```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv1D, MaxPooling1D, UpSampling1D, LSTM, Repe
```

```
# Load data
data = pd.read_excel('ex1.xlsx')
original_amplitude = data['Amplitude'].values
noised_amplitude = data['Amplitude_noise'].values

# Reshape and scale data
original_amplitude = original_amplitude.reshape(-1, 1)
noised_amplitude = noised_amplitude.reshape(-1, 1)

scaler = MinMaxScaler()
original_amplitude = scaler.fit_transform(original_amplitude)
noised_amplitude = scaler.transform(noised_amplitude)

# Windowing function
window_size = 4500

def create_windows(data, window_size):
    windows = []
    for i in range(len(data) - window_size + 1):
        windows.append(data[i:i + window_size])
    return np.array(windows)

X = create_windows(noised_amplitude, window_size)
y = create_windows(original_amplitude, window_size)

# Reshape for training
X_train = X.reshape(-1, window_size, 1)
y_train = y.reshape(-1, window_size, 1)
```

```
# Conv1D Autoencoder Model
input_signal = Input(shape=(window_size, 1))
x = Conv1D(16, 3, activation='relu', padding='same')(input_signal)
x = MaxPooling1D(2, padding='same')(x)
x = Conv1D(8, 3, activation='relu', padding='same')(x)
encoded = MaxPooling1D(2, padding='same')(x)

x = Conv1D(8, 3, activation='relu', padding='same')(encoded)
x = UpSampling1D(2)(x)
x = Conv1D(16, 3, activation='relu', padding='same')(x)
x = UpSampling1D(2)(x)
decoded = Conv1D(1, 3, activation='relu', padding='same')(x)
```

```
autoencoder = Model(input_signal, decoded)
autoencoder.compile(optimizer='adam', loss='mean_squared_error')

# Train Conv1D Autoencoder
autoencoder.fit(X_train, y_train, epochs=20, batch_size=64, validation_split=0.2)
denoised_signal = autoencoder.predict(X_train)
denoised_signal = scaler.inverse_transform(denoised_signal.reshape(-1, 1))
```

```
Epoch 1/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 6s 358ms/step - loss: 0.3554 - val_loss: 0.3041
Epoch 2/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 4s 218ms/step - loss: 0.2737 - val_loss: 0.2015
Epoch 3/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 243ms/step - loss: 0.1694 - val_loss: 0.0906
Epoch 4/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 232ms/step - loss: 0.0738 - val_loss: 0.0598
Epoch 5/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 309ms/step - loss: 0.0626 - val_loss: 0.0531
Epoch 6/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 212ms/step - loss: 0.0496 - val_loss: 0.0456
Epoch 7/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 227ms/step - loss: 0.0435 - val_loss: 0.0361
Epoch 8/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 229ms/step - loss: 0.0338 - val_loss: 0.0285
Epoch 9/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 238ms/step - loss: 0.0266 - val_loss: 0.0203
Epoch 10/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 224ms/step - loss: 0.0187 - val_loss: 0.0132
Epoch 11/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 367ms/step - loss: 0.0119 - val_loss: 0.0076
Epoch 12/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 216ms/step - loss: 0.0068 - val_loss: 0.0041
Epoch 13/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 243ms/step - loss: 0.0037 - val_loss: 0.0025
Epoch 14/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 223ms/step - loss: 0.0024 - val_loss: 0.0020
Epoch 15/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 216ms/step - loss: 0.0020 - val_loss: 0.0018
Epoch 16/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 294ms/step - loss: 0.0018 - val_loss: 0.0016
Epoch 17/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 344ms/step - loss: 0.0017 - val_loss: 0.0016
Epoch 18/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 223ms/step - loss: 0.0017 - val_loss: 0.0016
Epoch 19/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 248ms/step - loss: 0.0017 - val_loss: 0.0016
Epoch 20/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 219ms/step - loss: 0.0017 - val_loss: 0.0016
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step
```

```
# LSTM Autoencoder Model
input_signal_lstm = Input(shape=(window_size, 1))
encoded_lstm = LSTM(64, activation='relu', return_sequences=True)(input_signal_lstm)
encoded_lstm = LSTM(32, activation='relu', return_sequences=False)(encoded_lstm)

decoded_lstm = RepeatVector(window_size)(encoded_lstm)
```

```
decoded_lstm = LSTM(32, activation='relu', return_sequences=True)(decoded_lstm)
decoded_lstm = LSTM(64, activation='relu', return_sequences=True)(decoded_lstm)
decoded_lstm = Conv1D(1, 3, activation='relu', padding='same')(decoded_lstm)

lstm_autoencoder = Model(input_signal_lstm, decoded_lstm)
lstm_autoencoder.compile(optimizer='adam', loss='mean_squared_error')

# Train LSTM Autoencoder
history_lstm = lstm_autoencoder.fit(X_train, y_train, epochs=20, batch_size=64, validatic
denoised_signal_lstm = lstm_autoencoder.predict(X_train)
denoised_signal_lstm = scaler.inverse_transform(denoised_signal_lstm.reshape(-1, 1))
```

```
Epoch 1/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 133s 18s/step - loss: 0.3430 - val_loss: 0.2608
Epoch 2/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 144s 18s/step - loss: 0.2227 - val_loss: 0.1387
Epoch 3/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 139s 18s/step - loss: 0.1533 - val_loss: 0.1367
Epoch 4/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 147s 19s/step - loss: 0.1399 - val_loss: 0.1315
Epoch 5/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 139s 18s/step - loss: 0.1364 - val_loss: 0.1284
Epoch 6/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 139s 18s/step - loss: 0.1323 - val_loss: 0.1286
Epoch 7/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 147s 19s/step - loss: 0.1294 - val_loss: 0.1285
Epoch 8/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 137s 18s/step - loss: 0.1285 - val_loss: 0.1278
Epoch 9/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 131s 19s/step - loss: 0.1268 - val_loss: 0.1277
Epoch 10/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 127s 18s/step - loss: 0.1262 - val_loss: 0.1279
Epoch 11/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 137s 17s/step - loss: 0.1256 - val_loss: 0.1282
Epoch 12/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 143s 17s/step - loss: 0.1255 - val_loss: 0.1280
Epoch 13/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 144s 18s/step - loss: 0.1255 - val_loss: 0.1280
Epoch 14/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 150s 19s/step - loss: 0.1255 - val_loss: 0.1279
Epoch 15/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 137s 18s/step - loss: 0.1254 - val_loss: 0.1279
Epoch 16/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 138s 17s/step - loss: 0.1254 - val_loss: 0.1278
Epoch 17/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 124s 18s/step - loss: 0.1254 - val_loss: 0.1279
Epoch 18/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 131s 19s/step - loss: 0.1255 - val_loss: 0.1279
Epoch 19/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 127s 18s/step - loss: 0.1254 - val_loss: 0.1281
Epoch 20/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 135s 19s/step - loss: 0.1252 - val_loss: 0.1279
16/16 ━━━━━━━━━━━━━━━━━━━━ 29s 2s/step
```

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv1D, MaxPooling1D, UpSampling1D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint
```

```
# Define the input layer
input_signal = Input(shape=(4500, 1))

# Build the convolutional layers
x = Conv1D(16, kernel_size=3, activation='relu', padding='same')(input_signal)
x = MaxPooling1D(pool_size=2, padding='same')(x)
x = Conv1D(8, kernel_size=3, activation='relu', padding='same')(x)
encoded = MaxPooling1D(pool_size=2, padding='same')(x)

# Build the upsampling layers
x = Conv1D(8, kernel_size=3, activation='relu', padding='same')(encoded)
x = UpSampling1D(size=2)(x)
x = Conv1D(16, kernel_size=3, activation='relu', padding='same')(x)
x = UpSampling1D(size=2)(x)
decoded = Conv1D(1, kernel_size=3, activation='relu', padding='same')(x)

# Compile the model
conv2_autoencoder = Model(inputs=input_signal, outputs=decoded)
conv2_autoencoder.compile(optimizer=Adam(), loss='mean_squared_error')


# Define checkpoint callback
checkpoint_path = "checkpoints/denoiser_model.h5"
checkpoint = ModelCheckpoint(filepath=checkpoint_path,
                             monitor='val_loss',
                             save_best_only=True,
                             verbose=1)
```

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

# ✅ Load your trained model
model = load_model("checkpoints/denoiser_model.h5")

# ✅ Prepare your input (reshape as needed)
# Example: assume 'data["Amplitude_noise"]' is the noisy signal
# X_noisy = data['Amplitude_noise'].values.reshape(1, -1, 1)  # shape: (1, time_steps, 1)

# Apply the same windowing as used for training
X_noisy_windowed = create_windows(data['Amplitude_noise'].values.reshape(-1, 1), window_s
X_noisy_windowed = X_noisy_windowed.reshape(-1, window_size, 1)


# ✅ Predict denoised output
denoised_signal_conv2 = model.predict(X_noisy_windowed)

# The prediction output will be windowed, we need to combine it back.
```

