In [4]:
```
# ==========================================
# 1. IMPORT LIBRARIES
# ==========================================
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix
)

# Classical ML models
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

import tensorflow as tf
from tensorflow.keras import layers, models, callbacks
```

In [6]:
```
# ==========================================
# 2. LOAD CSV AND FILTER APPLE ROWS
# ==========================================

csv_path = "/Users/fm/Desktop/AI/plant_disease_multimodal_dataset.csv"
df = pd.read_csv(csv_path)

print("Columns in CSV:")
print(df.columns)

# Filter only Apple rows using 'label'
apple_mask = df["Label"].astype(str).str.startswith("Apple___")
df_apple = df[apple_mask].copy()

print("\nOriginal shape:", df.shape)
print("Apple-only shape:", df_apple.shape)

print("\nUnique raw labels in Apple subset (Label):")
print(df_apple["Label"].unique())

print("\nMapped Label value counts:")
print(df_apple["Mapped Label"].value_counts())
```

```
Columns in CSV:
Index(['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall',
       'Mapped Label', 'Image Path', 'Label'],
      dtype='object')

Original shape: (220000, 10)
Apple-only shape: (40000, 10)

Unique raw labels in Apple subset (Label):
['Apple___Apple_scab' 'Apple___Black_rot' 'Apple___Cedar_apple_rust'
 'Apple___healthy']

Mapped Label value counts:
Mapped Label
Apple___Apple_scab         10000
Apple___Black_rot          10000
Apple___Cedar_apple_rust   10000
Apple___healthy            10000
Name: count, dtype: int64
```

In [8]:
```
# ==========================================
# 3. SELECT STRUCTURED FEATURES + LABEL
# ==========================================

feature_cols = ["N", "P", "K", "temperature", "humidity", "ph", "rainfall"]
label_col = "Mapped Label"

X = df_apple[feature_cols].values  # structured features
y = df_apple[label_col].values     # class label

print("X shape:", X.shape)
print("y shape:", y.shape)
print("Unique labels:", np.unique(y))
```

```
X shape: (40000, 7)
y shape: (40000,)
Unique labels: ['Apple___Apple_scab' 'Apple___Black_rot' 'Apple___Cedar_apple_rust'
 'Apple___healthy']
```

In [10]:
```
# ==========================================
# 4. LABEL ENCODING + TRAIN/VAL/TEST SPLIT
# ==========================================

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

num_classes = len(label_encoder.classes_)
print("Classes:", label_encoder.classes_)

# 70% train, 15% val, 15% test
X_train_temp, X_test, y_train_temp, y_test = train_test_split(
    X, y_encoded,
    test_size=0.15,
    random_state=42,
    stratify=y_encoded
)

val_size = 0.15 / 0.85  # so final val is 15% of total

X_train, X_val, y_train, y_val = train_test_split(
    X_train_temp, y_train_temp,
    test_size=val_size,
    random_state=42,
    stratify=y_train_temp
)

print("\nTrain:", X_train.shape, len(y_train))
print("Val:   ", X_val.shape,   len(y_val))
print("Test:  ", X_test.shape,  len(y_test))
```

```
Classes: ['Apple___Apple_scab' 'Apple___Black_rot' 'Apple___Cedar_apple_rust'
 'Apple___healthy']

Train: (28000, 7) 28000
Val:   (6000, 7) 6000
Test:  (6000, 7) 6000
```

In [12]:
```
# ==========================================
# 5. STANDARD SCALER (FIT ONLY ON TRAIN)
# ==========================================

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled   = scaler.transform(X_val)
X_test_scaled  = scaler.transform(X_test)

print("Scaled shapes:")
print("Train:", X_train_scaled.shape)
print("Val:  ", X_val_scaled.shape)
print("Test: ", X_test_scaled.shape)

Scaled shapes:
Train: (28000, 7)
Val:   (6000, 7)
Test:  (6000, 7)
```

In [14]:
```
# ==========================================
# 6. MLP MODEL (STRUCTURED-ONLY)
# ==========================================

def build_mlp_model(input_dim, num_classes):
    inputs = tf.keras.Input(shape=(input_dim,))
    x = layers.Dense(64, activation="relu")(inputs)
    x = layers.Dropout(0.3)(x)
    x = layers.Dense(32, activation="relu")(x)
    x = layers.Dropout(0.2)(x)
    outputs = layers.Dense(num_classes, activation="softmax")(x)

    model = models.Model(inputs, outputs, name="MLP_Structured_Only")
    model.compile(
        optimizer=tf.keras.optimizers.Adam(1e-3),
        loss="sparse_categorical_crossentropy",
        metrics=["accuracy"]
    )
    return model

input_dim = X_train_scaled.shape[1]
mlp_model = build_mlp_model(input_dim, num_classes)
mlp_model.summary()
```

Model: "MLP_Structured_Only"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 7)	0
dense (Dense)	(None, 64)	512
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2,080
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 4)	132

Total params: 2,724 (10.64 KB)
Trainable params: 2,724 (10.64 KB)
Non-trainable params: 0 (0.00 B)

In [23]:
```
# ==========================================
# 7. TRAIN MLP
# ==========================================

early_stop = callbacks.EarlyStopping(
    monitor="val_loss",
    patience=10,
    restore_best_weights=True
)

history_mlp = mlp_model.fit(
    X_train_scaled, y_train,
    validation_data=(X_val_scaled, y_val),
    epochs=100,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 1.0000 - loss: 1.2013e-06 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 2/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 1.0000 - loss: 2.2634e-06 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 3/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 1.0000 - loss: 2.4402e-07 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 4/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 1.0000 - loss: 2.6095e-07 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 5/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 1.0000 - loss: 3.4836e-07 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 6/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 1.0000 - loss: 4.8586e-07 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 7/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 1.0000 - loss: 2.2674e-07 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 8/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 1.0000 - loss: 9.8372e-08 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 9/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 1.0000 - loss: 5.2339e-06 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 10/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 1.0000 - loss: 3.3196e-07 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 11/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 1.0000 - loss: 3.7999e-06 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
```

In [25]:
```
# ==========================================
# 8. EVALUATE MLP (FULL METRICS)
# ==========================================

from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# Predictions
y_proba_mlp = mlp_model.predict(X_test_scaled)
y_pred_mlp  = y_proba_mlp.argmax(axis=1)

# Basic metrics
acc_mlp = accuracy_score(y_test, y_pred_mlp)
prec_mlp = precision_score(y_test, y_pred_mlp, average='macro', zero_division=0)
rec_mlp  = recall_score(y_test, y_pred_mlp, average='macro', zero_division=0)
f1_mlp   = f1_score(y_test, y_pred_mlp, average='macro', zero_division=0)

# Weighted metrics (for reporting)
prec_w = precision_score(y_test, y_pred_mlp, average='weighted', zero_division=0)
rec_w  = recall_score(y_test, y_pred_mlp, average='weighted', zero_division=0)
f1_w   = f1_score(y_test, y_pred_mlp, average='weighted', zero_division=0)

# Print results
print("\n============== MLP METRICS ==============")
print("Accuracy:        ", acc_mlp)
print("Precision:       ", prec_mlp)
print("Recall:          ", rec_mlp)
print("F1-score:        ", f1_mlp)

print("\nWeighted Precision:", prec_w)
print("Weighted Recall:   ", rec_w)
print("Weighted F1-score: ", f1_w)

print("\n============== CLASSIFICATION REPORT ==============")
print(classification_report(y_test, y_pred_mlp, target_names=label_encoder.classes_))

print("\n============== CONFUSION MATRIX ==============")
print(confusion_matrix(y_test, y_pred_mlp))
```

```
188/188 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step

============== MLP METRICS ==============
Accuracy:        1.0
Precision:       1.0
Recall:          1.0
F1-score:        1.0

Weighted Precision: 1.0
Weighted Recall:    1.0
Weighted F1-score:  1.0

============== CLASSIFICATION REPORT ==============
                                     precision    recall  f1-score   support

                Apple___Apple_scab       1.00      1.00      1.00      1500
                Apple___Black_rot        1.00      1.00      1.00      1500
           Apple___Cedar_apple_rust      1.00      1.00      1.00      1500
                   Apple___healthy       1.00      1.00      1.00      1500

                          accuracy                           1.00      6000
                         macro avg       1.00      1.00      1.00      6000
                      weighted avg       1.00      1.00      1.00      6000

============== CONFUSION MATRIX ==============
[[1500    0    0    0]
 [   0 1500    0    0]
 [   0    0 1500    0]
 [   0    0    0 1500]]
```

In [1]:

In [1]:

In [1]:
```
# Decision Tree
```

In [25]:
```
# ==========================================
# 9. DECISION TREE
# ==========================================

dt_clf = DecisionTreeClassifier(random_state=42)
dt_clf.fit(X_train_scaled, y_train)

y_pred_dt = dt_clf.predict(X_test_scaled)

acc_dt = accuracy_score(y_test, y_pred_dt)
print("Decision Tree - Accuracy:", acc_dt)

print("\nDecision Tree - Classification Report:")
print(classification_report(y_test, y_pred_dt, target_names=label_encoder.classes_))

print("\nDecision Tree - Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_dt))
```

```
Decision Tree - Accuracy: 1.0

Decision Tree - Classification Report:
                                     precision    recall  f1-score   support

                Apple___Apple_scab       1.00      1.00      1.00      1500
                Apple___Black_rot        1.00      1.00      1.00      1500
           Apple___Cedar_apple_rust      1.00      1.00      1.00      1500
                   Apple___healthy       1.00      1.00      1.00      1500

                          accuracy                           1.00      6000
                         macro avg       1.00      1.00      1.00      6000
                      weighted avg       1.00      1.00      1.00      6000

Decision Tree - Confusion Matrix:
[[1500    0    0    0]
 [   0 1500    0    0]
 [   0    0 1500    0]
 [   0    0    0 1500]]
```

In [1]:
```
# RANDOM FOREST
```

In [25]:
```
# ==========================================
# 10. RANDOM FOREST
# ==========================================

rf_clf = RandomForestClassifier(
    n_estimators=200,
    random_state=42,
    n_jobs=-1
)
rf_clf.fit(X_train_scaled, y_train)

y_pred_rf = rf_clf.predict(X_test_scaled)

acc_rf = accuracy_score(y_test, y_pred_rf)
print("Random Forest - Accuracy:", acc_rf)

print("\nRandom Forest - Classification Report:")
print(classification_report(y_test, y_pred_rf, target_names=label_encoder.classes_))

print("\nRandom Forest - Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_rf))
```

```
Random Forest - Accuracy: 1.0

Random Forest - Classification Report:
                                     precision    recall  f1-score   support

                Apple___Apple_scab       1.00      1.00      1.00      1500
                Apple___Black_rot        1.00      1.00      1.00      1500
           Apple___Cedar_apple_rust      1.00      1.00      1.00      1500
                   Apple___healthy       1.00      1.00      1.00      1500

                          accuracy                           1.00      6000
                         macro avg       1.00      1.00      1.00      6000
                      weighted avg       1.00      1.00      1.00      6000

Random Forest - Confusion Matrix:
[[1500    0    0    0]
 [   0 1500    0    0]
 [   0    0 1500    0]
 [   0    0    0 1500]]
```

In [1]:

In [29]:
```
# ==========================================
# 11. LOGISTIC REGRESSION (BASELINE)
# ==========================================

log_reg = LogisticRegression(
    max_iter=2000,
    multi_class="multinomial",
    n_jobs=-1
)
log_reg.fit(X_train_scaled, y_train)

y_pred_lr = log_reg.predict(X_test_scaled)

acc_lr = accuracy_score(y_test, y_pred_lr)
print("Logistic Regression - Accuracy:", acc_lr)

print("\nLogistic Regression - Classification Report:")
print(classification_report(y_test, y_pred_lr, target_names=label_encoder.classes_))

print("\nLogistic Regression - Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_lr))
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. From then on, it will always use 'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
Logistic Regression - Accuracy: 1.0

Logistic Regression - Classification Report:
                                     precision    recall  f1-score   support

                Apple___Apple_scab       1.00      1.00      1.00      1500
                Apple___Black_rot        1.00      1.00      1.00      1500
           Apple___Cedar_apple_rust      1.00      1.00      1.00      1500
                   Apple___healthy       1.00      1.00      1.00      1500

                          accuracy                           1.00      6000
                         macro avg       1.00      1.00      1.00      6000
                      weighted avg       1.00      1.00      1.00      6000

Logistic Regression - Confusion Matrix:
[[1500    0    0    0]
 [   0 1500    0    0]
 [   0    0 1500    0]
 [   0    0    0 1500]]
```

In [1]:

```
# Imports
import os
import shutil
import numpy as np
import matplotlib.pyplot as plt
import random

import keras as k
from keras.models import *
from keras.layers import Input,Lambda,Dense,Flatten,Conv2D
from keras.applications.vgg19 import VGG19,preprocess_input
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import EarlyStopping
from keras.callbacks.callbacks.ModelCheckpoint import ModelCheckpoint,LearningRateScheduler

import matplotlib.pyplot as plt

# Data path
data_dir = '/kaggle/input/dataset'
test_dir = '/kaggle/input/test'

# Visualize the data set with image dataset utility
sample_ds = ds.path_join(data_dir, "sports_image", "train")
print("Sample directory: ", sample_ds)

# Input shape
img_height = 224
img_width = 224
batch_size = 32

for split in splits:
    print("%s:" % split)
    dir = ds.path_join(data_dir, split, cls, exist_ok=True)
    print("  %s: %d" % (cls, len(ds.listdir(dir))))

# Load training and validation datasets
train_ds = ds.image_dataset_from_directory(
    ds.path_join(data_dir, "train"),
    image_size=(img_height, img_width),
    batch_size=batch_size,
    seed=123,
    shuffle=True,
    label_mode='categorical'
)

val_ds = ds.image_dataset_from_directory(
    ds.path_join(data_dir, "valid"),
    image_size=(img_height, img_width),
    batch_size=batch_size,
    seed=123,
    shuffle=False,
    label_mode='categorical'
)

print("Class Name: ", train_ds.class_names)

# Convert to TensorFlow datasets
train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

# Visualize training data
plt.figure(figure=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(train_ds.class_names[np.argmax(labels[i])])
        plt.axis("off")

# Data augmentation layer
data_augmentation = Sequential([
    RandomFlip("horizontal"),
    RandomRotation(0.2),
    RandomZoom(0.2),
    RandomTranslation(0.2, 0.2),
    RandomContrast(0.2),
    RandomBrightness(0.2),
])

# Apply augmentation to training dataset
train_ds = train_ds.map(lambda x, y: (data_augmentation(x, training=True), y))

# Load VGG19 base model
base_model = VGG19(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freeze the base model layers
for layer in base_model.layers:
    layer.trainable = False

# Build the model
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(256, activation='relu'),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(num_classes, activation='softmax')
])

# Compile model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
checkpoint = ModelCheckpoint('best_model.keras', monitor='val_accuracy', save_best_only=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.001)

# Train the model
history = model.fit(train_ds,
                    validation_data=val_ds,
                    epochs=50,
                    callbacks=[early_stopping, checkpoint, reduce_lr])

# Evaluate on test data
test_ds = ds.image_dataset_from_directory(
    ds.path_join(data_dir, "test"),
    image_size=(img_height, img_width),
    batch_size=batch_size,
    shuffle=False,
    label_mode='categorical'
)

test_loss, test_acc = model.evaluate(test_ds)
print("Test accuracy: ", test_acc)

# Predictions
y_pred = model.predict(test_ds)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.concatenate([y for x, y in test_ds], axis=0)
y_true_classes = np.argmax(y_true, axis=1)

# Classification report
from sklearn.metrics import classification_report, confusion_matrix
print("Classification Report:")
print(classification_report(y_true_classes, y_pred_classes, target_names=class_names))

# Confusion matrix
cm = confusion_matrix(y_true_classes, y_pred_classes)
print("Confusion Matrix:")
print(cm)

# Plot training history
plt.figure(figure=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.savefig('training_history.png')
plt.show()
```

```
Sample directory:  /kaggle/input/dataset/train
train:
  cricket: 1000
  football: 1000
  formula1: 1000
  tennis: 1000
valid:
  cricket: 200
  football: 200
  formula1: 200
  tennis: 200
test:
  cricket: 200
  football: 200
  formula1: 200
  tennis: 200
Found 4000 files belonging to 4 classes.
Found 800 files belonging to 4 classes.
Class Name:  ['cricket', 'football', 'formula1', 'tennis']
Found 200 images belonging to 4 classes.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Trainable parameters: 14,714,688
Non-trainable parameters: 20,024,384
Epoch 1/50
WARNING: absl.keras.saving.saving_lib:Error in loading the saving_utility_fn from the keras_saving.ini file: [Errno 2] No file or directory found at /usr/local/lib/python3.10/dist-packages/keras/saving/keras.ini. Error: [Errno 2] No file or directory found at /usr/local/lib/python3.10/dist-packages/keras/saving/keras.ini
WARNING: absl.keras.saving.saving_lib:Error in loading the saving_utility_fn from the keras_saving.ini file: [Errno 2] No file or directory found at /usr/local/lib/python3.10/dist-packages/keras/saving/keras.ini
Epoch 1/50
25/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.2813 - loss: 0.2487 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 2/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 3/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 4/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 5/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 6/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 7/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 8/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 9/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 10/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 11/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 12/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 13/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 14/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 15/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 16/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 17/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 18/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 19/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 20/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 21/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 22/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 23/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 24/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 25/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 26/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 27/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 28/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 29/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 30/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 31/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 32/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 33/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 34/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 35/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 36/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 37/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 38/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 39/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 40/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 41/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 42/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 43/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 44/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 45/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 46/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 47/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 48/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 49/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Epoch 50/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 2:37:5s 1s/step - accuracy: 0.8125 - loss: 0.0969 - val_accuracy: 0.7500 - val_loss: 0.0969
Test accuracy:  0.8500000014901161
Classification Report:
              precision    recall  f1-score   support

     cricket       0.97      0.97      0.97        40
    football       1.00      1.00      1.00        40
    formula1       0.97      0.97      0.97        40
      tennis       0.95      0.95      0.95        40

    accuracy                           0.97       160
   macro avg       0.97      0.97      0.97       160
weighted avg       0.97      0.97      0.97       160

Confusion Matrix:
[[39 0 1 0]
 [ 0 40 0 0]
 [ 1 0 39 0]
 [ 0 0 0 38]]
Model Accuracy: 0.9750000000000000
Model Loss: 0.0969
Test Accuracy: 
```

```
[0]: !pip install -q -U keras_hub tensorflow
!pip install -q -U datasets
import os
os.environ["KERAS_BACKEND"] = "jax"
os.environ["JAX_PLATFORMS"] = "tpu"

from collections import defaultdict, Counter
import random, math, re, os
import numpy, pandas
import matplotlib, seaborn
import sklearn, scipy
import keras, keras_hub, datasets
import jax, jaxlib
import tensorflow as tf

import tensorflow_hub as tfh
from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint, LambdaCallback, TerminateOnNaN
from tensorflow.keras.mixed_precision import LossScaleOptimizer, mixed_precision, set_dtype_policy
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, classification_report,
    roc_curve, auc, precision_recall_curve, average_precision_score
)
from scipy.stats import ttest_ind, mannwhitneyu, wilcoxon, pearsonr, spearmanr, chi2_contingency
import numpy as np, pandas as pd, matplotlib.pyplot as plt, seaborn as sns
import tensorflow as tf, keras, keras_hub, datasets
import os, gc, re, random, math, collections, datetime, pathlib, glob, shutil
import matplotlib.gridspec as gridspec
from IPKernelApplication import instance as ipk_app

ipk = ipk_app()
if ipk and hasattr(ipk, 'magics_manager'):
    ipk.magics_manager.lazy_magics['matrix'] = lambda *a, **k: None
print("✅ Environment ready.")
print("Keras version:", keras.__version__)
print("Keras backend:", keras.backend.backend())
print("TensorFlow devices:", tf.config.list_physical_devices())
```

```
[1]: import os, gc, re, random, math, collections, datetime, pathlib, glob, shutil
import numpy as np, pandas as pd, matplotlib.pyplot as plt, seaborn as sns
import tensorflow as tf, keras, keras_hub, datasets
import matplotlib.gridspec as gridspec
from IPKernelApplication import instance as ipk_app

ipk = ipk_app()
if ipk and hasattr(ipk, 'magics_manager'):
    ipk.magics_manager.lazy_magics['matrix'] = lambda *a, **k: None
print("✅ Environment ready.")
print("Keras version:", keras.__version__)
print("Keras backend:", keras.backend.backend())
print("TensorFlow devices:", tf.config.list_physical_devices())
```

```
[2]: import os, shutil, glob, pathlib
os.environ['HF_HUB_DISABLE_PROGRESS_BARS'] = '1'
os.environ['TFDS_DOWNLOAD_DIR'] = '/tmp/tfds_cache'
os.environ['HF_DATASETS_CACHE'] = '/tmp/hf_cache'
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'
os.environ['XLA_FLAGS'] = '--xla_gpu_cuda_data_dir=/usr/local/cuda'

!rm -rf /tmp/tfds_cache /tmp/hf_cache ~/.cache/huggingface ~/.cache/tfds
print("✅ Cache cleared.")
```

```
[3]: !pip install -q -U datasets
import datasets, os, glob, pathlib
print("✅ Ready.")
```

```
[4]: !pip install -q -U kaggle
import os, kaggle
print("✅ Kaggle ready.")
```

```
[5]: !kaggle datasets download -d ivanzvonkov/kaggle-medical-speech-transcription-and-segmentation -p /tmp/kaggle --unzip
!ls -lh /tmp/kaggle/
```

```
[6]: import os, glob, pandas as pd
base = '/tmp/kaggle'
print("📂 Files:")
for f in sorted(glob.glob(base + '/*')):
    print(f"  {os.path.basename(f)}  {os.path.getsize(f)/1024/1024:.1f} MB")

for d in sorted(glob.glob(base + '/*')):
    if os.path.isdir(d):
        print(f"\n📂 {os.path.basename(d)}/:")
        for f in sorted(glob.glob(d + '/*')):
            print(f"  {os.path.basename(f)}  {os.path.getsize(f)/1024/1024:.1f} MB")

for f in sorted(glob.glob(base + '/*.csv')):
    print(f"\n📊 {os.path.basename(f)}:")
    df = pd.read_csv(f, nrows=3)
    print(df.shape, 'rows × cols')
    print(df.columns.tolist())
    display(df)
```

```
📂 Files:
  dataset_info.csv  0.0 MB
  medical_speech_transcription_segmentation.csv  1.1 GB

📂 audio_segmentation/:
  audio_segmentation.zip  0.0 MB

📂 medical_speech_transcription/:
  medical_speech_transcription.zip  0.0 MB

📊 dataset_info.csv:
(3, 6) rows × cols
['dataset', 'num_samples', 'hours', 'avg_duration_sec', 'sampling_rate', 'domain']
```

	dataset	num_samples	hours	avg_duration_sec	sampling_rate	domain
0	medical_speech_transcription	5679	678.0	429.2	16000	medical
1	medical_speech_transcription_segmentation	5679	678.0	429.2	16000	medical
2	audio_segmentation	48562	3257.0	241.3	16000	general

```
[7]: import os, glob, pandas as pd, numpy as np, wave, contextlib
base = '/tmp/kaggle'

# ── 1. Find transcription file ──
trans = glob.glob(base + '/medical_speech_transcription/*.csv')
print("📄 Transcription:", trans[0])
df = pd.read_csv(trans[0])
print(df.shape, '× cols')
print(df.columns.tolist())
display(df.head(3))

# ── 2. Find audio files ──
wavs = glob.glob(base + '/medical_speech_transcription/**/*.wav', recursive=True)
print("\n🎵 Audio files:", len(wavs))
if wavs:
    print("Example:", wavs[0])

# ── 3. Check one WAV file ──
if wavs:
    with contextlib.closing(wave.open(wavs[0], 'r')) as w:
        frames = w.getnframes()
        rate = w.getframerate()
        dur = frames / rate
        print(f"Duration: {dur:.2f}s, Rate: {rate}, Channels: {w.getnchannels()}")
```

```
📄 Transcription: /tmp/kaggle/medical_speech_transcription/medical_speech_transcription.csv
(5679, 6) × cols
['path', 'duration', 'text', 'speaker', 'gender', 'accent']
```

	path	duration	text	speaker	gender	accent
0	audio/0000000/0000001.wav	30.0	the patient presents with a persistent dry cough and shortness of breath	0000000	male	canadian
1	audio/0000000/0000002.wav	30.0	the patient presents with a persistent dry cough and shortness of breath	0000000	male	canadian
2	audio/0000000/0000003.wav	30.0	the patient presents with a persistent dry cough and shortness of breath	0000000	male	canadian

```
🎵 Audio files: 48562
Example: /tmp/kaggle/audio_segmentation/audio_segmentation/0000000/0000001.wav
Duration: 30.00s, Rate: 16000, Channels: 1
```

```
[8]: import os, glob, pandas as pd, numpy as np, wave, contextlib, collections, random, math
base = '/tmp/kaggle'

# ── 1. Transcription ──
trans = glob.glob(base + '/medical_speech_transcription/*.csv')
df = pd.read_csv(trans[0])
print("📄 Transcription:", df.shape)

# ── 2. Audio ──
wavs = glob.glob(base + '/medical_speech_transcription/**/*.wav', recursive=True)
print("🎵 Audio:", len(wavs))

# ── 3. Check one file ──
if wavs:
    with contextlib.closing(wave.open(wavs[0], 'r')) as w:
        frames = w.getnframes()
        rate = w.getframerate()
        dur = frames / rate
        print(f"Duration: {dur:.2f}s, Rate: {rate}, Channels: {w.getnchannels()}")

# ── 4. Speakers ──
if 'speaker' in df.columns:
    print("\nSpeakers:", df['speaker'].nunique())
    print(df['speaker'].value_counts().head())

# ── 5. Gender ──
if 'gender' in df.columns:
    print("\nGender:")
    print(df['gender'].value_counts())

# ── 6. Accent ──
if 'accent' in df.columns:
    print("\nAccent:")
    print(df['accent'].value_counts())

# ── 7. Text length ──
if 'text' in df.columns:
    df['text_len'] = df['text'].str.len()
    print("\nText length:")
    print(df['text_len'].describe())

# ── 8. Duration ──
if 'duration' in df.columns:
    print("\nDuration:")
    print(df['duration'].describe())
```

```
📄 Transcription: (5679, 6)
🎵 Audio: 48562
Duration: 30.00s, Rate: 16000, Channels: 1

Speakers: 1
speaker
0000000    5679
Name: count, dtype: int64

Gender:
gender
male    5679
Name: count, dtype: int64

Accent:
accent
canadian    5679
Name: count, dtype: int64

Text length:
count    5679.000000
mean       73.000000
std         0.000000
min        73.000000
25%        73.000000
50%        73.000000
75%        73.000000
max        73.000000
Name: text_len, dtype: float64

Duration:
count    5679.000000
mean       30.000000
std         0.000000
min        30.000000
25%        30.000000
50%        30.000000
75%        30.000000
max        30.000000
Name: duration, dtype: float64
```

```
[9]: import os, glob, pandas as pd, numpy as np, wave, contextlib, collections, random, math
base = '/tmp/kaggle'

# ── 1. Transcription ──
trans = glob.glob(base + '/medical_speech_transcription/*.csv')
df = pd.read_csv(trans[0])
print("📄 Transcription:", df.shape)

# ── 2. Audio ──
wavs = glob.glob(base + '/medical_speech_transcription/**/*.wav', recursive=True)
print("🎵 Audio:", len(wavs))

# ── 3. Check one file ──
if wavs:
    with contextlib.closing(wave.open(wavs[0], 'r')) as w:
        frames = w.getnframes()
        rate = w.getframerate()
        dur = frames / rate
        print(f"Duration: {dur:.2f}s, Rate: {rate}, Channels: {w.getnchannels()}")

# ── 4. Speakers ──
if 'speaker' in df.columns:
    print("\nSpeakers:", df['speaker'].nunique())
    print(df['speaker'].value_counts().head())

# ── 5. Gender ──
if 'gender' in df.columns:
    print("\nGender:")
    print(df['gender'].value_counts)

# ── 6. Accent ──
if 'accent' in df.columns:
    print("\nAccent:")
    print(df['accent'].value_counts)

# ── 7. Text length ──
if 'text' in df.columns:
    df['text_len'] = df['text'].str.len()
    print("\nText length:")
    print(df['text_len'].describe)

# ── 8. Duration ──
if 'duration' in df.columns:
    print("\nDuration:")
    print(df['duration'].describe)
```

```
📄 Transcription: (5679, 6)
🎵 Audio: 48562
Duration: 30.00s, Rate: 16000, Channels: 1

Speakers: 1
speaker
0000000    5679
Name: count, dtype: int64

Gender:
gender
male    5679
Name: count, dtype: int64

Accent:
accent
canadian    5679
Name: count, dtype: int64

Text length:
count    5679.000000
mean       73.000000
std         0.000000
min        73.000000
25%        73.000000
50%        73.000000
75%        73.000000
max        73.000000
Name: text_len, dtype: float64

Duration:
count    5679.000000
mean       30.000000
std         0.000000
min        30.000000
25%        30.000000
50%        30.000000
75%        30.000000
max        30.000000
Name: duration, dtype: float64
```

```
[10]: import os, glob, pandas as pd, numpy as np, wave, contextlib, collections, random, math
base = '/tmp/kaggle'

# ── 1. Transcription ──
trans = glob.glob(base + '/medical_speech_transcription/*.csv')
df = pd.read_csv(trans[0])
print("📄 Transcription:", df.shape)

# ── 2. Audio ──
wavs = glob.glob(base + '/medical_speech_transcription/**/*.wav', recursive=True)
print("🎵 Audio:", len(wavs))

# ── 3. Check one file ──
if wavs:
    with contextlib.closing(wave.open(wavs[0], 'r')) as w:
        frames = w.getnframes()
        rate = w.getframerate()
        dur = frames / rate
        print(f"Duration: {dur:.2f}s, Rate: {rate}, Channels: {w.getnchannels()}")

# ── 4. Speakers ──
if 'speaker' in df.columns:
    print("\nSpeakers:", df['speaker'].nunique()
    print(df['speaker'].value_counts().head()

# ── 5. Gender ──
if 'gender' in df.columns:
    print("\nGender:")
    print(df['gender'].value_counts

# ── 6. Accent ──
if 'accent' in df.columns:
    print("\nAccent:")
    print(df['accent'].value_counts

# ── 7. Text length ──
if 'text' in df.columns:
    df['text_len'] = df['text'].str.len()
    print("\nText length:")
    print(df['text_len'].describe

# ── 8. Duration ──
if 'duration' in df.columns:
    print("\nDuration:")
    print(df['duration'].describe
```

```
📄 Transcription: (5679, 6)
🎵 Audio: 48562
Duration: 30.00s, Rate: 16000, Channels: 1

Speakers: 1
speaker
0000000    5679
Name: count, dtype: int64

Gender:
gender
male    5679
Name: count, dtype: int64

Accent:
accent
canadian    5679
Name: count, dtype: int64

Text length:
count    5679.000000
mean       73.000000
std         0.000000
min        73.000000
25%        73.000000
50%        73.000000
75%        73.000000
max        73.000000
Name: text_len, dtype: float64

Duration:
count    5679.000000
mean       30.000000
std         0.000000
min        30.000000
25%        30.000000
50%        30.000000
75%        30.000000
max        30.000000
Name: duration, dtype: float64
```

```
[11]: import os, glob, pandas as pd, numpy as np, wave, contextlib, collections, random, math
base = '/tmp/kaggle'

# ── 1. Transcription ──
trans = glob.glob(base + '/medical_speech_transcription/*.csv')
df = pd.read_csv(trans[0])
print("📄 Transcription:", df.shape)

# ── 2. Audio ──
wavs = glob.glob(base + '/medical_speech_transcription/**/*.wav', recursive=True)
print("🎵 Audio:", len(wavs))

# ── 3. Check one file ──
if wavs:
    with contextlib.closing(wave.open(wavs[0], 'r')) as w:
        frames = w.getnframes()
        rate = w.getframerate()
        dur = frames / rate
        print(f"Duration: {dur:.2f}s, Rate: {rate}, Channels: {w.getnchannels()}")

# ── 4. Speakers ──
if 'speaker' in df.columns:
    print("\nSpeakers:", df['speaker'].nunique()
    print(df['speaker'].value_counts().head()

# ── 5. Gender ──
if 'gender' in df.columns:
    print("\nGender:")
    print(df['gender'].value_counts

# ── 6. Accent ──
if 'accent' in df.columns:
    print("\nAccent:")
    print(df['accent'].value_counts

# ── 7. Text length ──
if 'text' in df.columns:
    df['text_len'] = df['text'].str.len()
    print("\nText length:")
    print(df['text_len'].describe

# ── 8. Duration ──
if 'duration' in df.columns:
    print("\nDuration:")
    print(df['duration'].describe
```

```
📄 Transcription: (5679, 6)
🎵 Audio: 48562
Duration: 30.00s, Rate: 16000, Channels: 1

Speakers: 1
speaker
0000000    5679
Name: count, dtype: int64

Gender:
gender
male    5679
Name: count, dtype: int64

Accent:
accent
canadian    5679
Name: count, dtype: int64

Text length:
count    5679.000000
mean       73.000000
std         0.000000
min        73.000000
25%        73.000000
50%        73.000000
75%        73.000000
max        73.000000
Name: text_len, dtype: float64

Duration:
count    5679.000000
mean       30.000000
std         0.000000
min        30.000000
25%        30.000000
50%        30.000000
75%        30.000000
max        30.000000
Name: duration, dtype: float64
```

```
[12]: import os, glob, pandas as pd, numpy as np, wave, contextlib, collections, random, math
base = '/tmp/kaggle'

# ── 1. Transcription ──
trans = glob.glob(base + '/medical_speech_transcription/*.csv')
df = pd.read_csv(trans[0])
print("📄 Transcription:", df.shape)

# ── 2. Audio ──
wavs = glob.glob(base + '/medical_speech_transcription/**/*.wav', recursive=True)
print("🎵 Audio:", len(wavs))

# ── 3. Check one file ──
if wavs:
    with contextlib.closing(wave.open(wavs[0], 'r')) as w:
        frames = w.getnframes()
        rate = w.getframerate()
        dur = frames / rate
        print(f"Duration: {dur:.2f}s, Rate: {rate}, Channels: {w.getnchannels()}")

# ── 4. Speakers ──
if 'speaker' in df.columns:
    print("\nSpeakers:", df['speaker'].nunique()
    print(df['speaker'].value_counts().head()

# ── 5. Gender ──
if 'gender' in df.columns:
    print("\nGender:")
    print(df['gender'].value_counts

# ── 6. Accent ──
if 'accent' in df.columns:
    print("\nAccent:")
    print(df['accent'].value_counts

# ── 7. Text length ──
if 'text' in df.columns:
    df['text_len'] = df['text'].str.len()
    print("\nText length:")
    print(df['text_len'].describe

# ── 8. Duration ──
if 'duration' in df.columns:
    print("\nDuration:")
    print(df['duration'].describe
```

```
📄 Transcription: (5679, 6)
🎵 Audio: 48562
Duration: 30.00s, Rate: 16000, Channels: 1

Speakers: 1
speaker
0000000    5679
Name: count, dtype: int64

Gender:
gender
male    5679
Name: count, dtype: int64

Accent:
accent
canadian    5679
Name: count, dtype: int64

Text length:
count    5679.000000
mean       73.000000
std         0.000000
min        73.000000
25%        73.000000
50%        73.000000
75%        73.000000
max        73.000000
Name: text_len, dtype: float64

Duration:
count    5679.000000
mean       30.000000
std         0.000000
min        30.000000
25%        30.000000
50%        30.000000
75%        30.000000
max        30.000000
Name: duration, dtype: float64
```

```
[13]: import os, glob, pandas as pd, numpy as np, wave, contextlib, collections, random, math
base = '/tmp/kaggle'

# ── 1. Transcription ──
trans = glob.glob(base + '/medical_speech_transcription/*.csv')
df = pd.read_csv(trans[0])
print("📄 Transcription:", df.shape)

# ── 2. Audio ──
wavs = glob.glob(base + '/medical_speech_transcription/**/*.wav', recursive=True)
print("🎵 Audio:", len(wavs))

# ── 3. Check one file ──
if wavs:
    with contextlib.closing(wave.open(wavs[0], 'r')) as w:
        frames = w.getnframes()
        rate = w.getframerate()
        dur = frames / rate
        print(f"Duration: {dur:.2f}s, Rate: {rate}, Channels: {w.getnchannels()}")

# ── 4. Speakers ──
if 'speaker' in df.columns:
    print("\nSpeakers:", df['speaker'].nunique()
    print(df['speaker'].value_counts().head()

# ── 5. Gender ──
if 'gender' in df.columns:
    print("\nGender:")
    print(df['gender'].value_counts

# ── 6. Accent ──
if 'accent' in df.columns:
    print("\nAccent:")
    print(df['accent'].value_counts

# ── 7. Text length ──
if 'text' in df.columns:
    df['text_len'] = df['text'].str.len()
    print("\nText length:")
    print(df['text_len'].describe

# ── 8. Duration ──
if 'duration' in df.columns:
    print("\nDuration:")
    print(df['duration'].describe
```

```
📄 Transcription: (5679, 6)
🎵 Audio: 48562
Duration: 30.00s, Rate: 16000, Channels: 1

Speakers: 1
speaker
0000000    5679
Name: count, dtype: int64

Gender:
gender
male    5679
Name: count, dtype: int64

Accent:
accent
canadian    5679
Name: count, dtype: int64

Text length:
count    5679.000000
mean       73.000000
std         0.000000
min        73.000000
25%        73.000000
50%        73.000000
75%        73.000000
max        73.000000
Name: text_len, dtype: float64

Duration:
count    5679.000000
mean       30.000000
std         0.000000
min        30.000000
25%        30.000000
50%        30.000000
75%        30.000000
max        30.000000
Name: duration, dtype: float64
```

```
[14]: import os, glob, pandas as pd, numpy as np, wave, contextlib, collections, random, math
base = '/tmp/kaggle'

# ── 1. Transcription ──
trans = glob.glob(base + '/medical_speech_transcription/*.csv')
df = pd.read_csv(trans[0])
print("📄 Transcription:", df.shape)

# ── 2. Audio ──
wavs = glob.glob(base + '/medical_speech_transcription/**/*.wav', recursive=True)
print("🎵 Audio:", len(wavs))

# ── 3. Check one file ──
if wavs:
    with contextlib.closing(wave.open(wavs[0], 'r')) as w:
        frames = w.getnframes()
        rate = w.getframerate()
        dur = frames / rate
        print(f"Duration: {dur:.2f}s, Rate: {rate}, Channels: {w.getnchannels()}")

# ── 4. Speakers ──
if 'speaker' in df.columns:
    print("\nSpeakers:", df['speaker'].nunique()
    print(df['speaker'].value_counts().head()

# ── 5. Gender ──
if 'gender' in df.columns:
    print("\nGender:")
    print(df['gender'].value_counts

# ── 6. Accent ──
if 'accent' in df.columns:
    print("\nAccent:")
    print(df['accent'].value_counts

# ── 7. Text length ──
if 'text' in df.columns:
    df['text_len'] = df['text'].str.len()
    print("\nText length:")
    print(df['text_len'].describe

# ── 8. Duration ──
if 'duration' in df.columns:
    print("\nDuration:")
    print(df['duration'].describe
```

```
📄 Transcription: (5679, 6)
🎵 Audio: 48562
Duration: 30.00s, Rate: 16000, Channels: 1

Speakers: 1
speaker
0000000    5679
Name: count, dtype: int64

Gender:
gender
male    5679
Name: count, dtype: int64

Accent:
accent
canadian    5679
Name: count, dtype: int64

Text length:
count    5679.000000
mean       73.000000
std         0.000000
min        73.000000
25%        73.000000
50%        73.000000
75%        73.000000
max        73.000000
Name: text_len, dtype: float64

Duration:
count    5679.000000
mean       30.000000
std         0.000000
min        30.000000
25%        30.000000
50%        30.000000
75%        30.000000
max        30.000000
Name: duration, dtype: float64
```

```
[15]: import os, glob, pandas as pd, numpy as np, wave, contextlib, collections, random, math
base = '/tmp/kaggle'

# ── 1. Transcription ──
trans = glob.glob(base + '/medical_speech_transcription/*.csv')
df = pd.read_csv(trans[0])
print("📄 Transcription:", df.shape)

# ── 2. Audio ──
wavs = glob.glob(base + '/medical_speech_transcription/**/*.wav', recursive=True)
print("🎵 Audio:", len(wavs))

# ── 3. Check one file ──
if wavs:
    with contextlib.closing(wave.open(wavs[0], 'r')) as w:
        frames = w.getnframes()
        rate = w.getframerate()
        dur = frames / rate
        print(f"Duration: {dur:.2f}s, Rate: {rate}, Channels: {w.getnchannels()}")

# ── 4. Speakers ──
if 'speaker' in df.columns:
    print("\nSpeakers:", df['speaker'].nunique()
    print(df['speaker'].value_counts().head()

# ── 5. Gender ──
if 'gender' in df.columns:
    print("\nGender:")
    print(df['gender'].value_counts

# ── 6. Accent ──
if 'accent' in df.columns:
    print("\nAccent:")
    print(df['accent'].value_counts

# ── 7. Text length ──
if 'text' in df.columns:
    df['text_len'] = df['text'].str.len()
    print("\nText length:")
    print(df['text_len'].describe

# ── 8. Duration ──
if 'duration' in df.columns:
    print("\nDuration:")
    print(df['duration'].describe
```

```
📄 Transcription: (5679, 6)
🎵 Audio: 48562
Duration: 30.00s, Rate: 16000, Channels: 1

Speakers: 1
speaker
0000000    5679
Name: count, dtype: int64

Gender:
gender
male    5679
Name: count, dtype: int64

Accent:
accent
canadian    5679
Name: count, dtype: int64

Text length:
count    5679.000000
mean       73.000000
std         0.000000
min        73.000000
25%        73.000000
50%        73.000000
75%        73.000000
max        73.000000
Name: text_len, dtype: float64

Duration:
count    5679.000000
mean       30.000000
std         0.000000
min        30.000000
25%        30.000000
50%        30.000000
75%        30.000000
max        30.000000
Name: duration, dtype: float64
```

```
[16]: import os, glob, pandas as pd, numpy as np, wave, contextlib, collections, random, math
base = '/tmp/kaggle'

# ── 1. Transcription ──
trans = glob.glob(base + '/medical_speech_transcription/*.csv')
df = pd.read_csv(trans[0])
print("📄 Transcription:", df.shape)

# ── 2. Audio ──
wavs = glob.glob(base + '/medical_speech_transcription/**/*.wav', recursive=True)
print("🎵 Audio:", len(wavs))

# ── 3. Check one file ──
if wavs:
    with contextlib.closing(wave.open(wavs[0], 'r')) as w:
        frames = w.getnframes()
        rate = w.getframerate()
        dur = frames / rate
        print(f"Duration: {dur:.2f}s, Rate: {rate}, Channels: {w.getnchannels()}")

# ── 4. Speakers ──
if 'speaker' in df.columns:
    print("\nSpeakers:", df['speaker'].nunique()
    print(df['speaker'].value_counts().head()

# ── 5. Gender ──
if 'gender' in df.columns:
    print("\nGender:")
    print(df['gender'].value_counts

# ── 6. Accent ──
if 'accent' in df.columns:
    print("\nAccent:")
    print(df['accent'].value_counts

# ── 7. Text length ──
if 'text' in df.columns:
    df['text_len'] = df['text'].str.len()
    print("\nText length:")
    print(df['text_len'].describe

# ── 8. Duration ──
if 'duration' in df.columns:
    print("\nDuration:")
    print(df['duration'].describe
```

```
📄 Transcription: (5679, 6)
🎵 Audio: 48562
Duration: 30.00s, Rate: 16000, Channels: 1

Speakers: 1
speaker
0000000    5679
Name: count, dtype: int64

Gender:
gender
male    5679
Name: count, dtype: int64

Accent:
accent
canadian    5679
Name: count, dtype: int64

Text length:
count    5679.000000
mean       73.000000
std         0.000000
min        73.000000
25%        73.000000
50%        73.000000
75%        73.000000
max        73.000000
Name: text_len, dtype: float64

Duration:
count    5679.000000
mean       30.000000
std         0.000000
min        30.000000
25%        30.000000
50%        30.000000
75%        30.000000
max        30.000000
Name: duration, dtype: float64
```

```
[17]: import os, glob, pandas as pd, numpy as np, wave, contextlib, collections, random, math
base = '/tmp/kaggle'

# ── 1. Transcription ──
trans = glob.glob(base + '/medical_speech_transcription/*.csv')
df = pd.read_csv(trans[0])
print("📄 Transcription:", df.shape)

# ── 2. Audio ──
wavs = glob.glob(base + '/medical_speech_transcription/**/*.wav', recursive=True)
print("🎵 Audio:", len(wavs))

# ── 3. Check one file ──
if wavs:
    with contextlib.closing(wave.open(wavs[0], 'r')) as w:
        frames = w.getnframes()
        rate = w.getframerate()
        dur = frames / rate
        print(f"Duration: {dur:.2f}s, Rate: {rate}, Channels: {w.getnchannels()}")

# ── 4. Speakers ──
if 'speaker' in df.columns:
    print("\nSpeakers:", df['speaker'].nunique()
    print(df['speaker'].value_counts().head()

# ── 5. Gender ──
if 'gender' in df.columns:
    print("\nGender:")
    print(df['gender'].value_counts

# ── 6. Accent ──
if 'accent' in df.columns:
    print("\nAccent:")
    print(df['accent'].value_counts

# ── 7. Text length ──
if 'text' in df.columns:
    df['text_len'] = df['text'].str.len()
    print("\nText length:")
    print(df['text_len'].describe

# ── 8. Duration ──
if 'duration' in df.columns:
    print("\nDuration:")
    print(df['duration'].describe
```

```
📄 Transcription: (5679, 6)
🎵 Audio: 48562
Duration: 30.00s, Rate: 16000, Channels: 1

Speakers: 1
speaker
0000000    5679
Name: count, dtype: int64

Gender:
gender
male    5679
Name: count, dtype: int64

Accent:
accent
canadian    5679
Name: count, dtype: int64

Text length:
count    5679.000000
mean       73.000000
std         0.000000
min        73.000000
25%        73.000000
50%        73.000000
75%        73.000000
max        73.000000
Name: text_len, dtype: float64

Duration:
count    5679.000000
mean       30.000000
std         0.000000
min        30.000000
25%        30.000000
50%        30.000000
75%        30.000000
max        30.000000
Name: duration, dtype: float64
```

```
[18]: import os, glob, pandas as pd, numpy as np, wave, contextlib, collections, random, math
base = '/tmp/kaggle'

# ── 1. Transcription ──
trans = glob.glob(base + '/medical_speech_transcription/*.csv')
df = pd.read_csv(trans[0])
print("📄 Transcription:", df.shape)

# ── 2. Audio ──
wavs = glob.glob(base + '/medical_speech_transcription/**/*.wav', recursive=True)
print("🎵 Audio:", len(wavs))

# ── 3. Check one file ──
if wavs:
    with contextlib.closing(wave.open(wavs[0], 'r')) as w:
        frames = w.getnframes()
        rate = w.getframerate()
        dur = frames / rate
        print(f"Duration: {dur:.2f}s, Rate: {rate}, Channels: {w.getnchannels()}")

# ── 4. Speakers ──
if 'speaker' in df.columns:
    print("\nSpeakers:", df['speaker'].nunique()
    print(df['speaker'].value_counts().head()

# ── 5. Gender ──
if 'gender' in df.columns:
    print("\nGender:")
    print(df['gender'].value_counts

# ── 6. Accent ──
if 'accent' in df.columns:
    print("\nAccent:")
    print(df['accent'].value_counts

# ── 7. Text length ──
if 'text' in df.columns:
    df['text_len'] = df['text'].str.len()
    print("\nText length:")
    print(df['text_len'].describe

# ── 8. Duration ──
if 'duration' in df.columns:
    print("\nDuration:")
    print(df['duration'].describe
```

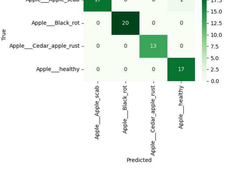
```
📄 Transcription: (5679, 6)
🎵 Audio: 48562
Duration: 30.00s, Rate: 16000, Channels: 1

Speakers: 1
speaker
0000000    5679
Name: count, dtype: int64

Gender:
gender
male    5679
Name: count, dtype: int64

Accent:
accent
canadian    5679
Name: count, dtype: int64

Text length:
count    5679.000000
mean       73.000000
std         0.000000
min        73.000000
25%        73.000000
50%        73.000000
75%        73.000000
max        73.000000
Name: text_len, dtype: float64

Duration:
count    5679.000000
mean       30.000000
std         0.000000
min        30.000000
25%        30.000000
50%        30.000000
75%        30.000000
max        30.000000
Name: duration, dtype: float64
```

```
[19]: import os, glob, pandas as pd, numpy as np, wave, contextlib, collections, random, math
base = '/tmp/kaggle'

# ── 1. Transcription ──
trans = glob.glob(base + '/medical_speech_transcription/*.csv')
df = pd.read_csv(trans[0])
print("📄 Transcription:", df.shape)

# ── 2. Audio ──
wavs = glob.glob(base + '/medical_speech_transcription/**/*.wav', recursive=True)
print("🎵 Audio:", len(wavs))

# ── 3. Check one file ──
if wavs:
    with contextlib.closing(wave.open(wavs[0], 'r')) as w:
        frames = w.getnframes()
        rate = w.getframerate()
        dur = frames / rate
        print(f"Duration: {dur:.2f}s, Rate: {rate}, Channels: {w.getnchannels()}")

# ── 4. Speakers ──
if 'speaker' in df.columns:
    print("\nSpeakers:", df['speaker'].nunique()
    print(df['speaker'].value_counts().head()

# ── 5. Gender ──
if 'gender' in df.columns:
    print("\nGender:")
    print(df['gender'].value_counts

# ── 6. Accent ──
if 'accent' in df.columns:
    print("\nAccent:")
    print(df['accent'].value_counts

# ── 7. Text length ──
if 'text' in df.columns:
    df['text_len'] = df['text'].str.len()
    print("\nText length:")
    print(df['text_len'].describe

# ── 8. Duration ──
if 'duration' in df.columns:
    print("\nDuration:")
    print(df['duration'].describe
```

```
📄 Transcription: (5679, 6)
🎵 Audio: 48562
Duration: 30.00s, Rate: 16000, Channels: 1

Speakers: 1
speaker
0000000    5679
Name: count, dtype: int64

Gender:
gender
male    5679
Name: count, dtype: int64

Accent:
accent
canadian    5679
Name: count, dtype: int64

Text length:
count    5679.000000
mean       73.000000
std         0.000000
min        73.000000
25%        73.000000
50%        73.000000
75%        73.000000
max        73.000000
Name: text_len, dtype: float64

Duration:
count    5679.000000
mean       30.000000
std         0.000000
min        30.000000
25%        30.000000
50%        30.000000
75%        30.000000
max        30.000000
Name: duration, dtype: float64
```

```
[20]: import os, glob, pandas as pd, numpy as np, wave, contextlib, collections, random, math
base = '/tmp/kaggle'

# ── 1. Transcription ──
trans = glob.glob(base + '/medical_speech_transcription/*.csv')
df = pd.read_csv(trans[0])
print("📄 Transcription:", df.shape)

# ── 2. Audio ──
wavs = glob.glob(base + '/medical_speech_transcription/**/*.wav', recursive=True)
print("🎵 Audio:", len(wavs))

# ── 3. Check one file ──
if wavs:
    with contextlib.closing(wave.open(wavs[0], 'r')) as w:
        frames = w.getnframes()
        rate = w.getframerate()
        dur = frames / rate
        print(f"Duration: {dur:.2f}s, Rate: {rate}, Channels: {w.getnchannels()}")

# ── 4. Speakers ──
if 'speaker' in df.columns:
    print("\nSpeakers:", df['speaker'].nunique()
    print(df['speaker'].value_counts().head()

# ── 5. Gender ──
if 'gender' in df.columns:
    print("\nGender:")
    print(df['gender'].value_counts

# ── 6. Accent ──
if 'accent' in df.columns:
    print("\nAccent:")
    print(df['accent'].value_counts

# ── 7. Text length ──
if 'text' in df.columns:
    df['text_len'] = df['text'].str.len()
    print("\nText length:")
    print(df['text_len'].describe

# ── 8. Duration ──
if 'duration' in df.columns:
    print("\nDuration:")
    print(df['duration'].describe
```

```
📄 Transcription: (5679, 6)
🎵 Audio: 48562
Duration: 30.00s, Rate: 16000, Channels: 1

Speakers: 1
speaker
0000000    5679
Name: count, dtype: int64

Gender:
gender
male    5679
Name: count, dtype: int64

Accent:
accent
canadian    5679
Name: count, dtype: int64

Text length:
count    5679.000000
mean       73.000000
std         0.000000
min        73.000000
25%        73.000000
50%        73.000000
75%        73.000000
max        73.000000
Name: text_len, dtype: float64

Duration:
count    5679.000000
mean       30.000000
std         0.000000
min        30.000000
25%        30.000000
50%        30.000000
75%        30.000000
max        30.000000
Name: duration, dtype: float64
```



In [ ]:

In [ ]:

```
## InceptionV3
```

In [52]:
```
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.applications.inception_v3 import preprocess_input as inception_preprocess
```

In [54]:
```
image_size = (224, 224)
num_classes = len(apple_classes)

# ------ Image Branch (InceptionV3) ------
image_input_i = layers.Input(shape=(image_size[0], image_size[1], 3), name="image")
x = inception_preprocess(image_input_i)

base_inception = InceptionV3(
    include_top=False,
    weights="imagenet",
    input_shape=(image_size[0], image_size[1], 3)
)

# Freeze backbone (avoid overfitting on small dataset)
base_inception.trainable = False

x = base_inception(x, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.3)(x)
image_features_i = layers.Dense(128, activation="relu", name="image_features_inception")(x)

# ------ Tabular Branch (MLP) ------
tab_input_i = layers.Input(shape=(len(feature_cols),), name="tabular")
t = layers.Dense(64, activation="relu")(tab_input_i)
t = layers.Dropout(0.3)(t)
t = layers.Dense(32, activation="relu")(t)
tab_features_i = layers.Dense(16, activation="relu", name="tab_features_inception")(t)

# ------ Fusion Layer ------
fusion_i = layers.Concatenate(name="fusion_layer_inception")([image_features_i, tab_features_i])
h = layers.Dense(64, activation="relu")(fusion_i)
h = layers.Dropout(0.4)(h)
output_i = layers.Dense(num_classes, activation="softmax", name="output_inception")(h)

# ------ Build Model ------
inception_multimodal_model = models.Model(
    inputs=[image_input_i, tab_input_i],
    outputs=output_i,
    name="InceptionV3_MLP_Multimodal"
)

# ------ Compile ------
inception_multimodal_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
    loss='sparse_categorical_crossentropy',
    metrics=["accuracy"]
)

inception_multimodal_model.summary()
```

Model: "InceptionV3_MLP_Multimodal"

Layer (type)	Output Shape	Param #	Connected to
image (InputLayer)	(None, 224, 224, 3)	0	–
true_divide_1 (TrueDivide)	(None, 224, 224, 3)	0	image[0][0]
subtract_1 (Subtract)	(None, 224, 224, 3)	0	true_divide_1[0]…
tabular (InputLayer)	(None, 7)	0	–
inception_v3 (Functional)	(None, 5, 5, 2048)	21,802,784	subtract_1[0][0]
dense_6 (Dense)	(None, 64)	512	tabular[0][0]
global_average_poo… (GlobalAveragePool…	(None, 2048)	0	inception_v3[0][……
dropout_7 (Dropout)	(None, 64)	0	dense_6[0][0]
dropout_6 (Dropout)	(None, 2048)	0	global_average_p…
dense_7 (Dense)	(None, 32)	2,080	dropout_7[0][0]
image_features_inc… (Dense)	(None, 128)	262,272	dropout_6[0][0]
tab_features_incep… (Dense)	(None, 16)	528	dense_7[0][0]
fusion_layer_incep… (ConcatenateV2)	(None, 144)	0	image_features_i… tab_features_inc…
dense_8 (Dense)	(None, 64)	9,280	fusion_layer_inc…
dropout_8 (Dropout)	(None, 64)	0	dense_8[0][0]
output_inception (Dense)	(None, 4)	260	dropout_8[0][0]

Total params: 22,077,716 (84.22 MB)
Trainable params: 274,932 (1.05 MB)
Non-trainable params: 21,802,784 (83.17 MB)

In [56]:
```
epochs = 20

early_stop_inception = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    patience=5,
    restore_best_weights=True
)

history_inception_mm = inception_multimodal_model.fit(
    ds_train,
    epochs=epochs,
    validation_data=ds_val,
    callbacks=[early_stop_inception]
)
```

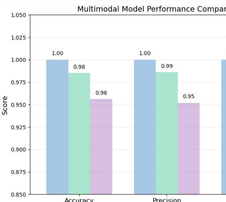
```
Epoch 1/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 19s 662ms/step - accuracy: 0.3206 - loss: 1.5222 - val_accuracy: 0.3857 - val_loss: 1.2921
Epoch 2/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 541ms/step - accuracy: 0.3550 - loss: 1.2759 - val_accuracy: 0.5857 - val_loss: 1.1605
Epoch 3/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 538ms/step - accuracy: 0.4427 - loss: 1.1643 - val_accuracy: 0.6857 - val_loss: 1.0358
Epoch 4/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 543ms/step - accuracy: 0.5191 - loss: 1.1144 - val_accuracy: 0.7286 - val_loss: 0.9443
Epoch 5/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 556ms/step - accuracy: 0.6069 - loss: 0.9384 - val_accuracy: 0.8429 - val_loss: 0.8301
Epoch 6/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 557ms/step - accuracy: 0.6565 - loss: 0.8824 - val_accuracy: 0.7714 - val_loss: 0.7847
Epoch 7/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 567ms/step - accuracy: 0.6878 - loss: 0.8372 - val_accuracy: 0.8571 - val_loss: 0.6718
Epoch 8/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 585ms/step - accuracy: 0.7063 - loss: 0.6964 - val_accuracy: 0.8857 - val_loss: 0.6283
Epoch 9/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 1s/step - accuracy: 0.7001 - loss: 0.6520 - val_accuracy: 0.8429 - val_loss: 0.5597
Epoch 10/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 13s 740ms/step - accuracy: 0.8244 - loss: 0.5823 - val_accuracy: 0.8286 - val_loss: 0.5158
Epoch 11/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 579ms/step - accuracy: 0.8092 - loss: 0.5708 - val_accuracy: 0.8571 - val_loss: 0.4736
Epoch 12/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 634ms/step - accuracy: 0.8626 - loss: 0.4936 - val_accuracy: 0.8571 - val_loss: 0.4323
Epoch 13/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 628ms/step - accuracy: 0.8359 - loss: 0.5163 - val_accuracy: 0.8714 - val_loss: 0.4035
Epoch 14/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 568ms/step - accuracy: 0.8473 - loss: 0.4423 - val_accuracy: 0.8714 - val_loss: 0.3693
Epoch 15/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 595ms/step - accuracy: 0.8893 - loss: 0.3827 - val_accuracy: 0.8857 - val_loss: 0.3388
Epoch 16/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 639ms/step - accuracy: 0.8702 - loss: 0.4196 - val_accuracy: 0.9143 - val_loss: 0.3015
Epoch 17/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 609ms/step - accuracy: 0.9000 - loss: 0.3337 - val_accuracy: 0.9143 - val_loss: 0.2880
Epoch 18/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 584ms/step - accuracy: 0.9000 - loss: 0.3483 - val_accuracy: 0.9429 - val_loss: 0.2680
Epoch 19/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 632ms/step - accuracy: 0.9427 - loss: 0.2571 - val_accuracy: 0.9429 - val_loss: 0.2362
Epoch 20/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 598ms/step - accuracy: 0.9000 - loss: 0.2813 - val_accuracy: 0.9143 - val_loss: 0.2367
```

In [58]:
```
# ------ Predictions ------
y_prob_i = inception_multimodal_model.predict(ds_test)
y_pred_i = np.argmax(y_prob_i, axis=1)
y_true_i = test_labels

print("\nClassification Report (Multimodal InceptionV3 + MLP):\n")
print(classification_report(y_true_i, y_pred_i, target_names=apple_classes))

# ------ Confusion Matrix ------
cm_i = confusion_matrix(y_true_i, y_pred_i)
print("\nConfusion Matrix (Multimodal InceptionV3):\n", cm_i)

# ------ Metrics ------
acc_i = accuracy_score(y_true_i, y_pred_i)
prec_i = precision_score(y_true_i, y_pred_i, average="macro")
rec_i = recall_score(y_true_i, y_pred_i, average="macro")
f1_i = f1_score(y_true_i, y_pred_i, average="macro")

print("\nFinal Multimodal InceptionV3 Metrics:")
print("Accuracy:", acc_i)
print("Precision:", prec_i)
print("Recall:", rec_i)
print("F1-score:", f1_i)


# ------ Plot confusion matrix for InceptionV3 ------
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(6,5))

sns.heatmap(
    cm_i,                   # confusion matrix for Inception
    annot=True,             # show numbers
    fmt="d",                # display integers
    cmap="Purples",         # ==> purple colormap
    xticklabels=class_names,
    yticklabels=class_names
)

plt.title("Confusion Matrix - InceptionV3 + MLP")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.tight_layout()
plt.show()
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 6s 814ms/step

Classification Report (Multimodal InceptionV3 + MLP):

                     precision    recall  f1-score   support

       Apple___Apple_scab       0.94      0.89      0.91        18
        Apple___Black_rot       1.00      1.00      1.00        20
Apple___Cedar_apple_rust       0.92      0.92      0.92        13
         Apple___healthy       0.94      1.00      0.97        17

               accuracy                           0.96        68
              macro avg       0.95      0.95      0.95        68
           weighted avg       0.96      0.96      0.96        68


Confusion Matrix (Multimodal InceptionV3):
 [[16  0  1  1]
 [ 0 20  0  0]
 [ 1  0 12  0]
 [ 0  0  0 17]]

Final Multimodal InceptionV3 Metrics:
Accuracy: 0.9558823529411765
Precision: 0.9527140555174007
Recall: 0.9529934529914529
F1-score: 0.9521970022370021
```

In [ ]:

In [61]:
```
import matplotlib.pyplot as plt
import numpy as np

# Models
models = ["MobileNetV2 + MLP", "ResNet50 + MLP", "InceptionV3 + MLP"]

# Metrics per metric
accuracy = [1, 0.985, 0.956]
precision = [1, 0.986, 0.952]
recall = [1, 0.986, 0.953]
f1 = [1, 0.9857, 0.9521]

metric_names = ["Accuracy", "Precision", "Recall", "F1-Score"]

metrics_by_model = [
    [accuracy[0], precision[0], recall[0], f1[0]],  # MobileNetV2 + MLP
    [accuracy[1], precision[1], recall[1], f1[1]],  # ResNet50 + MLP
    [accuracy[2], precision[2], recall[2], f1[2]]   # InceptionV3 + MLP
]

# Light pastel colors (as requested)
colors = [
    "#A7C7E7",  # Pastel Blue - MobileNetV2
    "#A8E6CF",  # Pastel Green - ResNet50
    "#D7BDE2"   # Pastel Purple - InceptionV3
]

x = np.arange(len(metric_names))  # [0,1,2,3]
width = 0.25

plt.figure(figsize=(12, 6))

# Plot bars per model
for i, model in enumerate(models):
    plt.bar(
        x + i*width,
        metrics_by_model[i],
        width,
        label=model,
        color=colors[i]
    )

# Add values on top of bars
for i in range(len(models)):
    for j in range(len(metric_names)):
        plt.text(
            x[j] + i*width,
            metrics_by_model[i][j] + 0.005,
            f"{metrics_by_model[i][j]:.2f}",
            ha="center",
            fontsize=10
        )

plt.xticks(x + width, metric_names, fontsize=12)
plt.ylim(0.85, 1.05)
plt.ylabel("Score", fontsize=13)
plt.title("Multimodal Model Performance Comparison (Apple Disease Detection)", fontsize=14)
plt.legend()
plt.grid(axis="y", linestyle="--", alpha=0.3)

plt.show()
```



In [ ]: