In [5]:

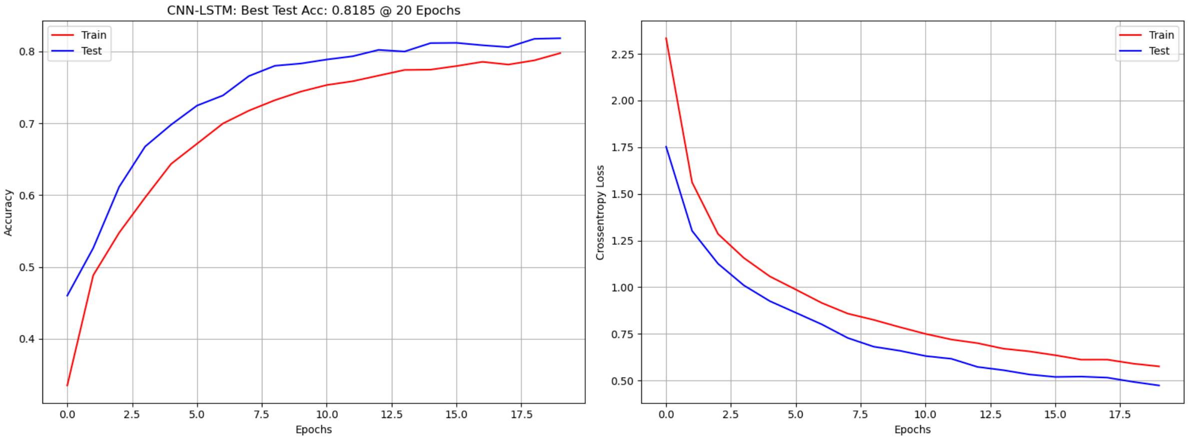
```
import os, glob
import pandas as pd, numpy as np
import torch, torch.nn as nn, torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

# --- DATA LOADER & PREPROCESSING ---
def load_and_merge_datasets(roots):
    merged = []
    for act_folder in sorted(os.listdir(roots[0])):
        if not act_folder.startswith('.'):
            sub_files = glob.glob(os.path.join(roots[0], act_folder, '*.csv'))
            for sub_file in sub_files:
                sub_name = os.path.basename(sub_file)
                triplet = []
                ok = True
                for r in roots:
                    path = os.path.join(r, act_folder, sub_name)
                    if os.path.exists(path):
                        triplet.append(pd.read_csv(path))
                    else:
                        ok = False
                        break
                if ok:
                    dfA, dfB, dfC = triplet
                    dfA = dfA.add_prefix('devicemotion_')
                    dfB = dfB.add_prefix('accelerometer_')
                    dfC = dfC.add_prefix('gyroscope_')
                    df = pd.concat([dfA, dfB, dfC], axis=1)
                    df['act'] = act_folder
                    merged.append(df)
    if not merged:
        raise FileNotFoundError("No data found. Check folder structure!")
    return pd.concat(merged, ignore_index=True)

def windows(df, window_size=100, stride=50):
    Xs, ys = [], []
    cols = [c for c in df.columns if c not in ['act']]
    arr = df[cols].values
    acts = df['act'].values
    for i in range(0, len(df) - window_size, stride):
        Xs.append(arr[i:i+window_size].T)
        ys.append(acts[i+window_size//2])
    return np.array(Xs), np.array(ys)

class MotionSense(Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype=torch.float32)
        self.y = torch.tensor(y, dtype=torch.long)
    def __len__(self):
        return len(self.X)
    def __getitem__(self, i):
        return self.X[i], self.y[i]

class CNN_LSTM(nn.Module):
    def __init__(self, n_features, n_timesteps, n_classes):
        super().__init__()
        self.conv1 = nn.Conv1d(n_features, 64, 3, padding=1)
```

```
        self.bn1 = nn.BatchNorm1d(64)
        self.relu1 = nn.ReLU()
        self.conv2 = nn.Conv1d(64, 128, 3, padding=1)
        self.bn2 = nn.BatchNorm1d(128)
        self.relu2 = nn.ReLU()
        self.lstm = nn.LSTM(128, 64, batch_first=True, num_layers=2, dropout=0.3)
        self.fc1 = nn.Linear(64, n_classes)
        self.dropout = nn.Dropout(0.5)
    def forward(self, x):
        x = self.relu1(self.bn1(self.conv1(x)))
        x = self.relu2(self.bn2(self.conv2(x)))
        x = x.permute(0,2,1)  # (batch, seq, feature)
        _, (h, _) = self.lstm(x)
        h = h[-1]
        h = self.dropout(h)
        return self.fc1(h)

def train_epoch(model, loader, criterion, optimizer, device):
    model.train()
    total_loss, correct, total = 0.0, 0, 0
    for Xb, yb in loader:
        Xb, yb = Xb.to(device), yb.to(device)
        optimizer.zero_grad()
        out = model(Xb)
        loss = criterion(out, yb)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        total_loss += loss.item() * Xb.size(0)
        pred = out.argmax(dim=1)
        correct += (pred == yb).sum().item()
        total += Xb.size(0)
    return total_loss / total, correct / total

def eval_loss(model, loader, criterion, device):
    model.eval()
    total_loss, correct, total = 0.0, 0, 0
    with torch.no_grad():
        for Xb, yb in loader:
            Xb, yb = Xb.to(device), yb.to(device)
            out = model(Xb)
            loss = criterion(out, yb)
            total_loss += loss.item() * Xb.size(0)
            pred = out.argmax(dim=1)
            correct += (pred == yb).sum().item()
            total += Xb.size(0)
    return total_loss / total, correct / total

def plot_history_pt(train_accuracies, test_accuracies, train_losses, test_losses, a
    fig, axs = plt.subplots(1, 2, figsize=(16, 6))
    axs[0].plot(train_accuracies, 'r', label='Train')
    axs[0].plot(test_accuracies, 'b', label='Test')
    axs[0].set_ylabel('Accuracy')
    axs[0].set_xlabel('Epochs')
    axs[0].set_title(add_title + f'Best Test Acc: {np.max(test_accuracies):.4f} @ {
    axs[0].legend(); axs[0].grid()
    axs[1].plot(train_losses, 'r', label='Train')
    axs[1].plot(test_losses, 'b', label='Test')
    axs[1].set_ylabel('Crossentropy Loss')
    axs[1].set_xlabel('Epochs')
    axs[1].legend(); axs[1].grid()
    plt.tight_layout()
    plt.show()
```

```python
if __name__ == "__main__":
    print("Loading and merging data from all sensor folders...")
    roots = ['A_DeviceMotion_data', 'B_Accelerometer_data', 'C_Gyroscope_data']
    df = load_and_merge_datasets(roots)
    df = df.fillna(0)
    print(f"Loaded dataset shape: {df.shape}")

    print("\nSegmenting into windowed samples for deep learning...")
    X, y_str = windows(df, window_size=100, stride=50)
    print(f"Feature array shape: {X.shape}, Label array shape: {y_str.shape}")

    print("\nNormalizing and encoding the data...")
    nsamples, nfeat, nsteps = X.shape
    X_reshaped = X.transpose(0,2,1).reshape(-1, nfeat)
    X_reshaped = np.nan_to_num(X_reshaped, nan=0.0, posinf=0.0, neginf=0.0)
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_reshaped)
    X = X_scaled.reshape(nsamples, nsteps, nfeat).transpose(0,2,1)
    X = np.nan_to_num(X, nan=0.0, posinf=0.0, neginf=0.0)

    le = LabelEncoder()
    y_enc = le.fit_transform(y_str)
    print(f"Classes found: {list(le.classes_)}")

    X_tr, X_te, y_tr, y_te = train_test_split(X, y_enc, test_size=0.2, random_state
    train_loader = DataLoader(MotionSense(X_tr, y_tr), batch_size=128, shuffle=True
    test_loader = DataLoader(MotionSense(X_te, y_te), batch_size=128, shuffle=False

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    n_features = X_tr.shape[1]
    n_timesteps = X_tr.shape[2]
    n_classes = len(le.classes_)
    print("\nBuilding CNN-LSTM model...")
    model = CNN_LSTM(n_features, n_timesteps, n_classes).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=1e-4)

    print("\nTraining model. For each epoch, loss and accuracy are reported for ana
    epochs = 20
    train_losses, train_accuracies, test_accuracies, test_losses = [], [], [], []
    for ep in range(1, epochs+1):
        loss, train_acc = train_epoch(model, train_loader, criterion, optimizer, de
        test_loss, test_acc = eval_loss(model, test_loader, criterion, device)
        train_losses.append(loss)
        train_accuracies.append(train_acc)
        test_accuracies.append(test_acc)
        test_losses.append(test_loss)
        print(f"Epoch {ep:02d} — Train Loss: {loss:.4f} — Train Acc: {train_acc:.4f

    print("\nPlotting training/validation accuracy and loss for each epoch. Use thi
    plot_history_pt(train_accuracies, test_accuracies, train_losses, test_losses, a

    print("\nEvaluating model on the test set for detailed classification performar
    model.eval()
    all_labels, all_preds = [], []
    with torch.no_grad():
        for Xb, yb in test_loader:
            Xb = Xb.to(device)
            out = model(Xb)
            preds = out.argmax(dim=1).cpu().numpy()
            all_preds.extend(preds)
            all_labels.extend(yb.numpy())
    all_labels = np.array(all_labels)
    all_preds = np.array(all_preds)
```

```
print("\nClassification report below shows precision, recall, and F1-score for
print("These metrics help you understand per-activity performance, not just ove
print(classification_report(all_labels, all_preds, target_names=le.classes_))

print("The following confusion matrix heatmap visualizes where the model is mos
cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(12, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_, yti
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.tight_layout()
plt.show()

print("Below are a few sample predictions from the test set (True Label vs Pred
print("This qualitative check lets you see actual predictions and errors.")
for i in range(10):
    true_label = le.classes_[all_labels[i]]
    pred_label = le.classes_[all_preds[i]]
    print(f"True: {true_label}, Pred: {pred_label}")
```

```
Loading and merging data from all sensor folders...