```
# This simple approach takes the first window's prediction as the start
# and then appends the last value of subsequent window predictions.
# A more sophisticated approach might involve averaging overlapping regions.
denoised_signal_conv2_combined = []
for i in range(len(denoised_signal_conv2)):
  if i == 0:
    denoised_signal_conv2_combined.extend(denoised_signal_conv2[i].flatten().tolist())
  else:
    denoised_signal_conv2_combined.append(denoised_signal_conv2[i].flatten()[-1])

denoised_signal_conv2 = np.array(denoised_signal_conv2_combined)


# ✅ Flatten for plotting
# denoised_signal_conv2 = denoised_signal_conv2.flatten() # Already flattened during comb
```

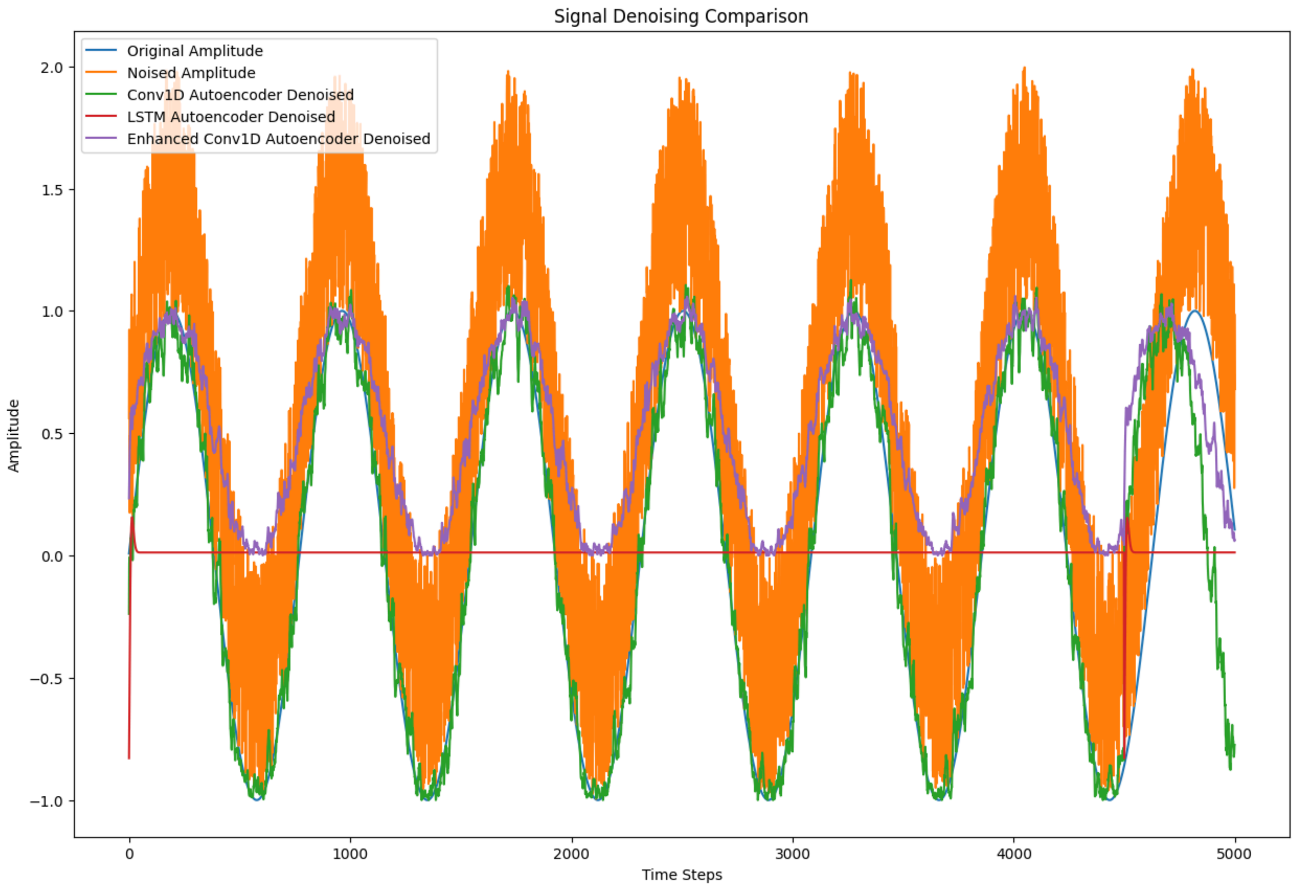
```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 74ms/step
```

```
# Train the model

history_enhanced_conv1d = conv2_autoencoder.fit(X_train, y_train,
                      epochs=20,
                      batch_size=64,
                      validation_split=0.2,
                      callbacks=[checkpoint])


# Predict using the trained (or best saved) model
denoised_signal_conv2 = conv2_autoencoder.predict(X_train)
```

```
Epoch 1/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 256ms/step - loss: 0.2308
Epoch 1: val_loss improved from inf to 0.12392, saving model to checkpoints/denois
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `kera
7/7 ━━━━━━━━━━━━━━━━━━━━ 6s 368ms/step - loss: 0.2277 - val_loss: 0.1239
Epoch 2/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 377ms/step - loss: 0.0976
Epoch 2: val_loss improved from 0.12392 to 0.03911, saving model to checkpoints/de
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `kera
7/7 ━━━━━━━━━━━━━━━━━━━━ 6s 461ms/step - loss: 0.0955 - val_loss: 0.0391
Epoch 3/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 228ms/step - loss: 0.0377
Epoch 3: val_loss did not improve from 0.03911
7/7 ━━━━━━━━━━━━━━━━━━━━ 4s 288ms/step - loss: 0.0377 - val_loss: 0.0415
Epoch 4/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 201ms/step - loss: 0.0386
Epoch 4: val_loss improved from 0.03911 to 0.02131, saving model to checkpoints/de
