_steps = 10
series = generate_time_series(4000, n_steps + n_future_steps)

# Split into training (80%) and validation (20%)
X_train = series[:3200, :n_steps]
y_train = series[:3200, n_steps:, 0]
X_valid = series[3200:, :n_steps]
y_valid = series[3200:, n_steps:, 0]

print(f"X_train shape: {X_train.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"X_valid shape: {X_valid.shape}")
print(f"y_valid shape: {y_valid.shape}")

# (a) Build and train a linear regression model
print("\n(a) Building and training linear regression model...")
def create_and_train_linear_model():
    tf.keras.backend.clear_session()  # Clear session to avoid
variable conflicts

    model = keras.Sequential([
        keras.layers.Flatten(input_shape=[n_steps, 1]),
        keras.layers.Dense(n_future_steps)
    ])
```

```
    model.compile(loss="mse", optimizer="adam")

    history = model.fit(
        X_train, y_train,
        epochs=20,
        validation_data=(X_valid, y_valid),
        verbose=1
    )

    return model, history

linear_model, linear_history = create_and_train_linear_model()

# (b) Build and train an LSTM model
print("\n(b) Building and training LSTM model...")
def create_and_train_lstm_model():
    tf.keras.backend.clear_session()  # Clear session to avoid
variable conflicts

    model = keras.Sequential([
        keras.layers.LSTM(10, return_sequences=True,
input_shape=[n_steps, 1]),
        keras.layers.LSTM(10),
        keras.layers.Dense(n_future_steps)
    ])

    model.compile(loss="mse", optimizer="adam")

    history = model.fit(
        X_train, y_train,
        epochs=20,
        validation_data=(X_valid, y_valid),
        verbose=1
    )

    return model, history

lstm_model, lstm_history = create_and_train_lstm_model()

# (c) Build and train an RNN model
print("\n(c) Building and training RNN model...")
def create_and_train_rnn_model():
    tf.keras.backend.clear_session()  # Clear session to avoid
variable conflicts

    model = keras.Sequential([
        keras.layers.SimpleRNN(10, return_sequences=True,
input_shape=[n_steps, 1]),
        keras.layers.SimpleRNN(10),
        keras.layers.Dense(n_future_steps)
```

```
    ])

    model.compile(loss="mse", optimizer="adam")

    history = model.fit(
        X_train, y_train,
        epochs=20,
        validation_data=(X_valid, y_valid),
        verbose=1
    )

    return model, history

rnn_model, rnn_history = create_and_train_rnn_model()

# Helper function to plot time series and predictions
def plot_series(x, y_true, y_pred, title, filename=None):
    plt.figure(figsize=(10, 6))
    plt.plot(range(len(x)), x, 'b-', label='Input sequence')
    plt.plot(range(len(x), len(x) + len(y_true)), y_true, 'g-',
label='True future')
    plt.plot(range(len(x), len(x) + len(y_pred)), y_pred, 'r--',
label='Predicted future')
    plt.axvline(x=len(x), color='k', linestyle='--')
    plt.grid(True)
    plt.legend()
    plt.title(title)
    plt.xlabel('Time Steps')
    plt.ylabel('Value')

    if filename:
        plt.savefig(filename)

    plt.show()

# Generate predictions for specific samples
print("\nGenerating predictions for sample time series...")

# 5th training sample (index 4)
train_idx = 4
X_sample_train = X_train[train_idx:train_idx+1]
y_sample_train = y_train[train_idx]

# 10th validation sample (index 9)
valid_idx = 9
X_sample_valid = X_valid[valid_idx:valid_idx+1]
y_sample_valid = y_valid[valid_idx]

# Make predictions
y_pred_linear_train = linear_model.predict(X_sample_train)[0]
```

```
y_pred_lstm_train = lstm_model.predict(X_sample_train)[0]
y_pred_rnn_train = rnn_model.predict(X_sample_train)[0]

y_pred_linear_valid = linear_model.predict(X_sample_valid)[0]
y_pred_lstm_valid = lstm_model.predict(X_sample_valid)[0]
y_pred_rnn_valid = rnn_model.predict(X_sample_valid)[0]

# Plot results for the 5th training sample
print("\nPlotting results for the 5th training sample...")
plot_series(
    X_train[train_idx, :, 0],
    y_train[train_idx],
    y_pred_linear_train,
    "Linear Regression: 5th Training Sample",
    "linear_train.png"
)

plot_series(
    X_train[train_idx, :, 0],
    y_train[train_idx],
    y_pred_lstm_train,
    "LSTM: 5th Training Sample",
    "lstm_train.png"
)

plot_series(
    X_train[train_idx, :, 0],
    y_train[train_idx],
    y_pred_rnn_train,
    "RNN: 5th Training Sample",
    "rnn_train.png"
)

# Plot results for the 10th validation sample
print("\nPlotting results for the 10th validation sample...")
plot_series(
    X_valid[valid_idx, :, 0],
    y_valid[valid_idx],
    y_pred_linear_valid,
    "Linear Regression: 10th Validation Sample",
    "linear_valid.png"
)

plot_series(
    X_valid[valid_idx, :, 0],
    y_valid[valid_idx],
    y_pred_lstm_valid,
    "LSTM: 10th Validation Sample",
    "lstm_valid.png"
)
```

```
plot_series(
    X_valid[valid_idx, :, 0],
    y_valid[valid_idx],
    y_pred_rnn_valid,
    "RNN: 10th Validation Sample",
    "rnn_valid.png"
)

# (d) Compare MSE among models
print("\n(d) Comparing MSE among models...")

# Calculate MSE for all samples
def calculate_mse(model, X, y_true):
    y_pred = model.predict(X)
    return np.mean(np.square(y_true - y_pred))

linear_train_mse = calculate_mse(linear_model, X_train, y_train)
linear_valid_mse = calculate_mse(linear_model, X_valid, y_valid)

lstm_train_mse = calculate_mse(lstm_model, X_train, y_train)
lstm_valid_mse = calculate_mse(lstm_model, X_valid, y_valid)

rnn_train_mse = calculate_mse(rnn_model, X_train, y_train)
rnn_valid_mse = calculate_mse(rnn_model, X_valid, y_valid)

# Print MSE results
print("\nMean Squared Error (MSE) Comparison:")
print(f"Linear Regression - Training MSE: {linear_train_mse:.6f},
Validation MSE: {linear_valid_mse:.6f}")
print(f"LSTM Model - Training MSE: {lstm_train_mse:.6f}, Validation
MSE: {lstm_valid_mse:.6f}")
print(f"RNN Model - Training MSE: {rnn_train_mse:.6f}, Validation MSE:
{rnn_valid_mse:.6f}")

# Create bar chart for MSE comparison
plt.figure(figsize=(12, 6))
models = ['Linear Regression', 'LSTM', 'RNN']
train_mse_values = [linear_train_mse, lstm_train_mse, rnn_train_mse]
valid_mse_values = [linear_valid_mse, lstm_valid_mse, rnn_valid_mse]

x = np.arange(len(models))
width = 0.35

plt.bar(x - width/2, train_mse_values, width, label='Training MSE')
plt.bar(x + width/2, valid_mse_values, width, label='Validation MSE')

plt.ylabel('Mean Squared Error (MSE)')
plt.title('MSE Comparison Among Models')
plt.xticks(x, models)
```

```python
plt.legend()
plt.grid(True, axis='y')
plt.savefig('mse_comparison.png')
plt.show()

# Compare learning curves
plt.figure(figsize=(12, 8))

plt.subplot(3, 1, 1)
plt.plot(linear_history.history['loss'], label='Training loss')
plt.plot(linear_history.history['val_loss'], label='Validation loss')
plt.title('Linear Regression Model Learning Curves')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid(True)
plt.legend()

plt.subplot(3, 1, 2)
plt.plot(lstm_history.history['loss'], label='Training loss')
plt.plot(lstm_history.history['val_loss'], label='Validation loss')
plt.title('LSTM Model Learning Curves')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid(True)
plt.legend()

plt.subplot(3, 1, 3)
plt.plot(rnn_history.history['loss'], label='Training loss')
plt.plot(rnn_history.history['val_loss'], label='Validation loss')
plt.title('RNN Model Learning Curves')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.savefig('learning_curves.png')
plt.show()

# Print model parameter counts
print("\nModel Parameter Comparison:")
print(f"Linear Regression: {linear_model.count_params()} parameters")
print(f"LSTM Model: {lstm_model.count_params()} parameters")
print(f"RNN Model: {rnn_model.count_params()} parameters")

print("\nAll tasks completed successfully!")
```

```
X_train shape: (3200, 80, 1)
y_train shape: (3200, 10)
X_valid shape: (800, 80, 1)
```

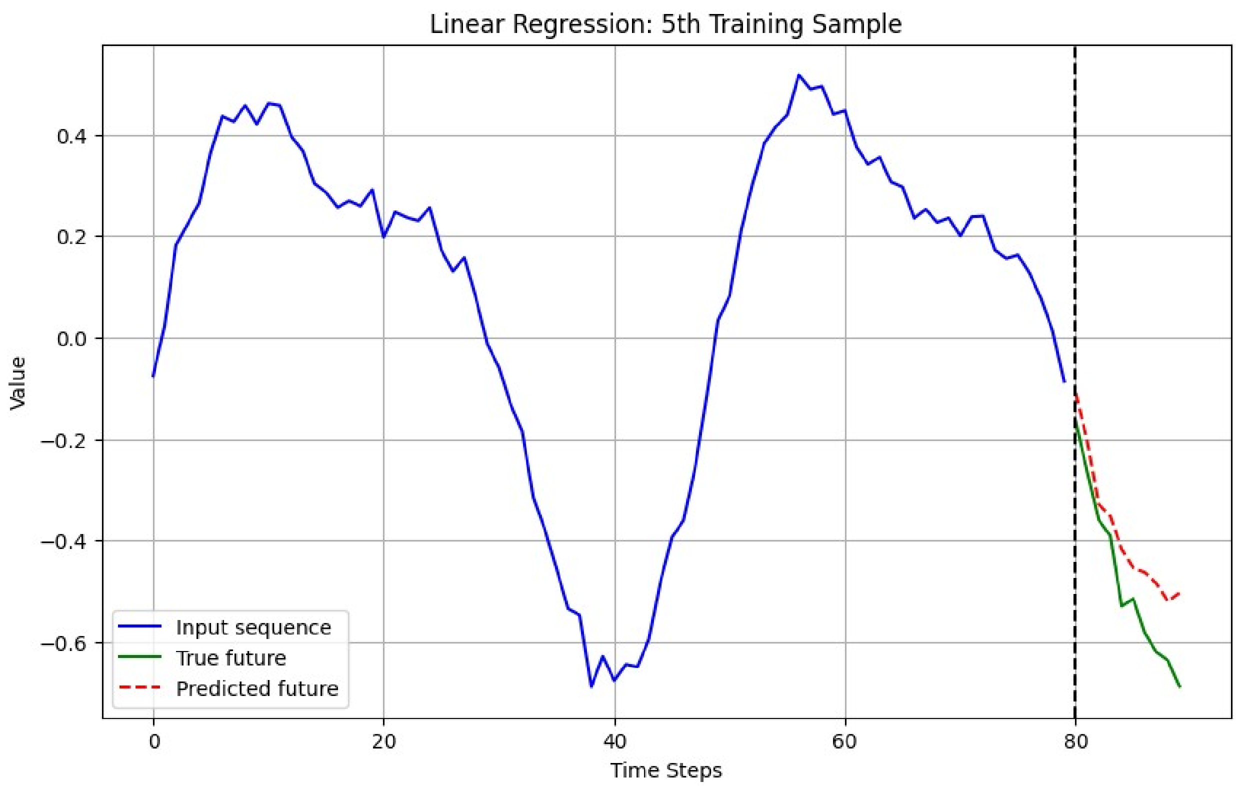
```
y_valid shape: (800, 10)

(a) Building and training linear regression model...
Epoch 1/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - loss: 0.1956 - val_loss:
0.0704
Epoch 2/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - loss: 0.0620 - val_loss:
0.0463
Epoch 3/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 32ms/step - loss: 0.0420 - val_loss:
0.0357
Epoch 4/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 4s 26ms/step - loss: 0.0329 - val_loss:
0.0298
Epoch 5/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 25ms/step - loss: 0.0279 - val_loss:
0.0261
Epoch 6/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 32ms/step - loss: 0.0248 - val_loss:
0.0236
Epoch 7/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - loss: 0.0227 - val_loss:
0.0219
Epoch 8/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - loss: 0.0213 - val_loss:
0.0207
Epoch 9/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step - loss: 0.0203 - val_loss:
0.0198
Epoch 10/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - loss: 0.0195 - val_loss:
0.0192
Epoch 11/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 25ms/step - loss: 0.0189 - val_loss:
0.0186
Epoch 12/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - loss: 0.0184 - val_loss:
0.0181
Epoch 13/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 34ms/step - loss: 0.0179 - val_loss:
0.0176
Epoch 14/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 4s 25ms/step - loss: 0.0175 - val_loss:
0.0172
Epoch 15/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - loss: 0.0171 - val_loss:
0.0168
Epoch 16/20
```

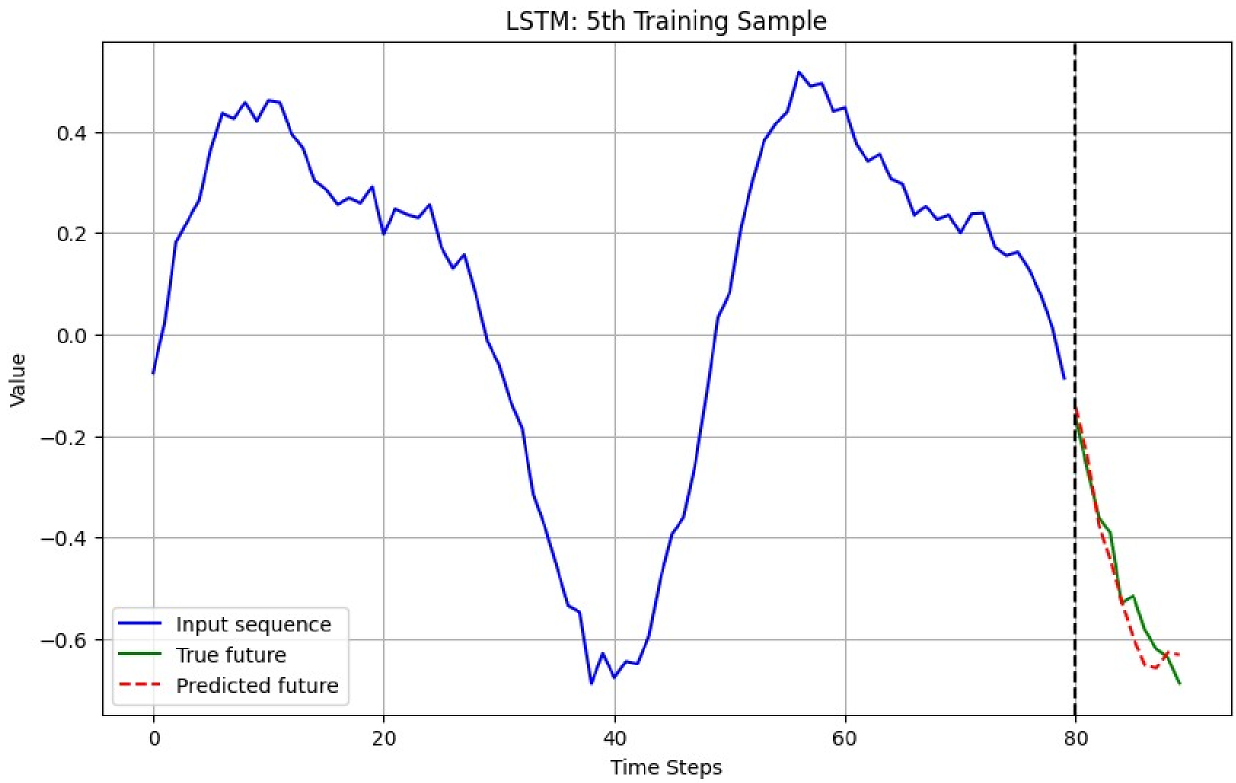
```
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 32ms/step - loss: 0.0167 - val_loss:
0.0165
Epoch 17/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - loss: 0.0163 - val_loss:
0.0161
Epoch 18/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 26ms/step - loss: 0.0160 - val_loss:
0.0158
Epoch 19/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - loss: 0.0156 - val_loss:
0.0155
Epoch 20/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - loss: 0.0153 - val_loss:
0.0152

(b) Building and training LSTM model...
Epoch 1/20

/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)

100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 95ms/step - loss: 0.1407 - val_loss:
0.0694
Epoch 2/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 93ms/step - loss: 0.0600 - val_loss:
0.0404
Epoch 3/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 89ms/step - loss: 0.0404 - val_loss:
0.0358
Epoch 4/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 99ms/step - loss: 0.0355 - val_loss:
0.0316
Epoch 5/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 103ms/step - loss: 0.0309 - val_loss:
0.0272
Epoch 6/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 96ms/step - loss: 0.0250 - val_loss:
0.0210
Epoch 7/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 89ms/step - loss: 0.0212 - val_loss:
0.0190
Epoch 8/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 102ms/step - loss: 0.0190 - val_loss:
0.0170
Epoch 9/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 95ms/step - loss: 0.0169 - val_loss:
0.0152
```

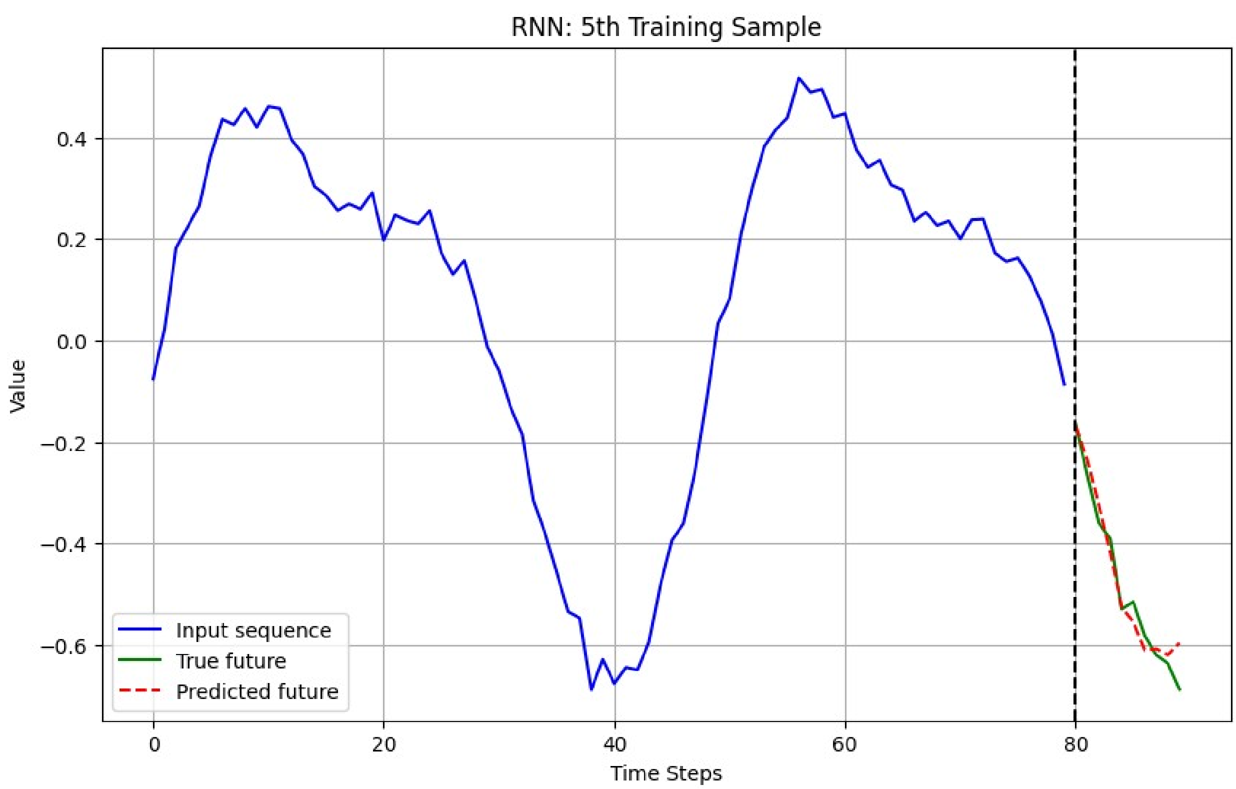
```
Epoch 10/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 101ms/step - loss: 0.0151 - val_loss:
0.0137
Epoch 11/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 95ms/step - loss: 0.0138 - val_loss:
0.0128
Epoch 12/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 93ms/step - loss: 0.0127 - val_loss:
0.0120
Epoch 13/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 94ms/step - loss: 0.0117 - val_loss:
0.0114
Epoch 14/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 93ms/step - loss: 0.0110 - val_loss:
0.0108
Epoch 15/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 101ms/step - loss: 0.0105 - val_loss:
0.0104
Epoch 16/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 102ms/step - loss: 0.0100 - val_loss:
0.0101
Epoch 17/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 101ms/step - loss: 0.0097 - val_loss:
0.0097
Epoch 18/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 98ms/step - loss: 0.0093 - val_loss:
0.0093
Epoch 19/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 87ms/step - loss: 0.0090 - val_loss:
0.0090
Epoch 20/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 90ms/step - loss: 0.0088 - val_loss:
0.0087

(c) Building and training RNN model...
Epoch 1/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 77s 759ms/step - loss: 0.1705 - val_loss:
0.0682
Epoch 2/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 76s 761ms/step - loss: 0.0545 - val_loss:
0.0351
Epoch 3/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 82s 759ms/step - loss: 0.0339 - val_loss:
0.0298
Epoch 4/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 81s 753ms/step - loss: 0.0299 - val_loss:
0.0277
Epoch 5/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 87s 875ms/step - loss: 0.0278 - val_loss:
```

```
0.0264
Epoch 6/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 132s 772ms/step - loss: 0.0266 -
val_loss: 0.0257
Epoch 7/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 76s 763ms/step - loss: 0.0257 - val_loss:
0.0251
Epoch 8/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 87s 869ms/step - loss: 0.0249 - val_loss:
0.0245
Epoch 9/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 130s 753ms/step - loss: 0.0242 -
val_loss: 0.0239
Epoch 10/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 86s 861ms/step - loss: 0.0237 - val_loss:
0.0234
Epoch 11/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 132s 760ms/step - loss: 0.0232 -
val_loss: 0.0228
Epoch 12/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 77s 775ms/step - loss: 0.0228 - val_loss:
0.0224
Epoch 13/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 80s 760ms/step - loss: 0.0224 - val_loss:
0.0219
Epoch 14/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 75s 752ms/step - loss: 0.0220 - val_loss:
0.0215
Epoch 15/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 77s 768ms/step - loss: 0.0217 - val_loss:
0.0211
Epoch 16/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 81s 754ms/step - loss: 0.0213 - val_loss:
0.0207
Epoch 17/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 92s 859ms/step - loss: 0.0210 - val_loss:
0.0204
Epoch 18/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 75s 751ms/step - loss: 0.0207 - val_loss:
0.0200
Epoch 19/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 83s 756ms/step - loss: 0.0203 - val_loss:
0.0196
Epoch 20/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 82s 760ms/step - loss: 0.0198 - val_loss:
0.0191

Generating predictions for sample time series...
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
```

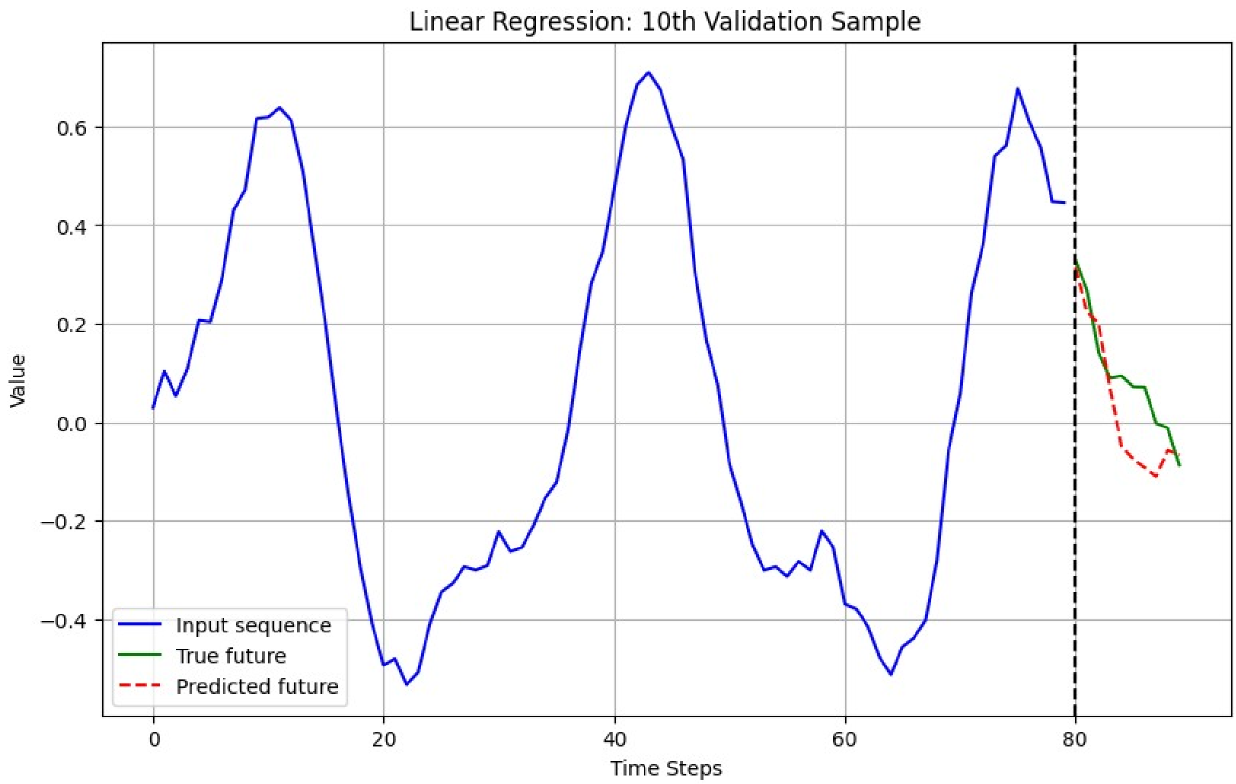
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 104ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 629ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 401ms/step

Plotting results for the 5th training sample...
```

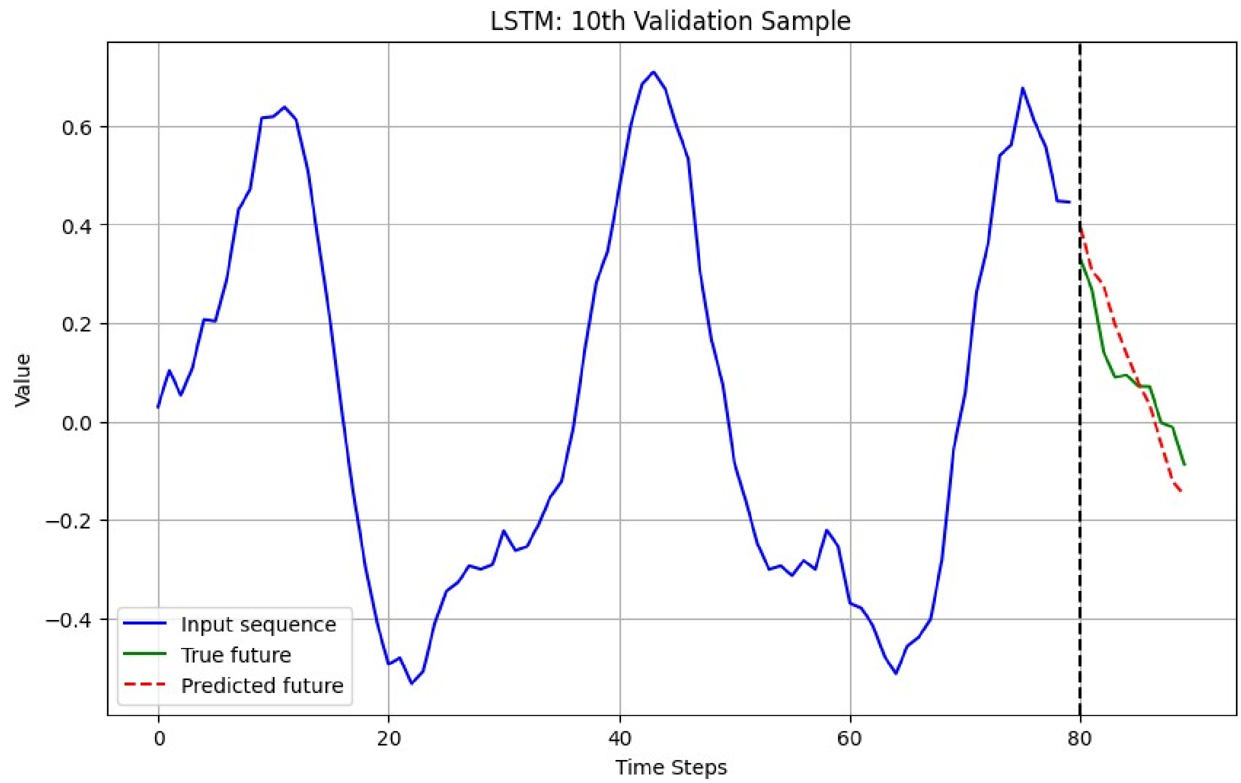


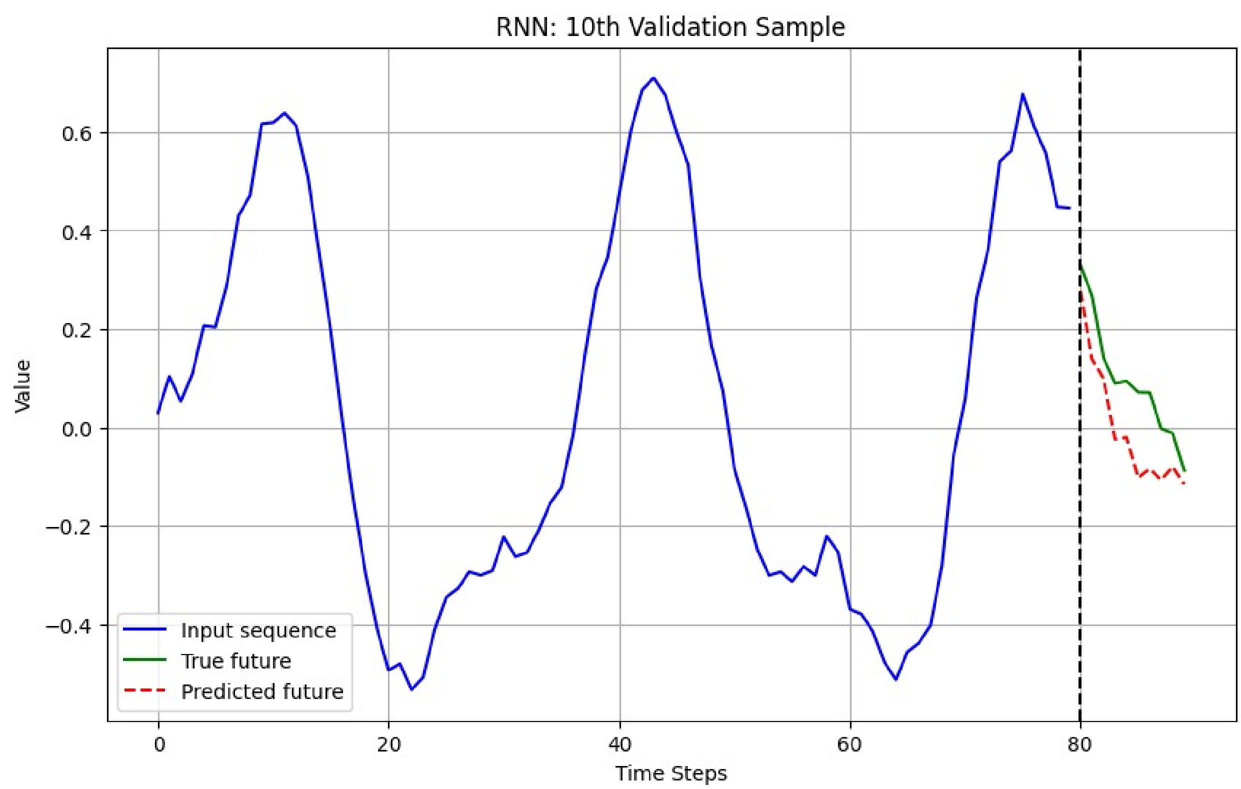

LSTM: 5th Training Sample
Value
Time Steps
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
20
40
60
80
Input sequence
True future
Predicted future


RNN: 5th Training Sample
Value
Time Steps
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
20
40
60
80
Input sequence
True future
Predicted future

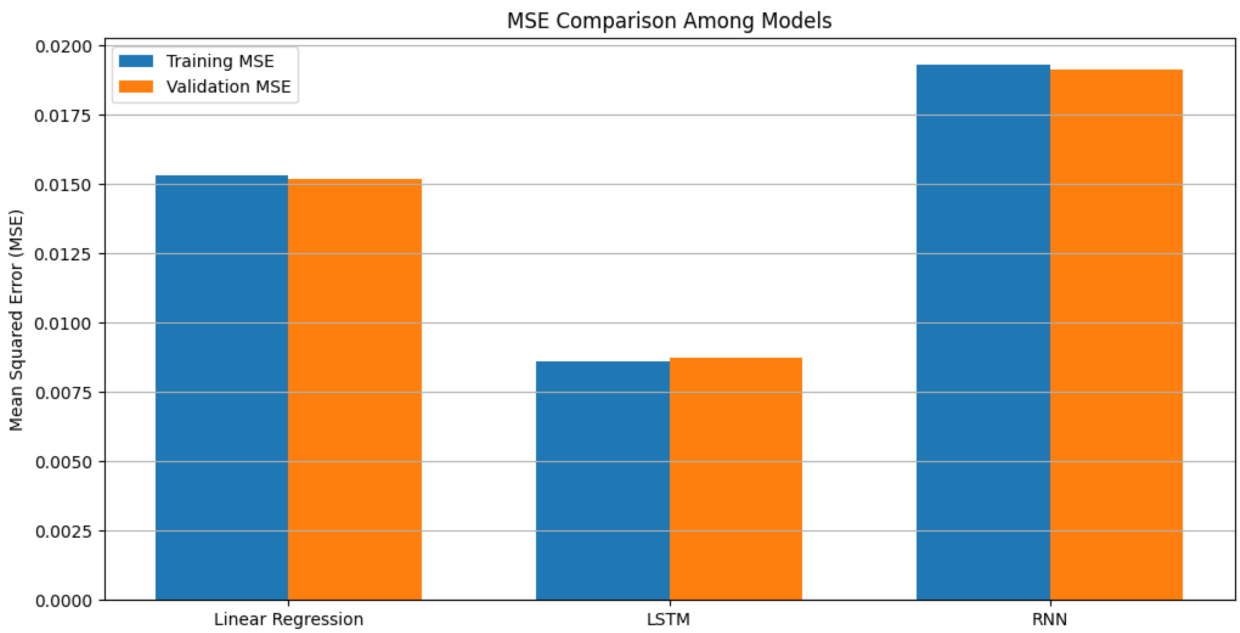
```
Plotting results for the 10th validation sample...
```

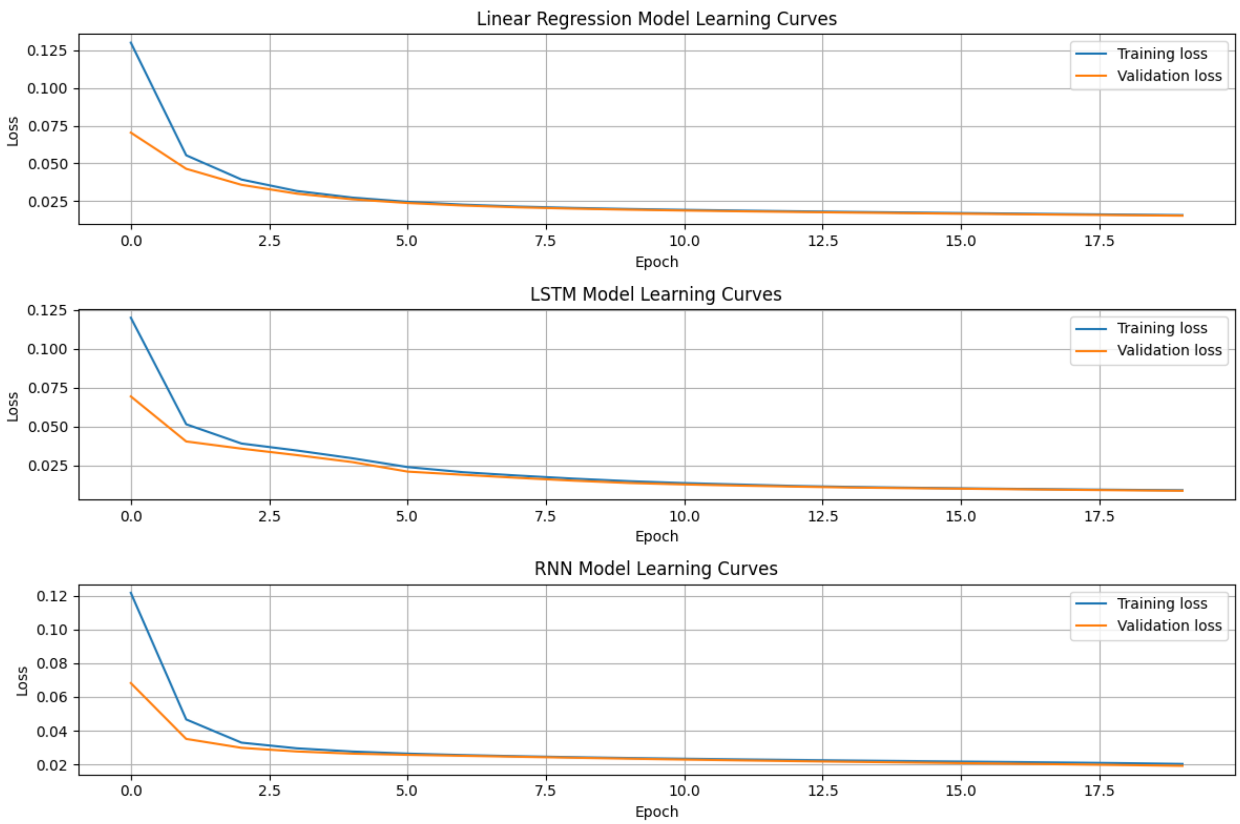



LSTM: 10th Validation Sample
Value
Time Steps
Input sequence
True future
Predicted future
0.6
0.4
0.2
0.0
−0.2
−0.4
0
20
40
60
80


RNN: 10th Validation Sample
Value
Time Steps
Input sequence
True future
Predicted future
0.6
0.4
0.2
0.0
−0.2
−0.4
0
20
40
60
80

```
(d) Comparing MSE among models...
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
100/100 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step
100/100 ━━━━━━━━━━━━━━━━━━━━ 39s 395ms/step
25/25 ━━━━━━━━━━━━━━━━━━━━ 10s 392ms/step

Mean Squared Error (MSE) Comparison:
Linear Regression - Training MSE: 0.015324, Validation MSE: 0.015172
LSTM Model - Training MSE: 0.008623, Validation MSE: 0.008735
RNN Model - Training MSE: 0.019286, Validation MSE: 0.019135
```






```
Model Parameter Comparison:
Linear Regression: 810 parameters
LSTM Model: 1430 parameters
RNN Model: 440 parameters

All tasks completed successfully!
```