Loaded dataset shape: (1433825, 22)

Segmenting into windowed samples for deep learning...
Feature array shape: (28675, 21, 100), Label array shape: (28675,)

Normalizing and encoding the data...
Classes found: ['dws_1', 'dws_11', 'dws_2', 'jog_16', 'jog_9', 'sit_13', 'sit_5',
'std_14', 'std_6', 'ups_12', 'ups_3', 'ups_4', 'wlk_15', 'wlk_7', 'wlk_8']

Building CNN-LSTM model...

Training model. For each epoch, loss and accuracy are reported for analysis.
Epoch 01 — Train Loss: 2.3337 — Train Acc: 0.3349 — Test Loss: 1.7526 — Test Acc:
0.4600
Epoch 02 — Train Loss: 1.5619 — Train Acc: 0.4883 — Test Loss: 1.3019 — Test Acc:
0.5262
Epoch 03 — Train Loss: 1.2852 — Train Acc: 0.5476 — Test Loss: 1.1258 — Test Acc:
0.6115
Epoch 04 — Train Loss: 1.1561 — Train Acc: 0.5966 — Test Loss: 1.0093 — Test Acc:
0.6677
Epoch 05 — Train Loss: 1.0575 — Train Acc: 0.6434 — Test Loss: 0.9250 — Test Acc:
0.6980
Epoch 06 — Train Loss: 0.9871 — Train Acc: 0.6716 — Test Loss: 0.8631 — Test Acc:
0.7247
Epoch 07 — Train Loss: 0.9154 — Train Acc: 0.6997 — Test Loss: 0.8007 — Test Acc:
0.7388
Epoch 08 — Train Loss: 0.8585 — Train Acc: 0.7177 — Test Loss: 0.7278 — Test Acc:
0.7658
Epoch 09 — Train Loss: 0.8248 — Train Acc: 0.7322 — Test Loss: 0.6812 — Test Acc:
0.7801
Epoch 10 — Train Loss: 0.7861 — Train Acc: 0.7442 — Test Loss: 0.6595 — Test Acc:
0.7833
Epoch 11 — Train Loss: 0.7497 — Train Acc: 0.7533 — Test Loss: 0.6312 — Test Acc:
0.7888
Epoch 12 — Train Loss: 0.7194 — Train Acc: 0.7586 — Test Loss: 0.6161 — Test Acc:
0.7934
Epoch 13 — Train Loss: 0.7000 — Train Acc: 0.7665 — Test Loss: 0.5727 — Test Acc:
0.8023
Epoch 14 — Train Loss: 0.6705 — Train Acc: 0.7743 — Test Loss: 0.5552 — Test Acc:
0.8000
Epoch 15 — Train Loss: 0.6559 — Train Acc: 0.7747 — Test Loss: 0.5322 — Test Acc:
0.8117
Epoch 16 — Train Loss: 0.6353 — Train Acc: 0.7798 — Test Loss: 0.5189 — Test Acc:
0.8120
Epoch 17 — Train Loss: 0.6116 — Train Acc: 0.7856 — Test Loss: 0.5208 — Test Acc:
0.8087
Epoch 18 — Train Loss: 0.6118 — Train Acc: 0.7818 — Test Loss: 0.5154 — Test Acc:
0.8061
Epoch 19 — Train Loss: 0.5904 — Train Acc: 0.7877 — Test Loss: 0.4927 — Test Acc:
0.8176
Epoch 20 — Train Loss: 0.5755 — Train Acc: 0.7977 — Test Loss: 0.4734 — Test Acc:
0.8185

Plotting training/validation accuracy and loss for each epoch. Use this to visuall
y assess overfitting or convergence.
