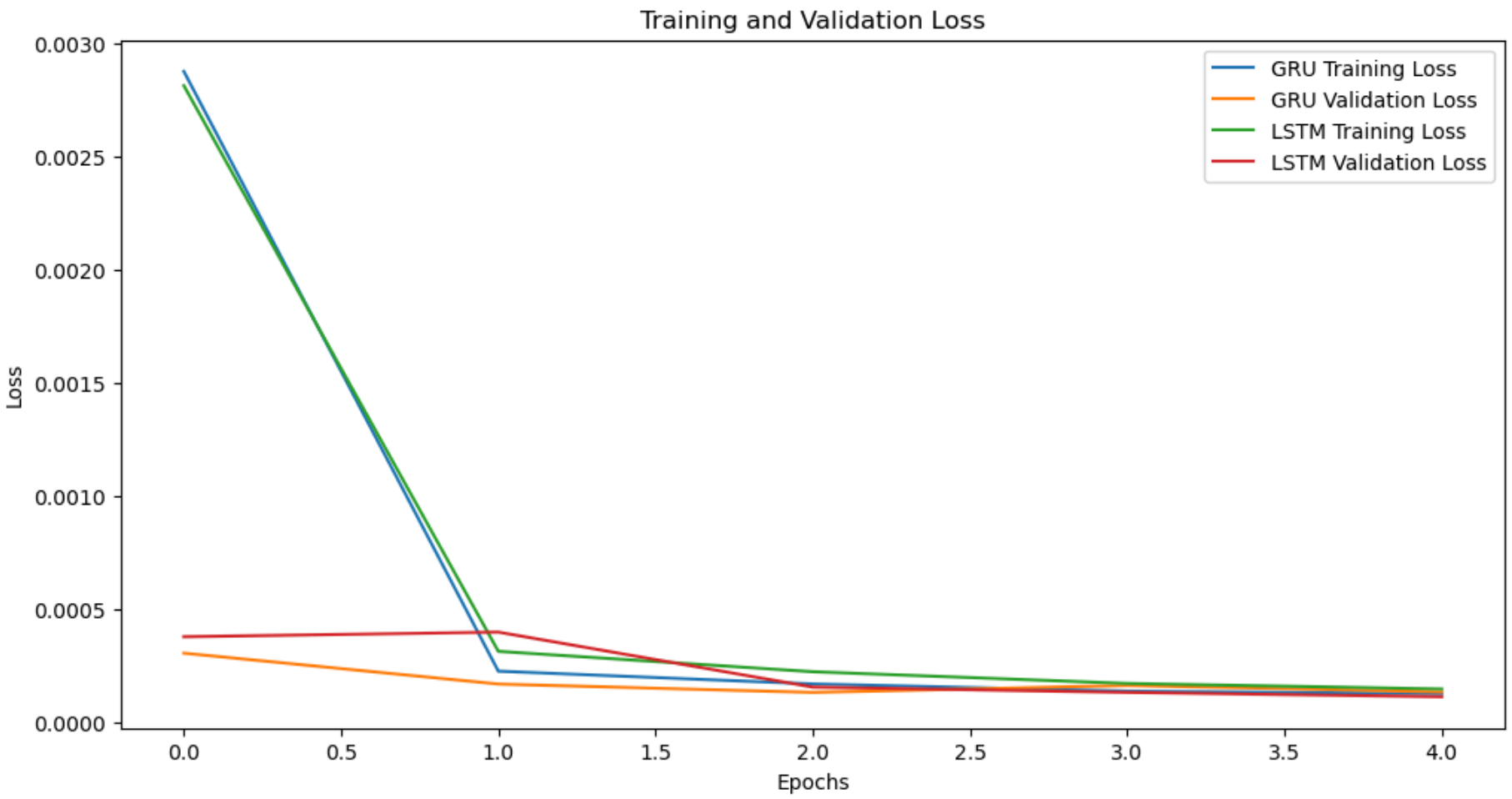
In [19]:
```
#Experiment 1
```

In [17]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GRU, LSTM
from tensorflow.keras.optimizers import Adam

# Example: Loading electricity consumption data
data = pd.read_csv('C:/Users/Syed Umar/Downloads/powerconsumption.csv')  # Replace with your dataset's path
data = data['PowerConsumption_Zone2'].values  # Replace name with your column name
data = data.reshape(-1, 1)

# Normalize the data using MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(data)

# Prepare the dataset for time series prediction
def create_sequences(data, sequence_length):
    X, y = [], []
    for i in range(len(data) - sequence_length):
        X.append(data[i:i + sequence_length])
        y.append(data[i + sequence_length])
    return np.array(X), np.array(y)

sequence_length = 50
X, y = create_sequences(data_scaled, sequence_length)

# Split into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# GRU Model
def build_gru_model(input_shape):
    model = Sequential([
        GRU(64, return_sequences=False, input_shape=input_shape),
        Dense(1)
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
    return model

# LSTM Model
def build_lstm_model(input_shape):
    model = Sequential([
        LSTM(64, return_sequences=False, input_shape=input_shape),
        Dense(1)
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
    return model

# Build and train the GRU model
gru_model = build_gru_model((sequence_length, 1))
gru_history = gru_model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=5, batch_size=32, verbose=1)

# Build and train the LSTM model
lstm_model = build_lstm_model((sequence_length, 1))
lstm_history = lstm_model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=5, batch_size=32, verbose=1)

# Evaluate the models
gru_loss = gru_model.evaluate(X_test, y_test, verbose=0)
lstm_loss = lstm_model.evaluate(X_test, y_test, verbose=0)

print(f"GRU Test Loss: {gru_loss}")
print(f"LSTM Test Loss: {lstm_loss}")

# Plot training and validation loss
plt.figure(figsize=(12, 6))
plt.plot(gru_history.history['loss'], label='GRU Training Loss')
plt.plot(gru_history.history['val_loss'], label='GRU Validation Loss')
plt.plot(lstm_history.history['loss'], label='LSTM Training Loss')
plt.plot(lstm_history.history['val_loss'], label='LSTM Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 1/5
```

```
C:\Anaconda\Lib\site-packages\keras\src\layers\rnn\rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argument t
o a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
1310/1310 ━━━━━━━━━━━━━━━━━━━━ 16s 11ms/step - loss: 0.0133 - val_loss: 3.0354e-04
Epoch 2/5
1310/1310 ━━━━━━━━━━━━━━━━━━━━ 14s 11ms/step - loss: 2.5612e-04 - val_loss: 1.6682e-04
Epoch 3/5
1310/1310 ━━━━━━━━━━━━━━━━━━━━ 14s 11ms/step - loss: 1.7569e-04 - val_loss: 1.2982e-04
Epoch 4/5
1310/1310 ━━━━━━━━━━━━━━━━━━━━ 15s 11ms/step - loss: 1.4028e-04 - val_loss: 1.6053e-04
Epoch 5/5
1310/1310 ━━━━━━━━━━━━━━━━━━━━ 14s 11ms/step - loss: 1.3308e-04 - val_loss: 1.3178e-04
Epoch 1/5
1310/1310 ━━━━━━━━━━━━━━━━━━━━ 15s 11ms/step - loss: 0.0112 - val_loss: 3.7621e-04
Epoch 2/5
1310/1310 ━━━━━━━━━━━━━━━━━━━━ 14s 11ms/step - loss: 3.5535e-04 - val_loss: 3.9666e-04
Epoch 3/5
1310/1310 ━━━━━━━━━━━━━━━━━━━━ 13s 10ms/step - loss: 2.3959e-04 - val_loss: 1.5333e-04
Epoch 4/5
1310/1310 ━━━━━━━━━━━━━━━━━━━━ 13s 10ms/step - loss: 1.7766e-04 - val_loss: 1.2984e-04
Epoch 5/5
1310/1310 ━━━━━━━━━━━━━━━━━━━━ 14s 10ms/step - loss: 1.4885e-04 - val_loss: 1.1067e-04
GRU Test Loss: 0.00013177732762414962
LSTM Test Loss: 0.00011066991282859817
```



In [20]:
```
#Experiment 2
```

In [21]:
```
import tensorflow as tf
import matplotlib.pyplot as plt

# Load a sample dataset (e.g., MNIST for demonstration)
(mnist_images, mnist_labels), (_, _) = tf.keras.datasets.mnist.load_data()

# Display the original image
plt.imshow(mnist_images[0], cmap='gray')
plt.title("Original Image")
plt.axis("off")
plt.show()

# Expand dimensions for compatibility with TensorFlow operations
mnist_images = tf.expand_dims(mnist_images, axis=-1)  # Add a channel dimension

# Convert the dataset to a tf.data.Dataset
dataset = tf.data.Dataset.from_tensor_slices((mnist_images, mnist_labels))

# Shuffle and batch the dataset
dataset = dataset.shuffle(buffer_size=10000).batch(32)

# Data augmentation functionzzzz
def augment_image(image, label):
    image = tf.image.flip_up_down(image)  # Flip the image vertically
    return image, label

# Apply data augmentation
dataset = dataset.map(augment_image)

# One-hot encode the labels
def one_hot_encode(image, label):
    label = tf.one_hot(label, depth=10)  # MNIST has 10 classes
    label = tf.squeeze(label)  # Remove unnecessary dimensions
    return image, label
```

```
dataset = dataset.map(one_hot_encode)

# Get an image and label from the processed dataset
for processed_image, processed_label in dataset.take(1):
    example_image = processed_image[0]
    break

# Convert the processed image to a tensor
example_image = tf.cast(example_image, dtype=tf.float32)
image_tensor = tf.convert_to_tensor(example_image)

# Resize and normalize the processed image
image_resized = tf.image.resize(image_tensor, [128, 128])  # Resize to 128x128
image_normalized = image_resized / 255.0  # Normalize to [0, 1]

# Display the processed image
plt.imshow(tf.squeeze(image_normalized), cmap='gray')
plt.title("Processed Image")
plt.axis("off")
plt.show()
```