Mukundi Mangena and Ntsako Phiri

Model preprocessed using windowing

Download the dataset and prepare it for preprocessing.

In [5]:
```
from scipy import fft
import numpy as np
import pandas as pd
from pandas import DataFrame
import pandas as pd

def get_ds_infos():
    """
    Read the file includes data subject information.

    Data Columns:
    0: code [1-24]
    1: weight [kg]
    2: height [cm]
    3: age [years]
    4: gender [0:Female, 1:Male]

    Returns:
        A pandas DataFrame that contains inforamtion about data subjects' attrib
    """

    dss = pd.read_csv("data_subjects_info.csv")
    print("[INFO] -- Data subjects' information is imported.")

    return dss

def set_data_types(data_types=["userAcceleration"]):
    """
    Select the sensors and the mode to shape the final dataset.

    Args:
        data_types: A list of sensor data type from this list: [attitude, gravit

    Returns:
        It returns a list of columns to use for creating time-series from files.
    """
    dt_list = []
    for t in data_types:
        if t != "attitude":
            dt_list.append([t+".x",t+".y",t+".z"])
        else:
            dt_list.append([t+".roll", t+".pitch", t+".yaw"])

    return dt_list


def creat_time_series(dt_list, act_labels, trial_codes, mode="mag", labeled=True
    """
    Args:
```

```
        dt_list: A list of columns that shows the type of data we want.
        act_labels: list of activites
        trial_codes: list of trials
        mode: It can be "raw" which means you want raw data
        for every dimention of each data type,
        [attitude(roll, pitch, yaw); gravity(x, y, z); rotationRate(x, y, z); us
        or it can be "mag" which means you only want the magnitude for each data
        labeled: True, if we want a labeld dataset. False, if we only want senso

    Returns:
        It returns a time-series of sensor data.

    """
    num_data_cols = len(dt_list) if mode == "mag" else len(dt_list*3)

    if labeled:
        dataset = np.zeros((0,num_data_cols+7)) # "7" --> [act, code, weight, he
    else:
        dataset = np.zeros((0,num_data_cols))

    ds_list = get_ds_infos()

    print("[INFO] -- Creating Time-Series")
    for sub_id in ds_list["code"]:
        for act_id, act in enumerate(act_labels):
            for trial in trial_codes[act_id]:
                fname = 'A_DeviceMotion_data/'+act+'_'+str(trial)+'/sub_'+str(in
                raw_data = pd.read_csv(fname)
                raw_data = raw_data.drop(['Unnamed: 0'], axis=1)
                vals = np.zeros((len(raw_data), num_data_cols))
                for x_id, axes in enumerate(dt_list):
                    if mode == "mag":
                        vals[:,x_id] = (raw_data[axes]**2).sum(axis=1)**0.5
                    else:
                        vals[:,x_id*3:(x_id+1)*3] = raw_data[axes].values
                    vals = vals[:,:num_data_cols]
                if labeled:
                    lbls = np.array([[act_id,
                            sub_id-1,
                            ds_list["weight"][sub_id-1],
                            ds_list["height"][sub_id-1],
                            ds_list["age"][sub_id-1],
                            ds_list["gender"][sub_id-1],
                            trial
                           ]]*len(raw_data))
                    vals = np.concatenate((vals, lbls), axis=1)
                dataset = np.append(dataset,vals, axis=0)
    cols = []
    for axes in dt_list:
        if mode == "raw":
            cols += axes
        else:
            cols += [str(axes[0][:-2])]

    if labeled:
        cols += ["act", "id", "weight", "height", "age", "gender", "trial"]

    dataset = pd.DataFrame(data=dataset, columns=cols)
    return dataset
#_____________________________________
```

```
ACT_LABELS = ["dws","ups", "wlk", "jog", "std", "sit"]
TRIAL_CODES = {
    ACT_LABELS[0]:[1,2,11],
    ACT_LABELS[1]:[3,4,12],
    ACT_LABELS[2]:[7,8,15],
    ACT_LABELS[3]:[9,16],
    ACT_LABELS[4]:[6,14],
    ACT_LABELS[5]:[5,13]
}

## Here we set parameter to build labeld time-series from dataset of "(A)DeviceM
## attitude(roll, pitch, yaw); gravity(x, y, z); rotationRate(x, y, z); userAcce
sdt = ["attitude", "userAcceleration","rotationRate"]
print("[INFO] -- Selected sensor data types: "+str(sdt))
act_labels = ACT_LABELS [0:6] # ["dws","ups", "wlk", "jog", "std"]
print("[INFO] -- Selected activites: "+str(act_labels))
trial_codes = [TRIAL_CODES[act] for act in act_labels]
dt_list = set_data_types(sdt)
dataset = creat_time_series(dt_list, act_labels, trial_codes, mode="raw", labele
print("[INFO] -- Shape of time-Series dataset:"+str(dataset.shape))
dataset.head()
```

```
[INFO] -- Selected sensor data types: ['attitude', 'userAcceleration', 'rotationR
ate']
[INFO] -- Selected activites: ['dws', 'ups', 'wlk', 'jog', 'std', 'sit']
[INFO] -- Data subjects' information is imported.
[INFO] -- Creating Time-Series
[INFO] -- Shape of time-Series dataset:(1412865, 16)
```

Out[5]:

	attitude.roll	attitude.pitch	attitude.yaw	userAcceleration.x	userAcceleration.y	user
0	1.528132	-0.733896	0.696372	0.294894	-0.184493	
1	1.527992	-0.716987	0.677762	0.219405	0.035846	
2	1.527765	-0.706999	0.670951	0.010714	0.134701	
3	1.516768	-0.704678	0.675735	-0.008389	0.136788	
4	1.493941	-0.703918	0.672994	0.199441	0.353996	

drop the redudant tables

In [6]:
```
dataset = dataset.drop(columns=["id", "weight", "height", "age", "gender", "tria
dataset.head()
```

Out[6]:

	attitude.roll	attitude.pitch	attitude.yaw	userAcceleration.x	userAcceleration.y	user
0	1.528132	-0.733896	0.696372	0.294894	-0.184493	
1	1.527992	-0.716987	0.677762	0.219405	0.035846	
2	1.527765	-0.706999	0.670951	0.010714	0.134701	
3	1.516768	-0.704678	0.675735	-0.008389	0.136788	
4	1.493941	-0.703918	0.672994	0.199441	0.353996	

Note that this database has all the data in the dataset it will require to be trimmed to alow for more effective models.

Data Pre-Processing Steps

Setting up the input and output data

In [7]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical

x  = dataset[dataset.columns[:-1]].values
y = dataset["act"].values
print(x[:10])
print(y[:10])
print("[INFO] -- Dataset is split into input and output data.")
```

```
[[ 1.528132 -0.733896  0.696372  0.294894 -0.184493  0.377542  0.316738
   0.77818   1.082764]
 [ 1.527992 -0.716987  0.677762  0.219405  0.035846  0.114866  0.842032
   0.424446  0.643574]
 [ 1.527765 -0.706999  0.670951  0.010714  0.134701 -0.167808 -0.138143
  -0.040741  0.343563]
 [ 1.516768 -0.704678  0.675735 -0.008389  0.136788  0.094958 -0.025005
  -1.048717  0.03586 ]
 [ 1.493941 -0.703918  0.672994  0.199441  0.353996 -0.044299  0.114253
  -0.91289   0.047341]
 [ 1.476302 -0.700807  0.669443  0.168241  0.145906  0.012455  0.187742
  -0.763656  0.226057]
 [ 1.455153 -0.694408  0.662593  0.079382 -0.026344 -0.19559   0.343096
  -0.80382   0.278468]
 [ 1.441702 -0.69071   0.656459  0.06936   0.072678 -0.10292   0.176202
  -0.172756  0.056415]
 [ 1.44344  -0.691905  0.651196  0.072889  0.079921 -0.075323  0.274786
   0.446585 -0.132766]
 [ 1.443071 -0.693039  0.638198  0.098347 -0.017021 -0.19731   0.633672
   0.316372 -0.115137]]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[INFO] -- Dataset is split into input and output data.
```

Preprocessed by FFT Based windowing

Since the data is time series data when implementing a CNN or RNN

In [ ]:
```
from scipy.fft import fft
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
import numpy as np

def fft_features(data, labels, window_size=50, step=25):
    X_fft, y_fft = [], []

    for start in range(0, len(data) - window_size, step):
        end = start + window_size
        segment = data[start:end]
        label = labels[end - 1]  # or use majority vote or center label

        # Apply FFT along each feature column (axis=0)
        fft_segment = np.abs(fft(segment, axis=0))[:window_size // 2]
        fft_flat = fft_segment.flatten()  # Flatten into 1D feature vector

        X_fft.append(fft_flat)
        y_fft.append(label)

    return np.array(X_fft), np.array(y_fft)
# Extract FFT features
x_fft, y_fft = fft_features(x, y, window_size=50, step=25)
print("[INFO] -- FFT features are extracted from time-series data.")

# Normalize FFT features
scaler = StandardScaler()
x_fft = scaler.fit_transform(x_fft)
print("[INFO] -- FFT features are normalized.")

# Train/Test Split
x_train, x_test, y_train_int, y_test_int = train_test_split(
    x_fft, y_fft, test_size=0.2, random_state=42, stratify=y_fft
)

# One-hot encode the labels
y_train = to_categorical(y_train_int)
y_test = to_categorical(y_test_int)

# Reshape for Conv1D (samples, timesteps, features)
x_train = x_train.reshape((x_train.shape[0], x_train.shape[1], 1))
x_test = x_test.reshape((x_test.shape[0], x_test.shape[1], 1))

print("[INFO] -- Data is preprocessed and ready for CNN.")
```

```
[INFO] -- FFT features are extracted from time-series data.
[INFO] -- FFT features are normalized.
[INFO] -- Data is preprocessed and ready for CNN.
```

Building the Model

In [10]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropou

model = Sequential()
# Input shape: (timesteps, features) = (x_fft.shape[1], 1)
model.add(Conv1D(64, kernel_size=3, activation='relu', input_shape=(x_train.shap
model.add(MaxPooling1D(pool_size=2))
```

```
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dense(y_train.shape[1], activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accur
```

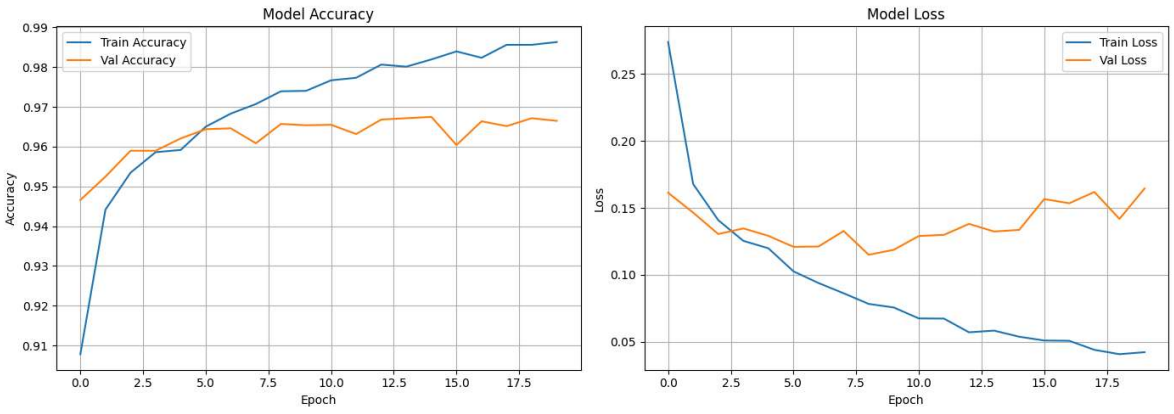
```
c:\Users\btmuk\OneDrive\Desktop\2025\DSP\.venv\Lib\site-packages\keras\src\layers
\convolutional\base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input
_dim` argument to a layer. When using Sequential models, prefer using an `Input(s
hape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Training The Model

In [11]:
```
trained = model.fit(x_train, y_train, epochs=20, batch_size=32, validation_split
```

```
Epoch 1/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 22s 17ms/step - accuracy: 0.8583 - loss: 0.4046 -
val_accuracy: 0.9466 - val_loss: 0.1613
Epoch 2/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 17s 15ms/step - accuracy: 0.9430 - loss: 0.1710 -
val_accuracy: 0.9524 - val_loss: 0.1465
Epoch 3/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 18s 16ms/step - accuracy: 0.9535 - loss: 0.1372 -
val_accuracy: 0.9590 - val_loss: 0.1305
Epoch 4/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 17s 15ms/step - accuracy: 0.9594 - loss: 0.1219 -
val_accuracy: 0.9590 - val_loss: 0.1347
Epoch 5/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 20s 18ms/step - accuracy: 0.9594 - loss: 0.1193 -
val_accuracy: 0.9621 - val_loss: 0.1291
Epoch 6/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 23s 20ms/step - accuracy: 0.9632 - loss: 0.1064 -
val_accuracy: 0.9644 - val_loss: 0.1209
Epoch 7/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - accuracy: 0.9689 - loss: 0.0916 -
val_accuracy: 0.9646 - val_loss: 0.1212
Epoch 8/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 17s 15ms/step - accuracy: 0.9704 - loss: 0.0853 -
val_accuracy: 0.9608 - val_loss: 0.1327
Epoch 9/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - accuracy: 0.9764 - loss: 0.0710 -
val_accuracy: 0.9657 - val_loss: 0.1150
Epoch 10/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 18s 16ms/step - accuracy: 0.9752 - loss: 0.0747 -
val_accuracy: 0.9654 - val_loss: 0.1187
Epoch 11/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 18s 16ms/step - accuracy: 0.9773 - loss: 0.0636 -
val_accuracy: 0.9655 - val_loss: 0.1290
Epoch 12/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 16s 14ms/step - accuracy: 0.9794 - loss: 0.0627 -
val_accuracy: 0.9632 - val_loss: 0.1299
Epoch 13/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 18s 16ms/step - accuracy: 0.9819 - loss: 0.0541 -
val_accuracy: 0.9668 - val_loss: 0.1381
Epoch 14/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 18s 16ms/step - accuracy: 0.9817 - loss: 0.0522 -
val_accuracy: 0.9672 - val_loss: 0.1324
Epoch 15/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 16s 14ms/step - accuracy: 0.9821 - loss: 0.0536 -
val_accuracy: 0.9675 - val_loss: 0.1335
Epoch 16/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 18s 16ms/step - accuracy: 0.9843 - loss: 0.0513 -
val_accuracy: 0.9604 - val_loss: 0.1566
Epoch 17/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 17s 15ms/step - accuracy: 0.9828 - loss: 0.0480 -
val_accuracy: 0.9664 - val_loss: 0.1535
Epoch 18/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 21s 18ms/step - accuracy: 0.9869 - loss: 0.0413 -
val_accuracy: 0.9652 - val_loss: 0.1619
Epoch 19/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 19s 17ms/step - accuracy: 0.9850 - loss: 0.0408 -
val_accuracy: 0.9672 - val_loss: 0.1418
Epoch 20/20
1131/1131 ━━━━━━━━━━━━━━━━━━━━ 17s 15ms/step - accuracy: 0.9865 - loss: 0.0394 -
val_accuracy: 0.9665 - val_loss: 0.1645
```

In [12]:
```
import matplotlib.pyplot as plt

def plot_training_history(history):
    """
    Plots training and validation accuracy and loss curves.
    `history` is the object returned by model.fit().
    """
    plt.figure(figsize=(14, 5))

    # Accuracy plot
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Val Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)

    # Loss plot
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()

plot_training_history(trained)
```



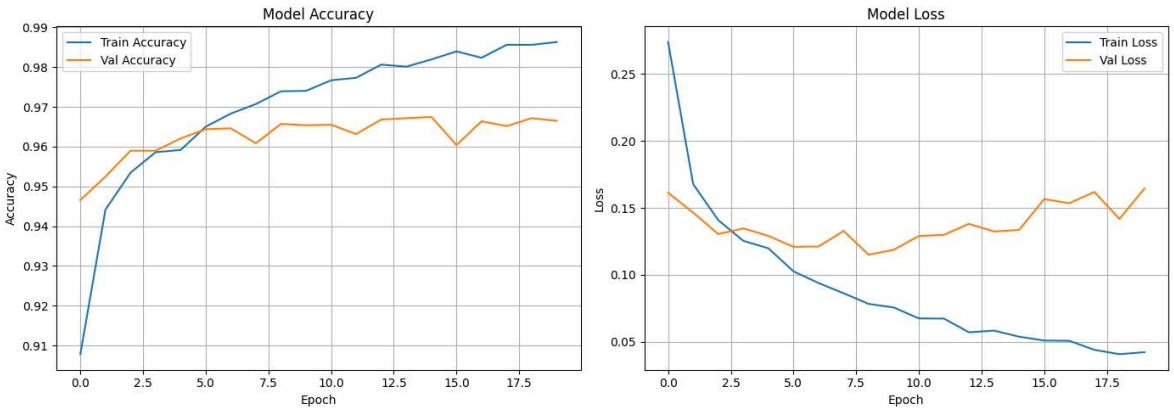
Testing the Model with The Test Data

In [13]:
```
# Evaluate the trained model
loss, acc = model.evaluate(x_test, y_test, verbose=0)
print(f"Test Loss: {loss:.4f}, Test Accuracy: {acc:.4f}")
```

```
Test Loss: 0.1449, Test Accuracy: 0.9626
```

Plot Training and Validation Accuracy/Loss

In [14]:
```
import matplotlib.pyplot as plt

def plot_training_history(history):
    """
    Plots training and validation accuracy and loss curves.
    `history` is the object returned by model.fit().
    """
    plt.figure(figsize=(14, 5))

    # Accuracy plot
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Val Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)

    # Loss plot
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()


plot_training_history(trained)
```

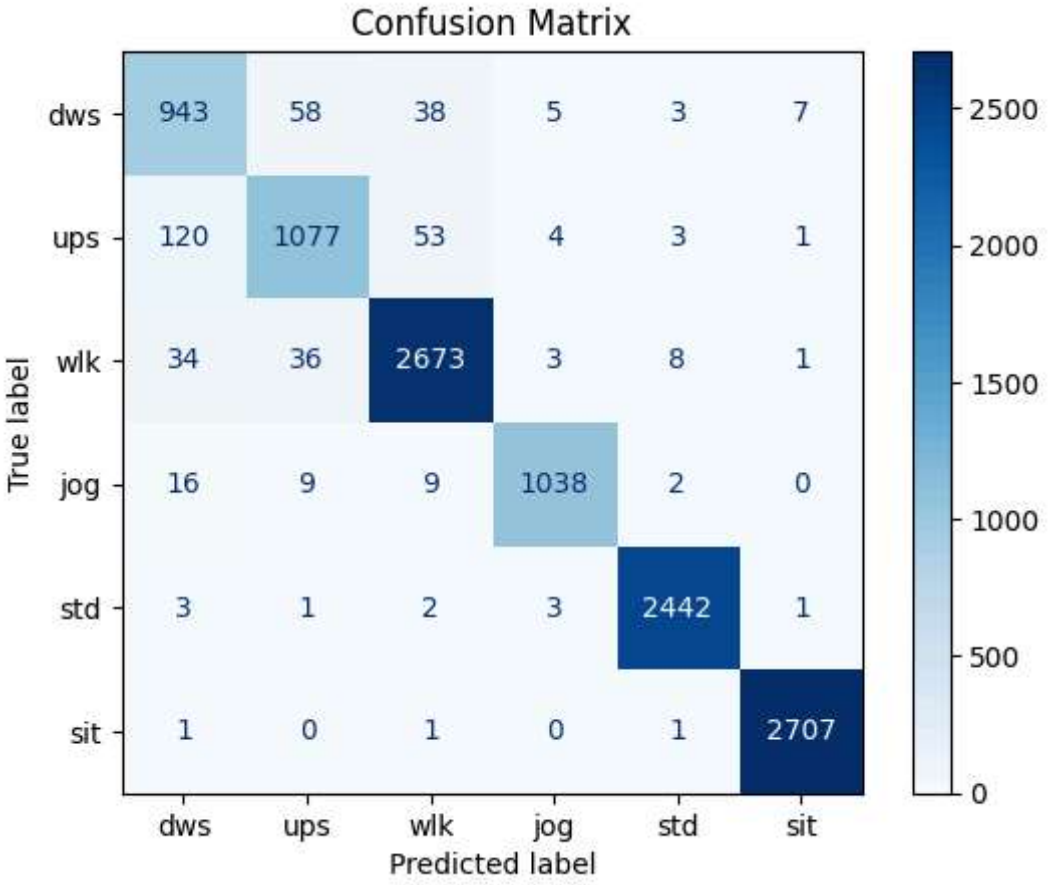


Confusion Matrix (with heatmap)

In [15]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import numpy as np

def plot_confusion_matrix(model, X_test, y_test, class_names, use_softmax=True):
    """
    Plots a confusion matrix for a trained Keras model.
    Assumes y_test is either integer labels (sparse) or one-hot.
    """
```

```
    # If labels are one-hot encoded, convert to integer
    if y_test.ndim == 2:
        y_true = np.argmax(y_test, axis=1)
    else:
        y_true = y_test

    # Predict class labels
    y_pred = model.predict(X_test)
    y_pred = np.argmax(y_pred, axis=1) if use_softmax else y_pred

    # Confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_name
    disp.plot(cmap=plt.cm.Blues)
    plt.title("Confusion Matrix")
    plt.grid(False)
    plt.show()
plot_confusion_matrix(model, x_test, y_test, class_names=ACT_LABELS)
```

```
354/354 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```



Classification Report

In [16]:

```
from sklearn.metrics import classification_report

def print_classification_report(model, X_test, y_test, class_names, use_softmax=
    """
    Prints precision, recall, F1-score for each class.
    """
    # Handle one-hot labels
    if y_test.ndim == 2:
        y_true = np.argmax(y_test, axis=1)
```

```
    else:
        y_true = y_test

    # Get predictions
    y_pred = model.predict(X_test)
    y_pred = np.argmax(y_pred, axis=1) if use_softmax else y_pred

    print("Classification Report:")
    print(classification_report(y_true, y_pred, target_names=class_names))
print_classification_report(model, x_test, y_test, class_names=ACT_LABELS)
```

```
354/354 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
Classification Report:
              precision    recall  f1-score   support

         dws       0.84      0.89      0.87      1054
         ups       0.91      0.86      0.88      1258
         wlk       0.96      0.97      0.97      2755
         jog       0.99      0.97      0.98      1074
         std       0.99      1.00      0.99      2452
         sit       1.00      1.00      1.00      2710

    accuracy                           0.96     11303
   macro avg       0.95      0.95      0.95     11303
weighted avg       0.96      0.96      0.96     11303
```