```



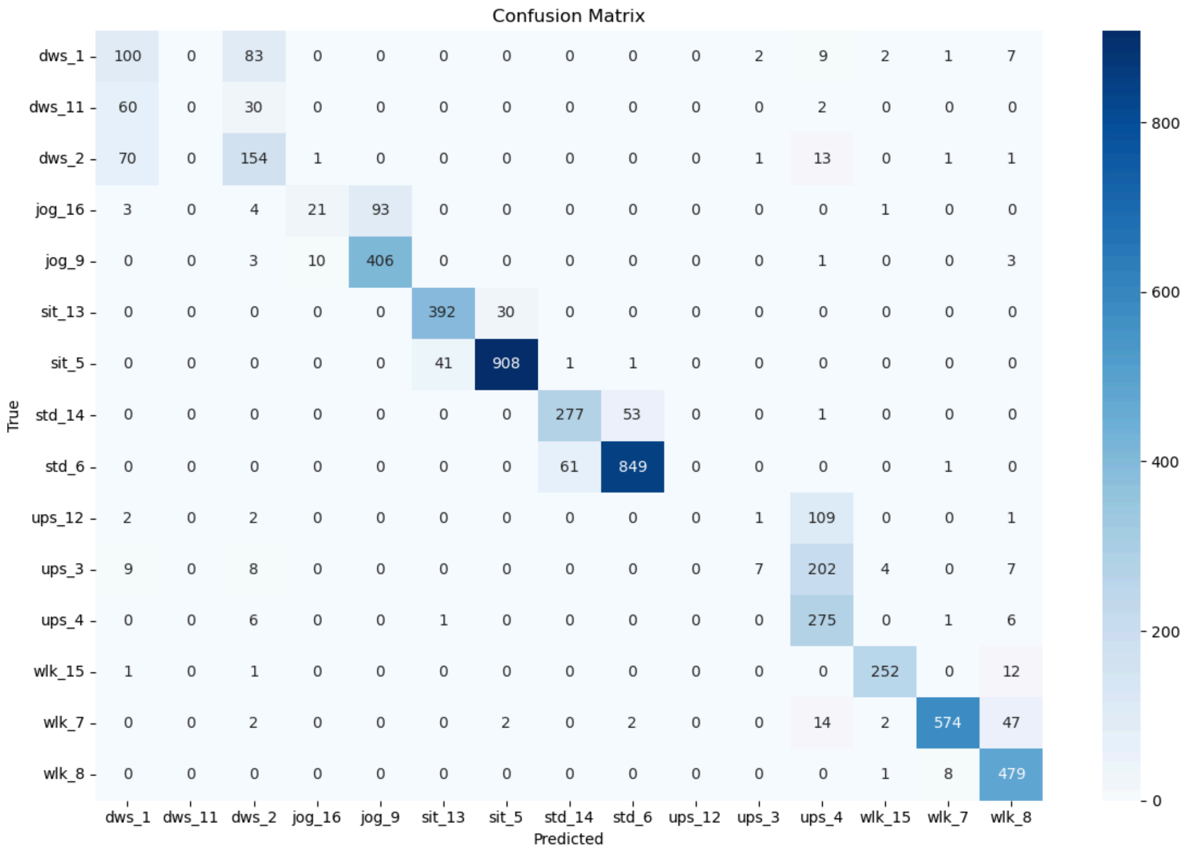
Evaluating model on the test set for detailed classification performance...

Classification report below shows precision, recall, and F1-score for each class. These metrics help you understand per-activity performance, not just overall accuracy.

```
              precision    recall  f1-score   support

       dws_1       0.41      0.49      0.45       204
      dws_11       0.00      0.00      0.00        92
       dws_2       0.53      0.64      0.58       241
      jog_16       0.66      0.17      0.27       122
       jog_9       0.81      0.96      0.88       423
      sit_13       0.90      0.93      0.92       422
       sit_5       0.97      0.95      0.96       951
      std_14       0.82      0.84      0.83       331
       std_6       0.94      0.93      0.94       911
      ups_12       0.00      0.00      0.00       115
       ups_3       0.64      0.03      0.06       237
       ups_4       0.44      0.95      0.60       289
      wlk_15       0.96      0.95      0.95       266
       wlk_7       0.98      0.89      0.93       643
       wlk_8       0.85      0.98      0.91       488

    accuracy                           0.82      5735
   macro avg       0.66      0.65      0.62      5735
weighted avg       0.81      0.82      0.79      5735
```

The following confusion matrix heatmap visualizes where the model is most accurate and which classes are being confused with others.

```
C:\Users\Sachin\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Sachin\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Sachin\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```




```
Below are a few sample predictions from the test set (True Label vs Predicted Labe
l).
This qualitative check lets you see actual predictions and errors.
True: ups_4, Pred: ups_4
True: wlk_8, Pred: wlk_8
True: jog_16, Pred: jog_9
True: sit_5, Pred: sit_5
True: std_6, Pred: std_6
True: std_6, Pred: std_6
True: wlk_15, Pred: wlk_15
True: ups_3, Pred: ups_4
True: sit_5, Pred: sit_5
True: wlk_8, Pred: wlk_8
```

In [ ]:

In [ ]: