1. **app.py-DESIGNING AN API USING FLASK FOR SPAM FILERTING (app.py)**

# app.py

from flask import Flask, request, jsonify

from utils import load\_model\_and\_vectorizer, train\_spam\_model\_with\_grid\_search

app = Flask(\_\_name\_\_)

# Global variables to hold the trained model and vectorizer

spam\_classifier = None

tfidf\_vectorizer = None

@app.route('/')

def home():

    """

    Home route to provide basic information about the API endpoints

    """

    return jsonify({

        "message": "Welcome to the Email Spam Detector API!",

        "endpoints": {

            "/predict": "POST request with 'email\_content' in form data to classify text.",

            "/training": "POST request with optional hyperparameters to train/retrain the model using Grid Search and MLflow."

        }

    })

@app.route('/training', methods=['POST'])

def training\_endpoint():

    """

    Endpoint to trigger model training with optional hyperparameter grid

    Logs results and artifacts to MLflow

    """

    global spam\_classifier, tfidf\_vectorizer

    # Get hyperparameters grid from incoming JSON request, or use empty dict

    hyperparams\_grid = request.get\_json() or {}

    # Call training function with hyperparameters grid

    training\_info = train\_spam\_model\_with\_grid\_search(hyperparams\_grid)

    # Reload trained model and vectorizer into global variables

    spam\_classifier, tfidf\_vectorizer = load\_model\_and\_vectorizer()

    # Prepare classification report for JSON response

    report\_output = {}

    if training\_info.get('report'):

        for class\_name, metrics in training\_info['report'].items():

            if isinstance(metrics, dict):

                report\_output[class\_name] = {k: float(v) if isinstance(v, (float, int)) else v for k, v in metrics.items()}

            else:

                report\_output[class\_name] = metrics

    # Prepare the full response data

    response\_data = {

        "status": training\_info.get('status'),

        "best\_hyperparameters": training\_info.get('best\_params'),

        "accuracy": f"{training\_info['accuracy']:.4f}" if training\_info.get('accuracy') else None,

        "classification\_report": report\_output if report\_output else None,

        "mlflow\_run\_id": training\_info.get('mlflow\_run\_id')

    }

    # Return appropriate HTTP response based on status

    if "Error" in training\_info.get('status', ''):

        return jsonify(response\_data), 500

    else:

        return jsonify(response\_data), 200

if \_\_name\_\_ == '\_\_main\_\_':

    # Load model and vectorizer at app startup

    spam\_classifier, tfidf\_vectorizer = load\_model\_and\_vectorizer()

    app.run(debug=True)

# utils.py

import pickle

import os

import mlflow

import mlflow.sklearn

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.metrics import classification\_report, accuracy\_score

# File paths for model and vectorizer persistence

MODEL\_PATH = 'spam\_model.pkl'

VECTORIZER\_PATH = 'tfidf\_vectorizer.pkl'

def train\_spam\_model\_with\_grid\_search(hyperparams\_grid):

    """

    Train Random Forest Classifier using Grid Search for hyperparameter tuning

    Log parameters, metrics, and model artifact to MLflow

    """

    try:

        # Load dataset (ensure 'spam\_data.csv' exists with 'text' and 'label' columns)

        df = pd.read\_csv('spam\_data.csv')

        X = df['text']

        y = df['label']

        # Convert text to numerical feature vectors

        vectorizer = TfidfVectorizer(max\_features=5000, ngram\_range=(1,2))

        X\_vectorized = vectorizer.fit\_transform(X)

        # Split into training and testing sets

        X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_vectorized, y, test\_size=0.2, random\_state=42)

        # Set default hyperparameter grid if not provided

        if not hyperparams\_grid:

            hyperparams\_grid = {

                'n\_estimators': [100, 200],

                'max\_depth': [10, 20, None],

                'min\_samples\_split': [2, 5],

                'min\_samples\_leaf': [1, 2]

            }

        # Initialize Random Forest Classifier

        rf = RandomForestClassifier(random\_state=42)

        # Setup Grid Search with cross-validation

        grid\_search = GridSearchCV(estimator=rf,

                                   param\_grid=hyperparams\_grid,

                                   cv=3,

                                   scoring='accuracy',

                                   n\_jobs=-1)

        # Start MLflow run for experiment tracking

        with mlflow.start\_run() as run:

            grid\_search.fit(X\_train, y\_train)

            best\_model = grid\_search.best\_estimator\_

            best\_params = grid\_search.best\_params\_

            # Make predictions and evaluate

            y\_pred = best\_model.predict(X\_test)

            accuracy = accuracy\_score(y\_test, y\_pred)

            report = classification\_report(y\_test, y\_pred, output\_dict=True)

            # Log best parameters and accuracy to MLflow

            mlflow.log\_params(best\_params)

            mlflow.log\_metric('accuracy', accuracy)

            # Log classification report metrics to MLflow

            for label, metrics in report.items():

                if isinstance(metrics, dict):

                    for metric\_name, value in metrics.items():

                        mlflow.log\_metric(f"{label}\_{metric\_name}", value)

            # Log model artifact to MLflow

            mlflow.sklearn.log\_model(best\_model, "spam\_rf\_model")

            mlflow.log\_artifact('spam\_data.csv')

            # Save model and vectorizer locally

            with open(MODEL\_PATH, 'wb') as f:

                pickle.dump(best\_model, f)

            with open(VECTORIZER\_PATH, 'wb') as f:

                pickle.dump(vectorizer, f)

        return {

            'status': 'Model trained successfully with Grid Search and logged to MLflow',

            'accuracy': accuracy,

            'report': report,

            'best\_params': best\_params,

            'mlflow\_run\_id': run.info.run\_id

        }

    except Exception as e:

        return {

            'status': f'Error during training: {str(e)}'

        }

def load\_model\_and\_vectorizer():

    """

    Load trained model and vectorizer from disk if available

    """

    if not os.path.exists(MODEL\_PATH) or not os.path.exists(VECTORIZER\_PATH):

        return None, None

    with open(MODEL\_PATH, 'rb') as f:

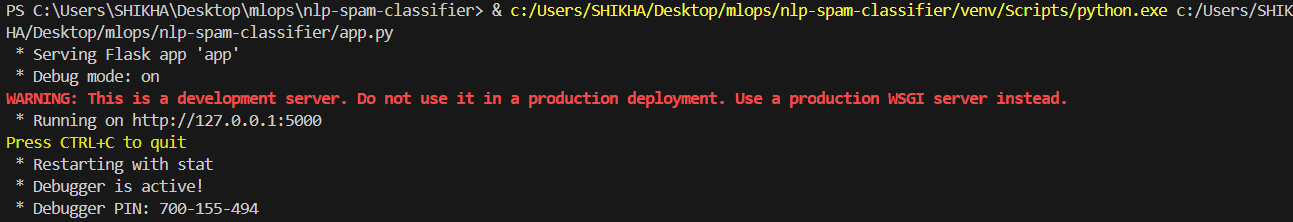
        model = pickle.load(f)

    with open(VECTORIZER\_PATH, 'rb') as f:

        vectorizer = pickle.load(f)

    return model, vectorizer

**OUTPUT:** Running app.py in Visual Studio Code shows that the API is running at <http://127.0.0.01:5000> as shown below,



1. **utils.py: TRAINING & TESTING THE MODEL (Random Forest) and tuning Hyper parameters using GridSearch & tracking using MLFLOW**

import pickle

import os

import mlflow

import mlflow.sklearn

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.metrics import classification\_report, accuracy\_score

# Set MLflow to use the running MLflow server

mlflow.set\_tracking\_uri("http://127.0.0.1:5001")

# File paths for local saving of model and vectorizer

MODEL\_PATH = 'spam\_model.pkl'

VECTORIZER\_PATH = 'tfidf\_vectorizer.pkl'

def train\_spam\_model\_with\_grid\_search(hyperparams\_grid):

    """

    Train Random Forest Classifier using Grid Search for hyperparameter tuning

    Log parameters, metrics, and model artifact to MLflow server

    """

    try:

        # Load dataset (must be present in the same directory)

        df = pd.read\_csv('spam\_data.csv')

        X = df['text']

        y = df['label']

        # Convert text to TF-IDF vectors

        vectorizer = TfidfVectorizer(max\_features=5000, ngram\_range=(1, 2))

        X\_vectorized = vectorizer.fit\_transform(X)

        # Split dataset

        X\_train, X\_test, y\_train, y\_test = train\_test\_split(

            X\_vectorized, y, test\_size=0.2, random\_state=42)

        # Set default hyperparameter grid if not provided

        if not hyperparams\_grid:

            hyperparams\_grid = {

                'n\_estimators': [100, 200],

                'max\_depth': [10, 20, None],

                'min\_samples\_split': [2, 5],

                'min\_samples\_leaf': [1, 2]

            }

        # Initialize model and GridSearchCV

        rf = RandomForestClassifier(random\_state=42)

        grid\_search = GridSearchCV(estimator=rf,

                                   param\_grid=hyperparams\_grid,

                                   cv=3,

                                   scoring='accuracy',

                                   n\_jobs=-1)

        with mlflow.start\_run() as run:

            # Train model

            grid\_search.fit(X\_train, y\_train)

            best\_model = grid\_search.best\_estimator\_

            best\_params = grid\_search.best\_params\_

            # Evaluate

            y\_pred = best\_model.predict(X\_test)

            accuracy = accuracy\_score(y\_test, y\_pred)

            report = classification\_report(y\_test, y\_pred, output\_dict=True)

            # Log parameters and metrics

            mlflow.log\_params(best\_params)

            mlflow.log\_metric('accuracy', accuracy)

            for label, metrics in report.items():

                if isinstance(metrics, dict):

                    for metric\_name, value in metrics.items():

                        mlflow.log\_metric(f"{label}\_{metric\_name}", value)

            # Log model artifact

            mlflow.sklearn.log\_model(best\_model, "spam\_rf\_model")

            # Log the dataset as artifact (optional)

            mlflow.log\_artifact('spam\_data.csv')

            # Save locally as well

            with open(MODEL\_PATH, 'wb') as f:

                pickle.dump(best\_model, f)

            with open(VECTORIZER\_PATH, 'wb') as f:

                pickle.dump(vectorizer, f)

        return {

            'status': 'Model trained successfully with Grid Search and logged to MLflow server',

            'accuracy': accuracy,

            'report': report,

            'best\_params': best\_params,

            'mlflow\_run\_id': run.info.run\_id

        }

    except Exception as e:

        return {

            'status': f'Error during training: {str(e)}'

        }

def load\_model\_and\_vectorizer():

    """

    Load model and vectorizer from disk if available

    """

    if not os.path.exists(MODEL\_PATH) or not os.path.exists(VECTORIZER\_PATH):

        return None, None

    with open(MODEL\_PATH, 'rb') as f:

        model = pickle.load(f)

    with open(VECTORIZER\_PATH, 'rb') as f:

        vectorizer = pickle.load(f)

    return model, vectorizer

**3. call\_training.py: Calling the util.py to train & test the model by passing the hyper-parameters,**

import requests

import json

# Define the URL of the Flask app's training endpoint

url = "http://127.0.0.1:5000/training"

# Optional: Define hyperparameter grid (can leave empty {})

hyperparams = {

    "n\_estimators": [100, 200],

    "max\_depth": [10, 20, None],

    "min\_samples\_split": [2, 5],

    "min\_samples\_leaf": [1, 2]

}

# Send POST request with hyperparameters as JSON

response = requests.post(url, json=hyperparams)

# Print the response from the server

print("Status Code:", response.status\_code)

try:

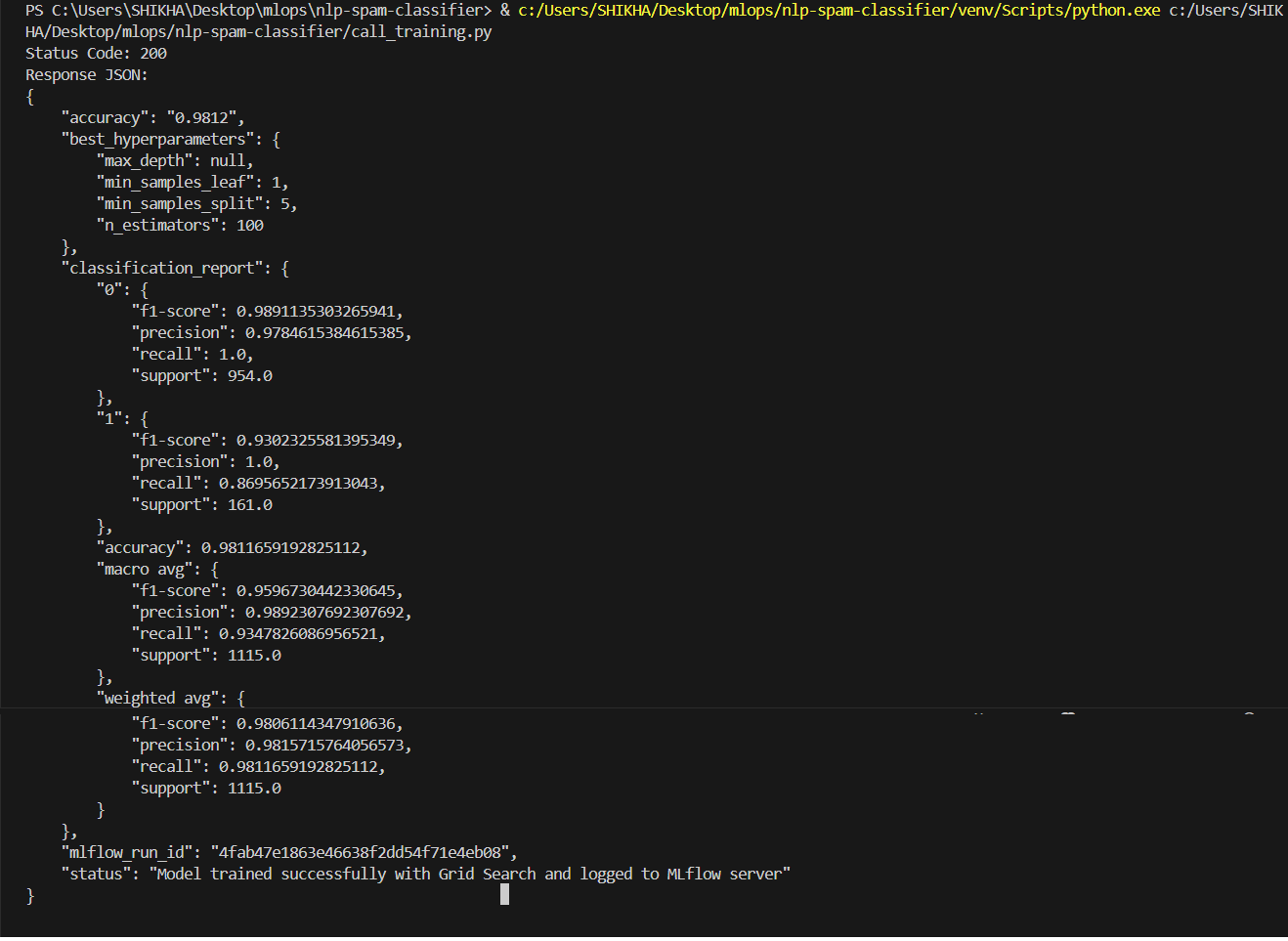
    print("Response JSON:")

    print(json.dumps(response.json(), indent=4))

except Exception as e:

    print("Error parsing response:", str(e))

**OUTPUT:** Running call\_training.py code shows the following output (hyper-parameter tuning) on terminal in Visual Studio Code,



**The MLFlow running @** [**http://127.0.0.1:5001**](http://127.0.0.1:5001) **shows the following progress,**