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `kera
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 243ms/step - loss: 0.0382 - val_loss: 0.0213
Epoch 5/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 224ms/step - loss: 0.0209
Epoch 5: val_loss improved from 0.02131 to 0.01983, saving model to checkpoints/de
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `kera
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 262ms/step - loss: 0.0209 - val_loss: 0.0198
```

```
Epoch 6/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 209ms/step - loss: 0.0170
Epoch 6: val_loss improved from 0.01983 to 0.00857, saving model to checkpoints/de
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `kera
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 271ms/step - loss: 0.0167 - val_loss: 0.0086
Epoch 7/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 301ms/step - loss: 0.0079
Epoch 7: val_loss improved from 0.00857 to 0.00378, saving model to checkpoints/de
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `kera
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 343ms/step - loss: 0.0078 - val_loss: 0.0038
Epoch 8/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 244ms/step - loss: 0.0034
Epoch 8: val_loss improved from 0.00378 to 0.00232, saving model to checkpoints/de
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `kera
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 282ms/step - loss: 0.0033 - val_loss: 0.0023
Epoch 9/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 215ms/step - loss: 0.0023
Epoch 9: val_loss improved from 0.00232 to 0.00210, saving model to checkpoints/de
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `kera
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 257ms/step - loss: 0.0023 - val_loss: 0.0021
Epoch 10/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 204ms/step - loss: 0.0023
Epoch 10: val_loss improved from 0.00210 to 0.00195, saving model to checkpoints/d
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `kera
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 242ms/step - loss: 0.0023 - val_loss: 0.0020
Epoch 11/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 226ms/step - loss: 0.0022
Epoch 11: val_loss improved from 0.00195 to 0.00174, saving model to checkpoints/d
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `kera
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 266ms/step - loss: 0.0022 - val_loss: 0.0017
Epoch 12/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 199ms/step - loss: 0.0019
```

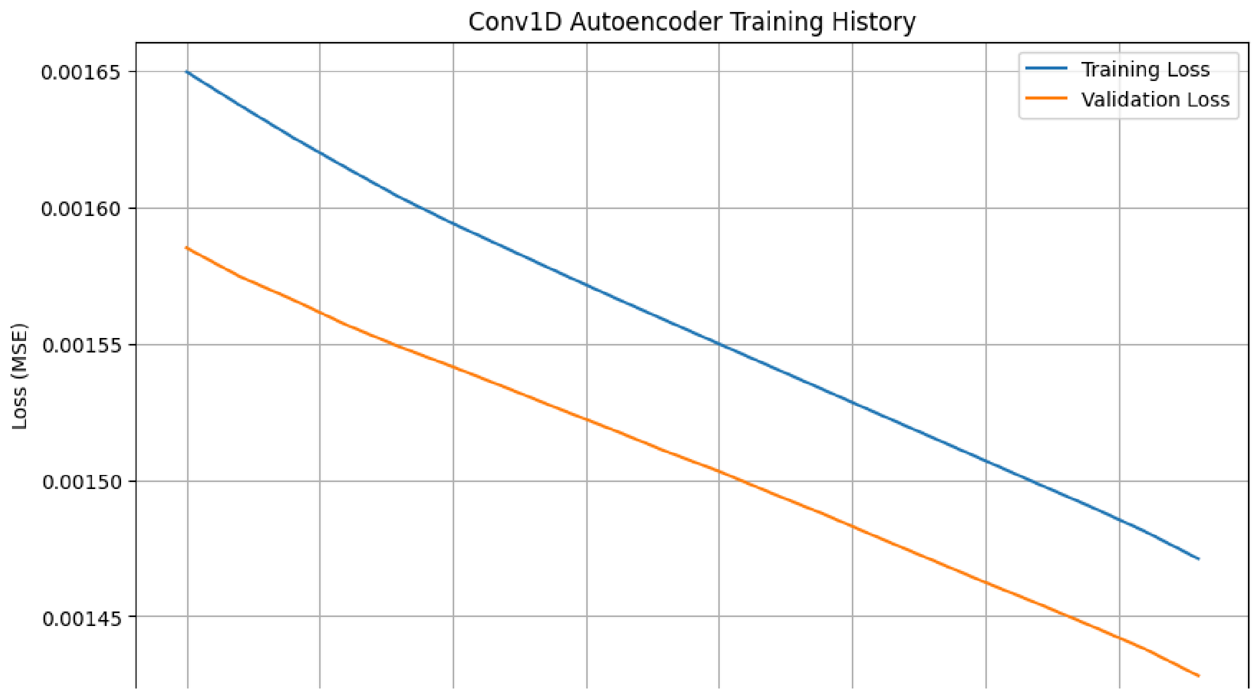
```python
# Visualization
time_steps = np.arange(len(data))
plt.figure(figsize=(15, 10))
plt.plot(time_steps, data['Amplitude'], label='Original Amplitude')
plt.plot(time_steps, data['Amplitude_noise'], label='Noised Amplitude')
plt.plot(time_steps, denoised_signal.flatten()[:len(data)], label='Conv1D Autoencoder Den
plt.plot(time_steps, denoised_signal_lstm.flatten()[:len(data)], label='LSTM Autoencoder
plt.plot(time_steps, denoised_signal_conv2.flatten()[:len(data)], label='Enhanced Conv1D

plt.xlabel('Time Steps')
plt.ylabel('Amplitude')
plt.title('Signal Denoising Comparison')
plt.legend()
plt.show()
```

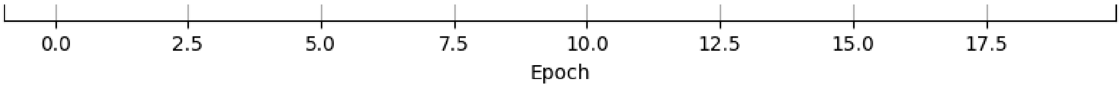


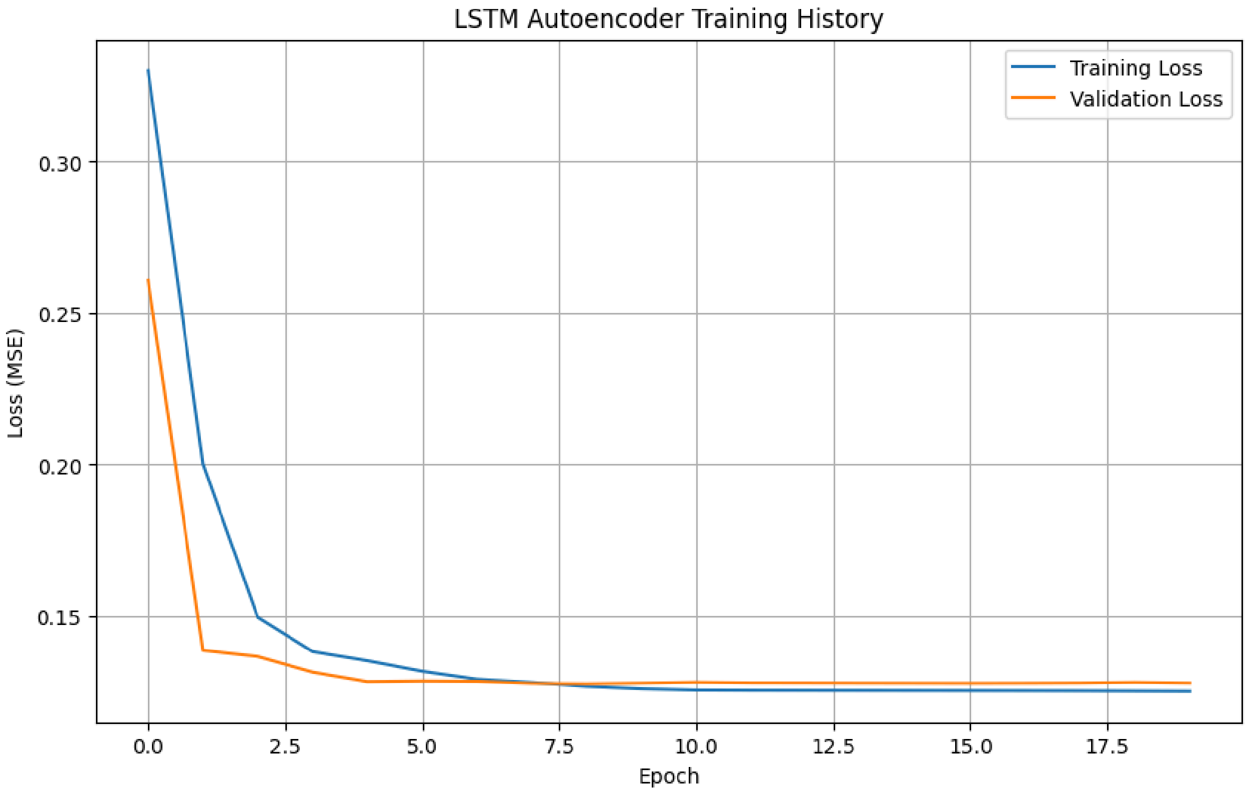

```
import matplotlib.pyplot as plt

# --- For Conv1D Autoencoder ---
# Assuming 'history_conv1d' is the History object from your first Conv1D model's training
history_conv1d = autoencoder.fit(X_train, y_train, epochs=20, batch_size=64, validation_s
plt.figure(figsize=(10, 6))
plt.plot(history_conv1d.history['loss'], label='Training Loss')
plt.plot(history_conv1d.history['val_loss'], label='Validation Loss')
plt.title('Conv1D Autoencoder Training History')
plt.xlabel('Epoch')
plt.ylabel('Loss (MSE)')
plt.legend()
```

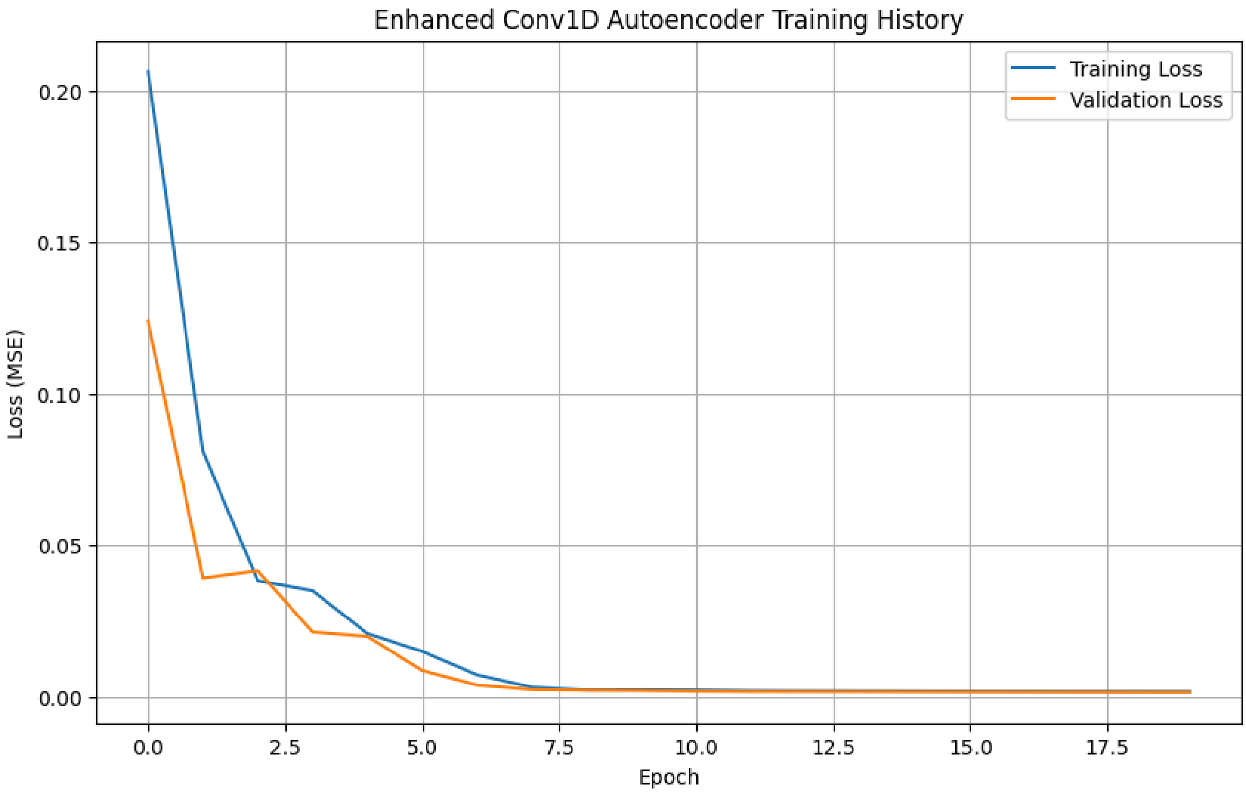
```
plt.grid(True)
plt.show()

# --- For LSTM Autoencoder ---
# Assuming 'history_lstm' is the History object from your LSTM model's training
plt.figure(figsize=(10, 6))
plt.plot(history_lstm.history['loss'], label='Training Loss')
plt.plot(history_lstm.history['val_loss'], label='Validation Loss')
plt.title('LSTM Autoencoder Training History')
plt.xlabel('Epoch')
plt.ylabel('Loss (MSE)')
plt.legend()
plt.grid(True)
plt.show()

# --- For Enhanced Conv1D Autoencoder ---
# Assuming 'history_enhanced_conv1d' is the History object from your enhanced Conv1D mode
plt.figure(figsize=(10, 6))
plt.plot(history_enhanced_conv1d.history['loss'], label='Training Loss')
plt.plot(history_enhanced_conv1d.history['val_loss'], label='Validation Loss')
plt.title('Enhanced Conv1D Autoencoder Training History')
plt.xlabel('Epoch')
plt.ylabel('Loss (MSE)')
plt.legend()
plt.grid(True)
plt.show()
```

```
Epoch 1/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 239ms/step - loss: 0.0016 - val_loss: 0.0016
Epoch 2/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 224ms/step - loss: 0.0016 - val_loss: 0.0016
Epoch 3/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 227ms/step - loss: 0.0016 - val_loss: 0.0016
Epoch 4/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 221ms/step - loss: 0.0016 - val_loss: 0.0016
Epoch 5/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 252ms/step - loss: 0.0016 - val_loss: 0.0015
Epoch 6/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 4s 341ms/step - loss: 0.0016 - val_loss: 0.0015
Epoch 7/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 4s 209ms/step - loss: 0.0016 - val_loss: 0.0015
Epoch 8/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 219ms/step - loss: 0.0016 - val_loss: 0.0015
Epoch 9/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 242ms/step - loss: 0.0016 - val_loss: 0.0015
Epoch 10/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 314ms/step - loss: 0.0016 - val_loss: 0.0015
Epoch 11/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 293ms/step - loss: 0.0015 - val_loss: 0.0015
Epoch 12/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 232ms/step - loss: 0.0015 - val_loss: 0.0015
Epoch 13/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 215ms/step - loss: 0.0015 - val_loss: 0.0015
Epoch 14/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 218ms/step - loss: 0.0015 - val_loss: 0.0015
Epoch 15/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 215ms/step - loss: 0.0015 - val_loss: 0.0015
Epoch 16/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 4s 353ms/step - loss: 0.0015 - val_loss: 0.0015
Epoch 17/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 217ms/step - loss: 0.0015 - val_loss: 0.0015
Epoch 18/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 215ms/step - loss: 0.0015 - val_loss: 0.0014
Epoch 19/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 246ms/step - loss: 0.0015 - val_loss: 0.0014
Epoch 20/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 232ms/step - loss: 0.0015 - val_loss: 0.0014
```




0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch


LSTM Autoencoder Training History
Training Loss
Validation Loss
Loss (MSE)
0.30
0.25
0.20
0.15
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch


Enhanced Conv1D Autoencoder Training History
Training Loss
Validation Loss
Loss (MSE)
0.20
0.15
0.10
0.05
0.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch

```
import numpy as np

# --- Prepare data for comparison ---
# Ensure all signals are of the same length and reshaped for calculations if needed
# Your denoised_signal, denoised_signal_lstm, denoised_signal_conv2 are likely 3D or 2D
# We need them flattened and clipped to the original data length for fair comparison
original_signal_flat = original_amplitude.flatten()[:len(data)]
noisy_signal_flat = noised_amplitude.flatten()[:len(data)]

# Flatten the denoised outputs and match original data length
# Assuming your denoised_signal, denoised_signal_lstm, denoised_signal_conv2 are 2D array
denoised_conv1d_flat = denoised_signal.flatten()[:len(data)]
denoised_lstm_flat = denoised_signal_lstm.flatten()[:len(data)]
denoised_enhanced_conv1d_flat = denoised_signal_conv2.flatten()[:len(data)]

# Define the maximum possible pixel value (for PSNR, since your data is normalized to 0-1
MAX_I = 1.0

# --- Define Metric Calculation Functions ---

def calculate_mse(original, reconstructed):
    """Calculates Mean Squared Error."""
    return np.mean(np.square(original - reconstructed))

def calculate_rmse(original, reconstructed):
    """Calculates Root Mean Squared Error."""
    return np.sqrt(calculate_mse(original, reconstructed))

def calculate_snr(original_signal, noisy_signal):
    """Calculates Signal-to-Noise Ratio in dB."""
    # Power of the original (clean) signal
    P_signal = np.mean(original_signal**2)
    # Power of the noise (difference between original and noisy)
    P_noise = np.mean((original_signal - noisy_signal)**2)

    if P_noise == 0:
        return np.inf # Handle the case of perfect reconstruction/no noise
    return 10 * np.log10(P_signal / P_noise)

def calculate_psnr(original_signal, reconstructed_signal, max_val=MAX_I):
    """Calculates Peak Signal-to-Noise Ratio in dB."""
    mse = calculate_mse(original_signal, reconstructed_signal)
    if mse == 0:
        return np.inf # Perfect reconstruction
    return 10 * np.log10((max_val**2) / mse)


# --- Calculate Metrics for Each Scenario ---

print("--- Quantitative Denoising Performance ---")

# 1. Noisy Signal (Baseline)
mse_noisy = calculate_mse(original_signal_flat, noisy_signal_flat)
rmse_noisy = calculate_rmse(original_signal_flat, noisy_signal_flat)
```

```
snr_noisy = calculate_snr(original_signal_flat, noisy_signal_flat)
psnr_noisy = calculate_psnr(original_signal_flat, noisy_signal_flat, MAX_I)

print(f"\nNoisy Signal (Baseline):")
print(f"  MSE: {mse_noisy:.6f}")
print(f"  RMSE: {rmse_noisy:.6f}")
print(f"  SNR: {snr_noisy:.2f} dB")
print(f"  PSNR: {psnr_noisy:.2f} dB")


# 2. Conv1D Autoencoder Denoised Signal
mse_conv1d = calculate_mse(original_signal_flat, denoised_conv1d_flat)
rmse_conv1d = calculate_rmse(original_signal_flat, denoised_conv1d_flat)
snr_conv1d = calculate_snr(original_signal_flat, denoised_conv1d_flat)
psnr_conv1d = calculate_psnr(original_signal_flat, denoised_conv1d_flat, MAX_I)

print(f"\nConv1D Autoencoder Denoised:")
print(f"  MSE: {mse_conv1d:.6f}")
print(f"  RMSE: {rmse_conv1d:.6f}")
print(f"  SNR: {snr_conv1d:.2f} dB")
print(f"  PSNR: {psnr_conv1d:.2f} dB")
print(f"  SNR Improvement: {(snr_conv1d - snr_noisy):.2f} dB")


# 3. LSTM Autoencoder Denoised Signal
mse_lstm = calculate_mse(original_signal_flat, denoised_lstm_flat)
rmse_lstm = calculate_rmse(original_signal_flat, denoised_lstm_flat)
snr_lstm = calculate_snr(original_signal_flat, denoised_lstm_flat)
psnr_lstm = calculate_psnr(original_signal_flat, denoised_lstm_flat, MAX_I)

print(f"\nLSTM Autoencoder Denoised:")
print(f"  MSE: {mse_lstm:.6f}")
print(f"  RMSE: {rmse_lstm:.6f}")
print(f"  SNR: {snr_lstm:.2f} dB")
print(f"  PSNR: {psnr_lstm:.2f} dB")
print(f"  SNR Improvement: {(snr_lstm - snr_noisy):.2f} dB")


# 4. Enhanced Conv1D Autoencoder Denoised Signal
mse_enhanced_conv1d = calculate_mse(original_signal_flat, denoised_enhanced_conv1d_flat)
rmse_enhanced_conv1d = calculate_rmse(original_signal_flat, denoised_enhanced_conv1d_flat
snr_enhanced_conv1d = calculate_snr(original_signal_flat, denoised_enhanced_conv1d_flat)
psnr_enhanced_conv1d = calculate_psnr(original_signal_flat, denoised_enhanced_conv1d_flat

print(f"\nEnhanced Conv1D Autoencoder Denoised:")
print(f"  MSE: {mse_enhanced_conv1d:.6f}")
print(f"  RMSE: {rmse_enhanced_conv1d:.6f}")
print(f"  SNR: {snr_enhanced_conv1d:.2f} dB")
print(f"  PSNR: {psnr_enhanced_conv1d:.2f} dB")
print(f"  SNR Improvement: {(snr_enhanced_conv1d - snr_noisy):.2f} dB")
```

```
--- Quantitative Denoising Performance ---

Noisy Signal (Baseline):
  MSE: 0.082993
```