**Lab session 6: Integrating Machine Learning Model in a Flutter application**

Task-1 TensorFlow Lite Tutorial for Flutter: Image Classification

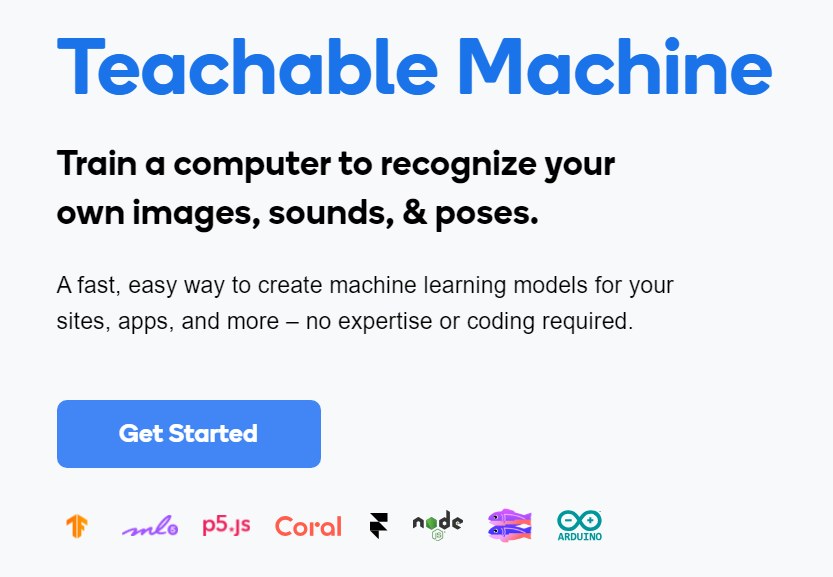
In this task, we will delve into the utilization of TensorFlow Lite within the Flutter framework. This involves training a machine learning model using Teachable Machine and seamlessly integrating the results into a Flutter mobile app.

Getting Started: To initiate this process, we will utilize a provided starter project as our foundation. While the project currently allows users to select or drag-and-drop images into the app, it lacks image recognition functionality. In the upcoming sections, we will employ TensorFlow Lite to address this limitation.

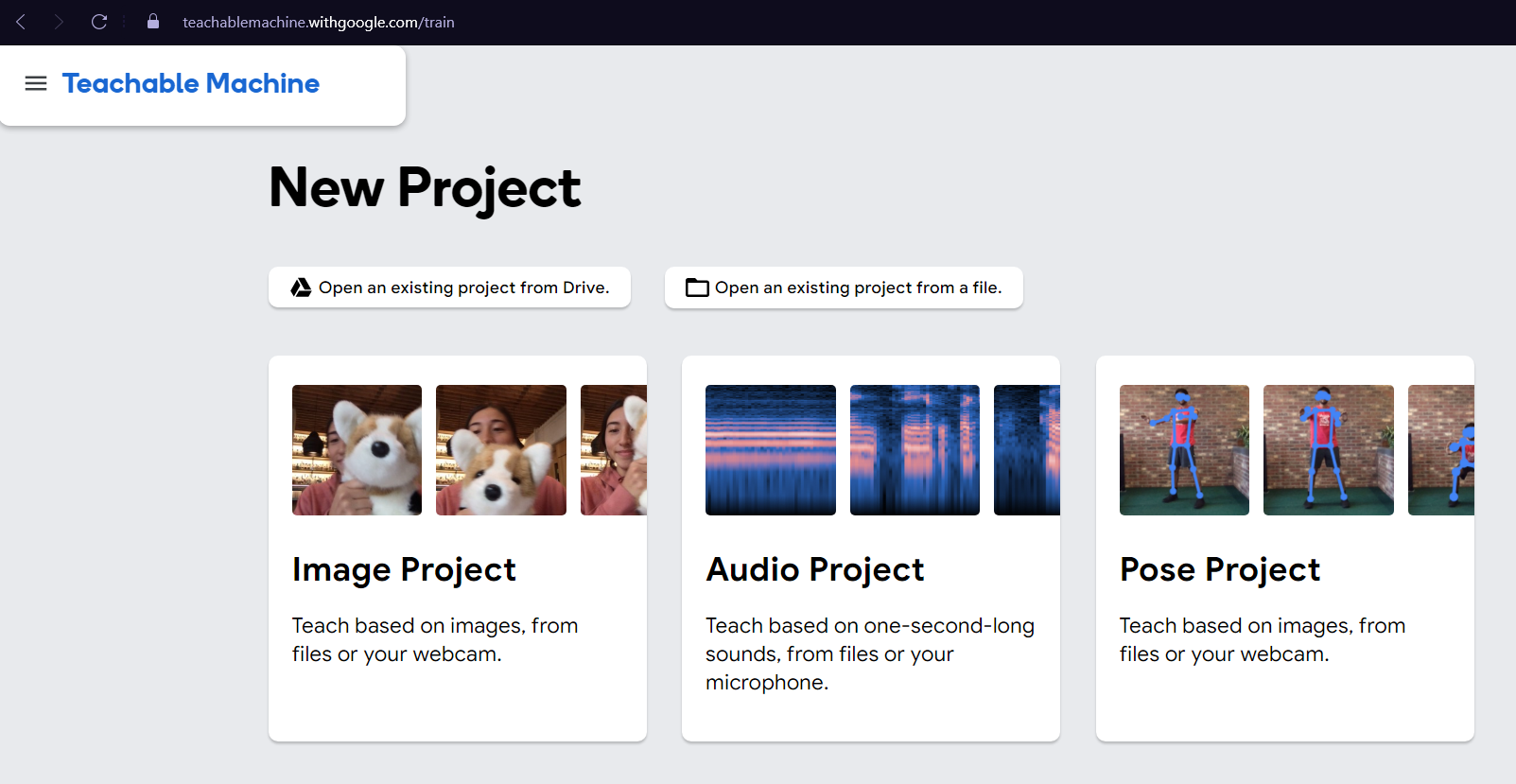
Building a Model with Teachable Machine: The starter project already incorporates a pre-trained model (model\_unquant.tflite) and classification labels stored in the labels.txt file. However, our focus will be on understanding and engaging in the training process itself.

Preparing the Dataset - Training the Model: Training is the pivotal stage where the computer acquires knowledge from data and formulates rules. Our exploration will involve learning how to train the model using Teachable Machine.

1. To commence, visit [https://teachablemachine.withgoogle.com](https://teachablemachine.withgoogle.com/) and click on "Get Started" to access the training tool.

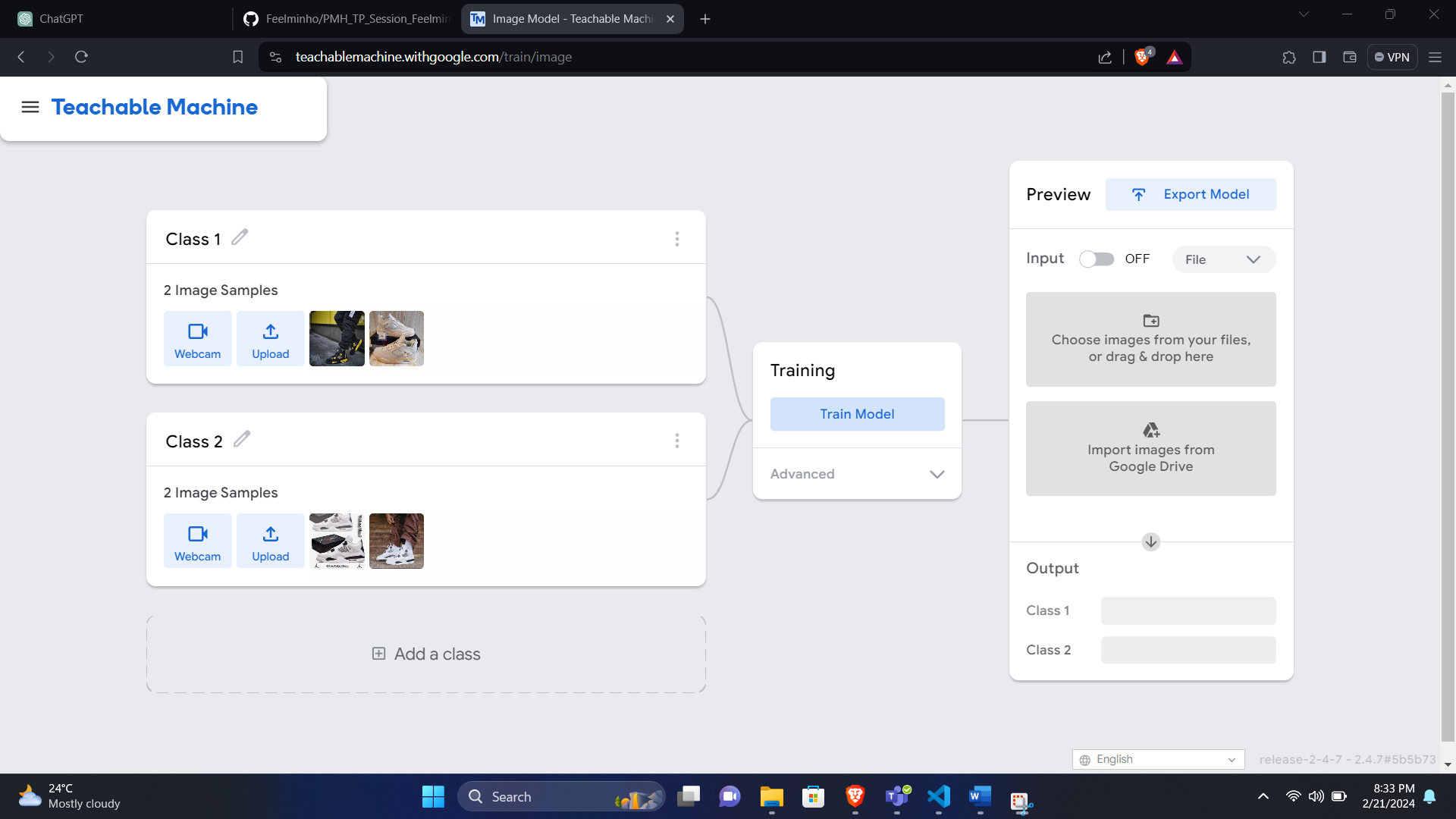


1. Then select Image Project. We Choose Standard Image Model, because we are not training a model to run on a microcontroller

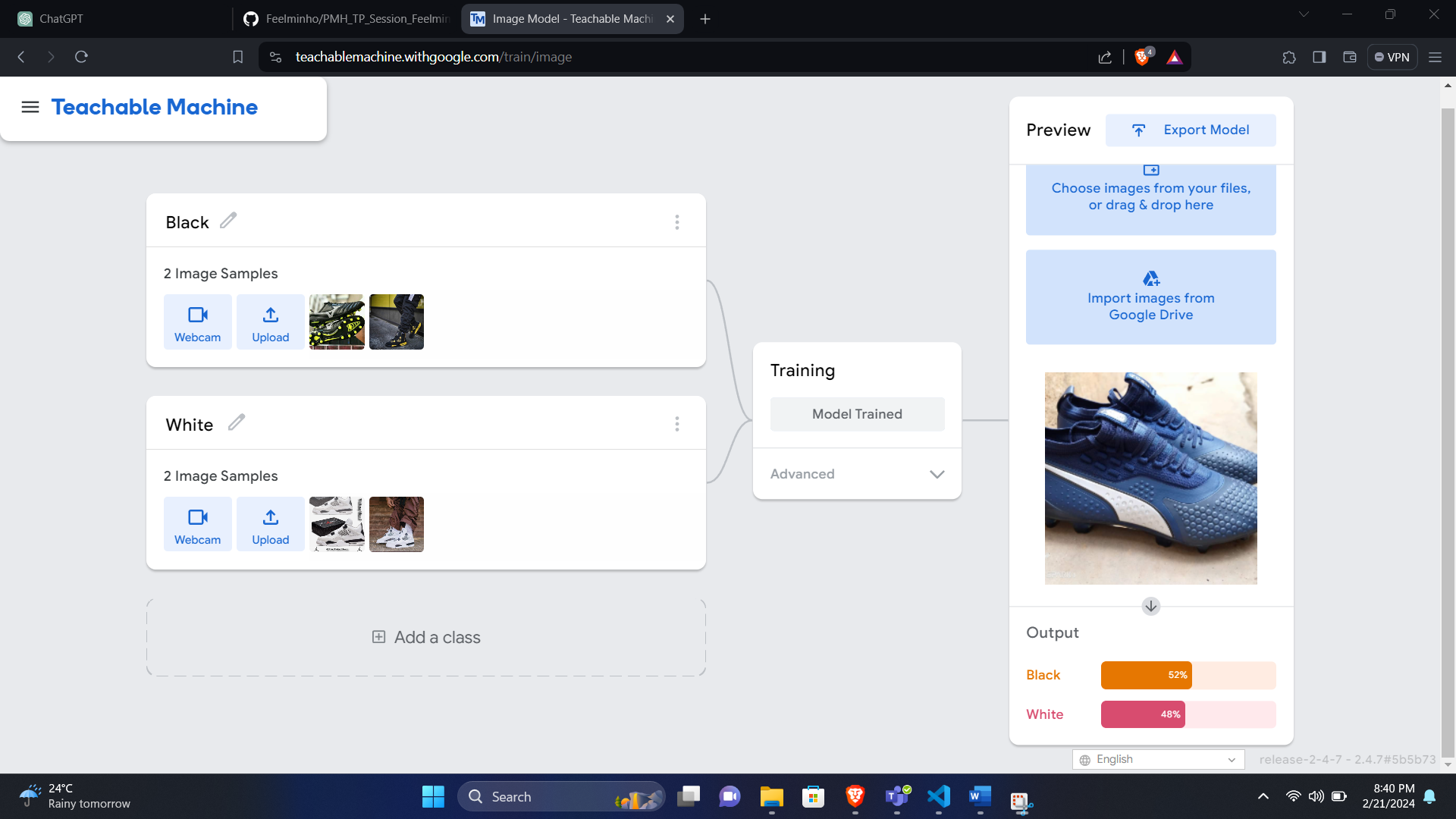




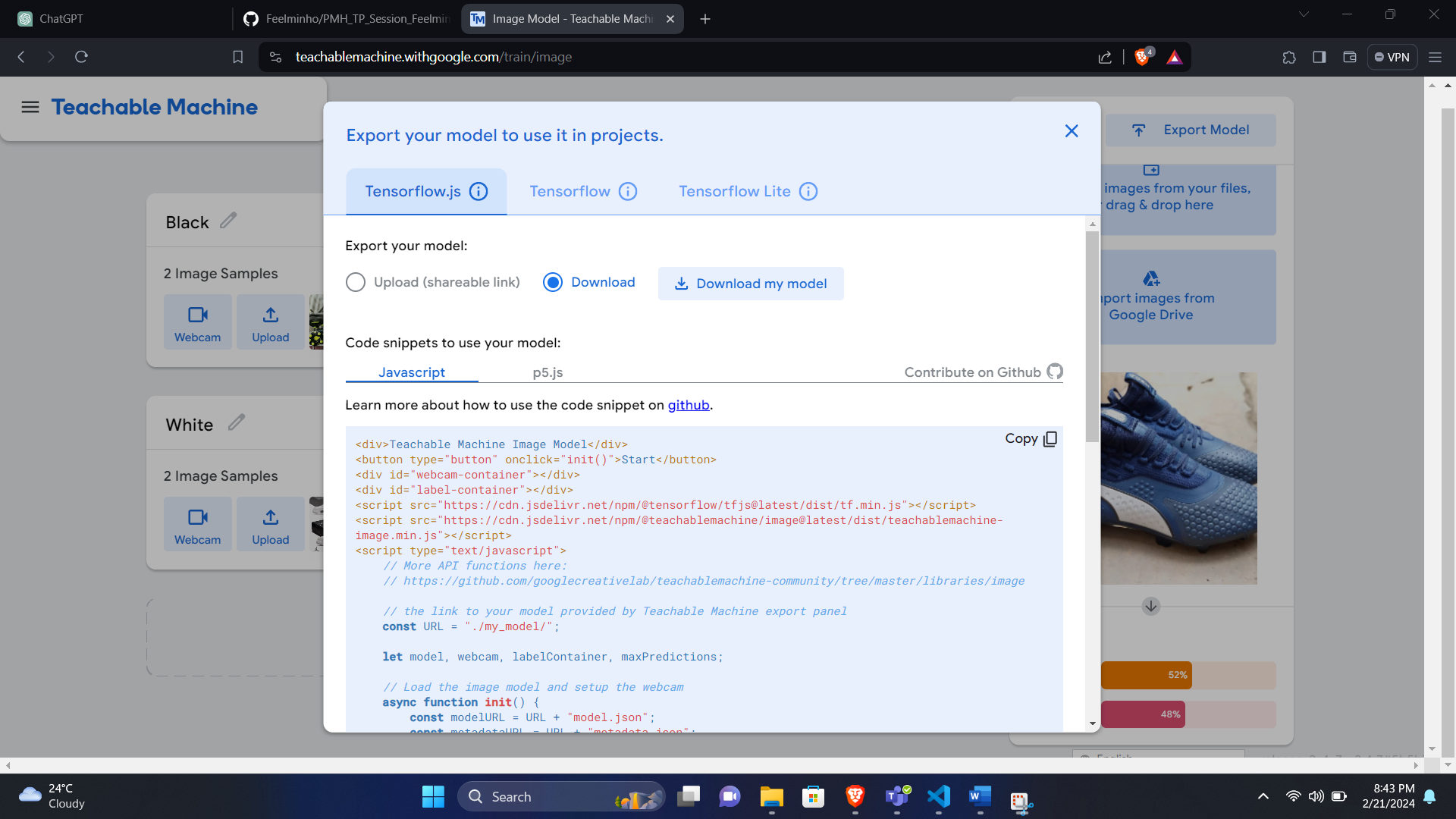
1. Next, add the training samples by clicking Upload under each class. Then, drag the folder of the appropriate plant type from the samples folder to the Choose images from your files … panel.



1. After the training completes, test the model with other images. Use the images in the samples-test folder, like so:



1. Finally, export the model by clicking Export Model on the Preview panel. In the dialog, choose TensorFlow Lite. That’s because the target platform is mobile.
2. Next, select Floating point conversion type for the best predictive performance. Then, click Download my model to convert and download the model.



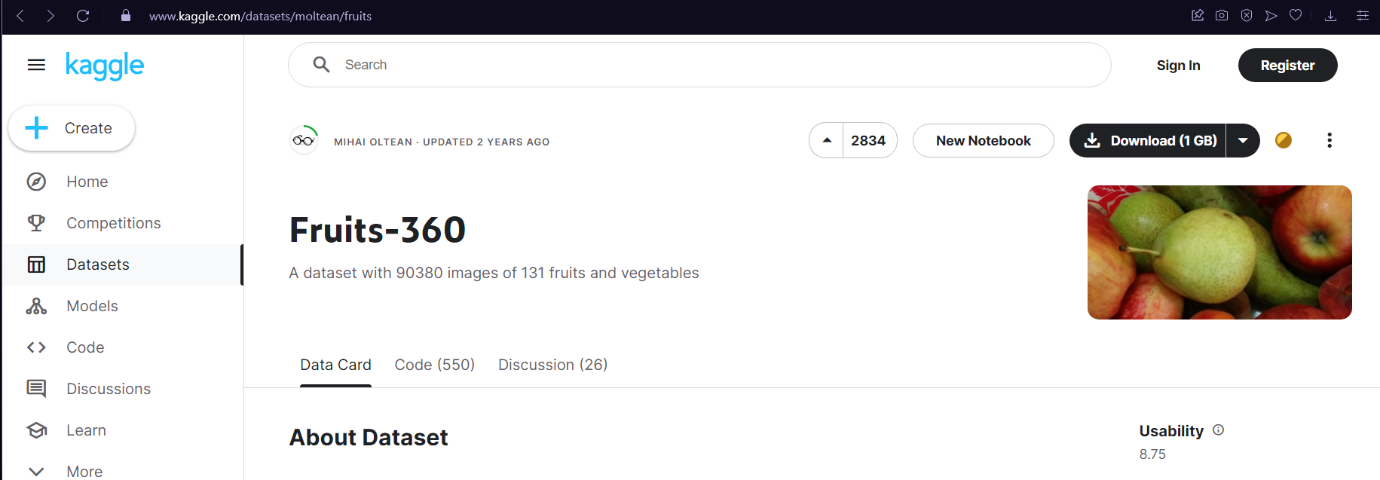
Task-2 Create an image classification Flutter application

In this Task, we will implement an image classification Flutter application that will recognize/identify fruits and vegetable images.

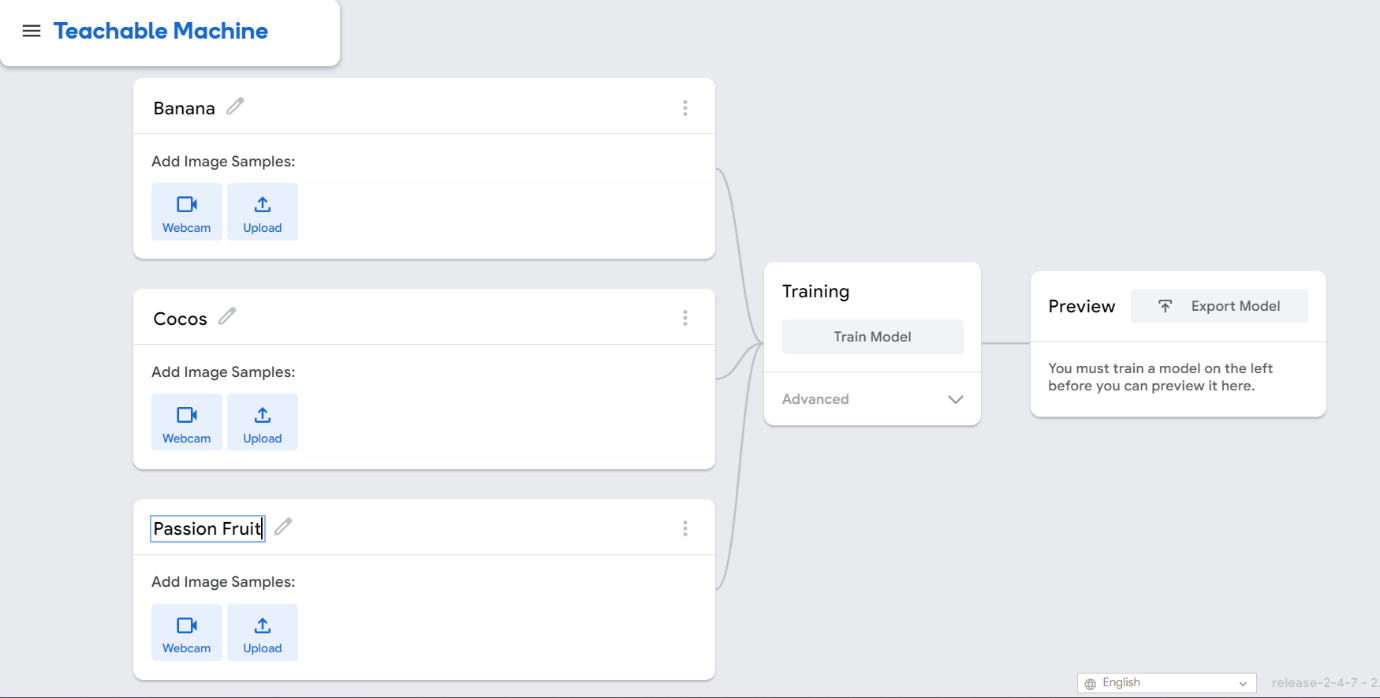
To create this application, we will build a model using Teachable Machine like we have just done in the previous task. This time we will use the Fruits-360 datasets from Kaggle to train the model.

Building a Model with Teachable Machine

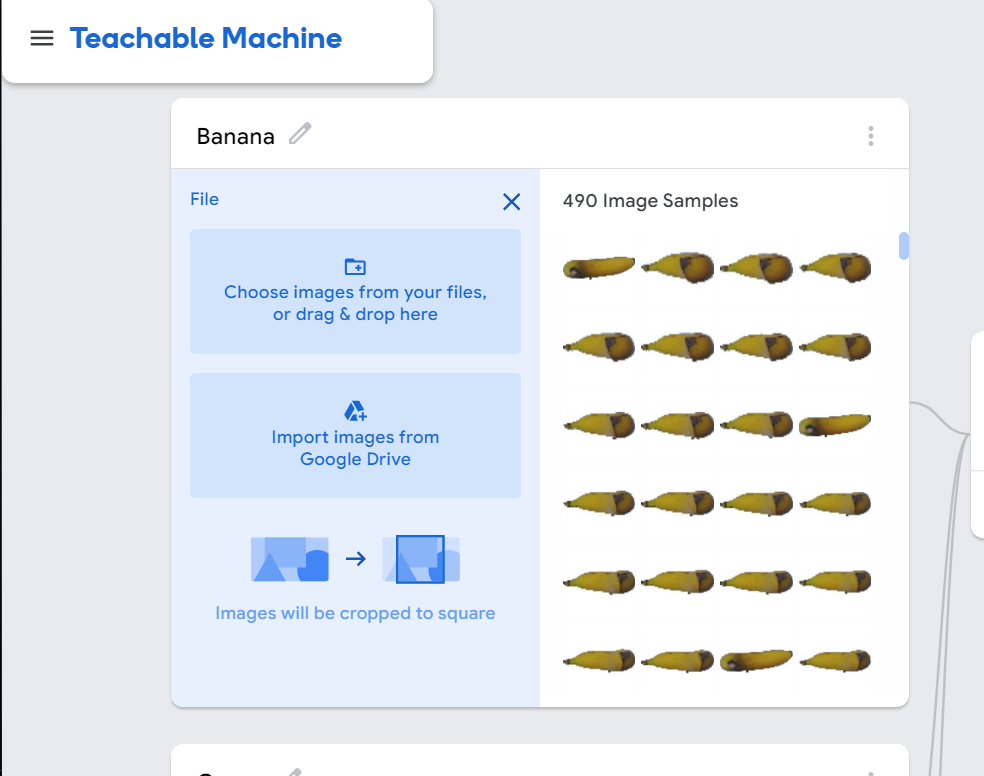
1. We will first download the dataset from Kaggle:



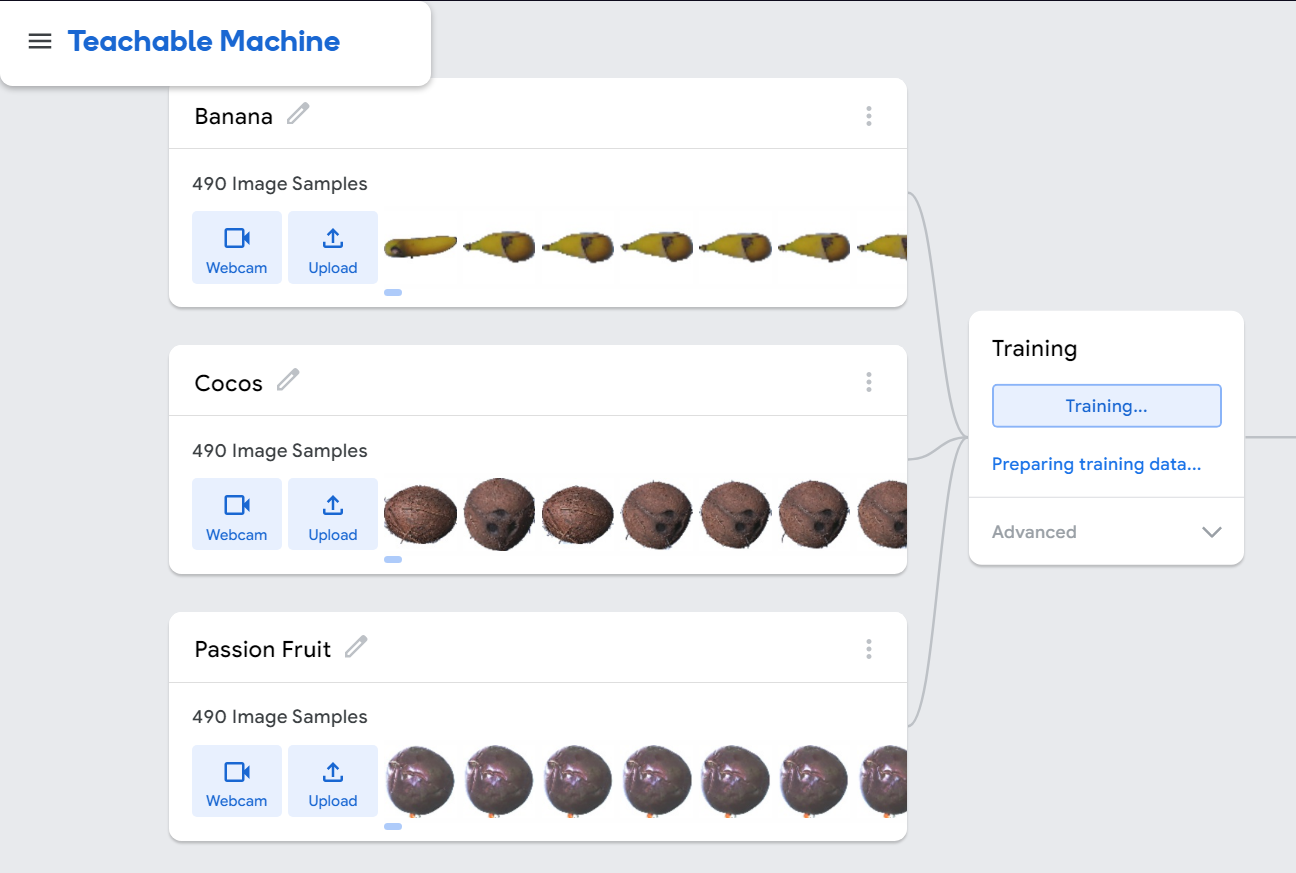
1. Next, we will create a new Image Project in Teachable machine and choose Standard Image Model when prompted.
2. Once in the training tool, add the classes and edit the labels of each class, as shown below:



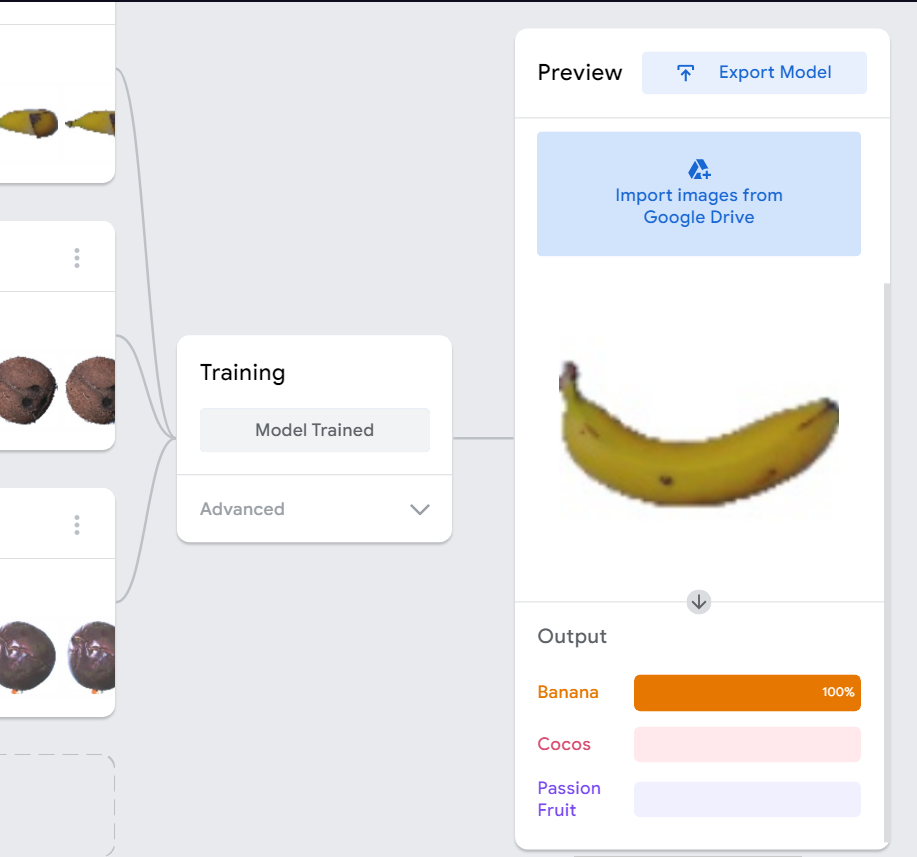
1. Next, add the training samples by clicking Upload under each class. Then, drag the folder of the appropriate plant type from the samples folder to the Choose images from your files panel.



1. After you’ve added all the training samples, click Train Model to train the model:



1. After the training completes, test the model with a fruit image. Use the images in the test folder, like so:

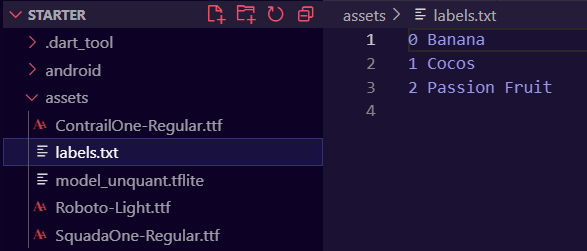


1. Finally, export the model by clicking Export Model on the Preview panel. In the dialog, choose TensorFlow Lite. That’s because the target platform is mobile.
2. After you have the model file converted\_tflite.zip in hand, decompress it and copy labels.txt and model\_unquant.tflite to the ./assets folder in the starter project.

Implement the flutter app

We will try to modify the project created in Task 1 to recognize the fruit instead of plant.

1. We will first replace the machine learning model – model\_unquant.tflite and the classification labels — labels.txt in the project plant recognition by those we got from the Teachable Machine platform.



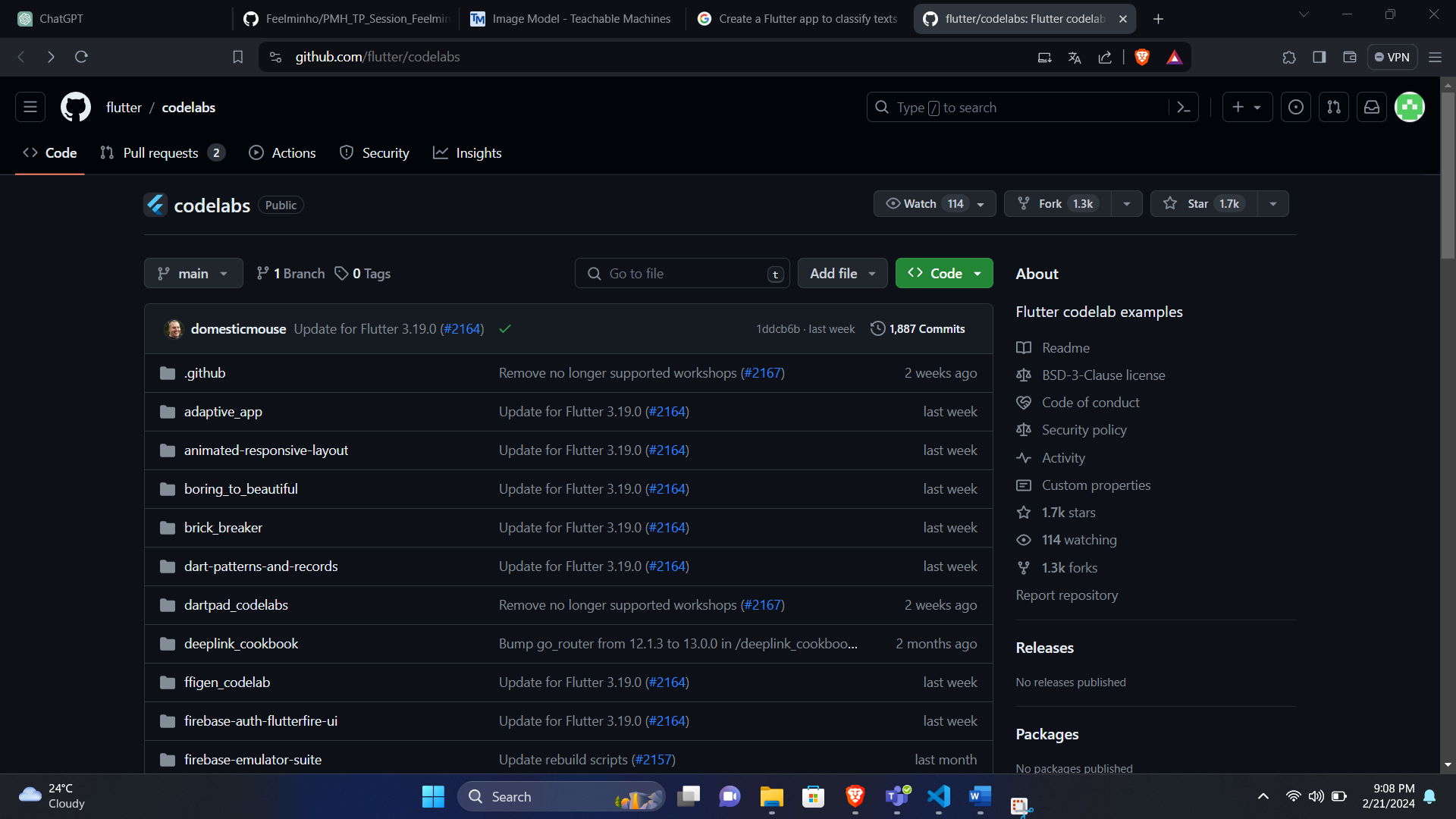
**Task-3**

Create a Flutter app to classify texts

This documentation guides you through executing a text classification inference from a Flutter application using TensorFlow Serving via both REST and gRPC protocols.

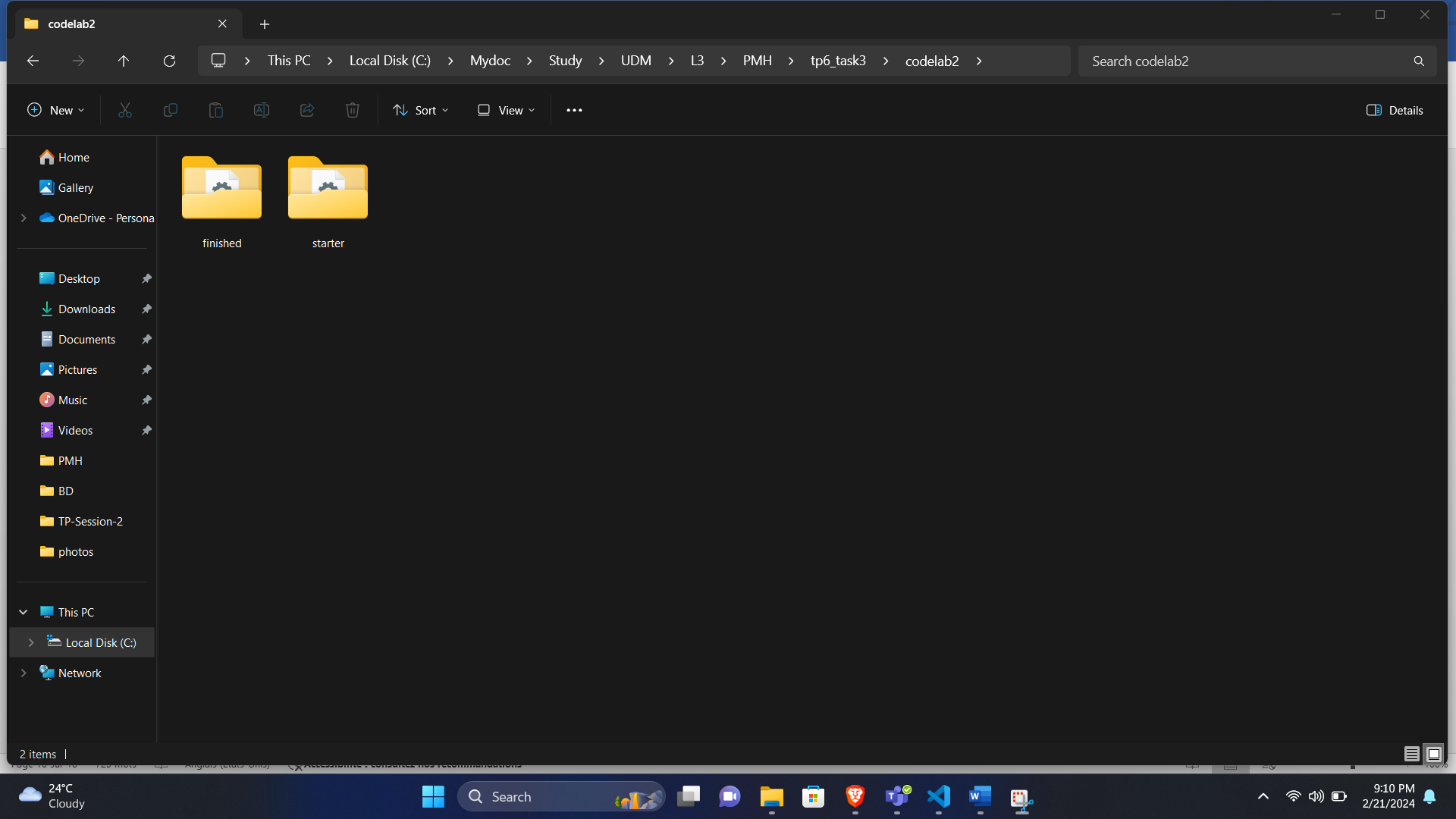
To obtain the code for this codelab, follow these steps:

1. Visit the GitHub repository dedicated to this codelab.
2. Click on the "Code" button and select "Download ZIP" to acquire all the necessary code files for this tutorial.



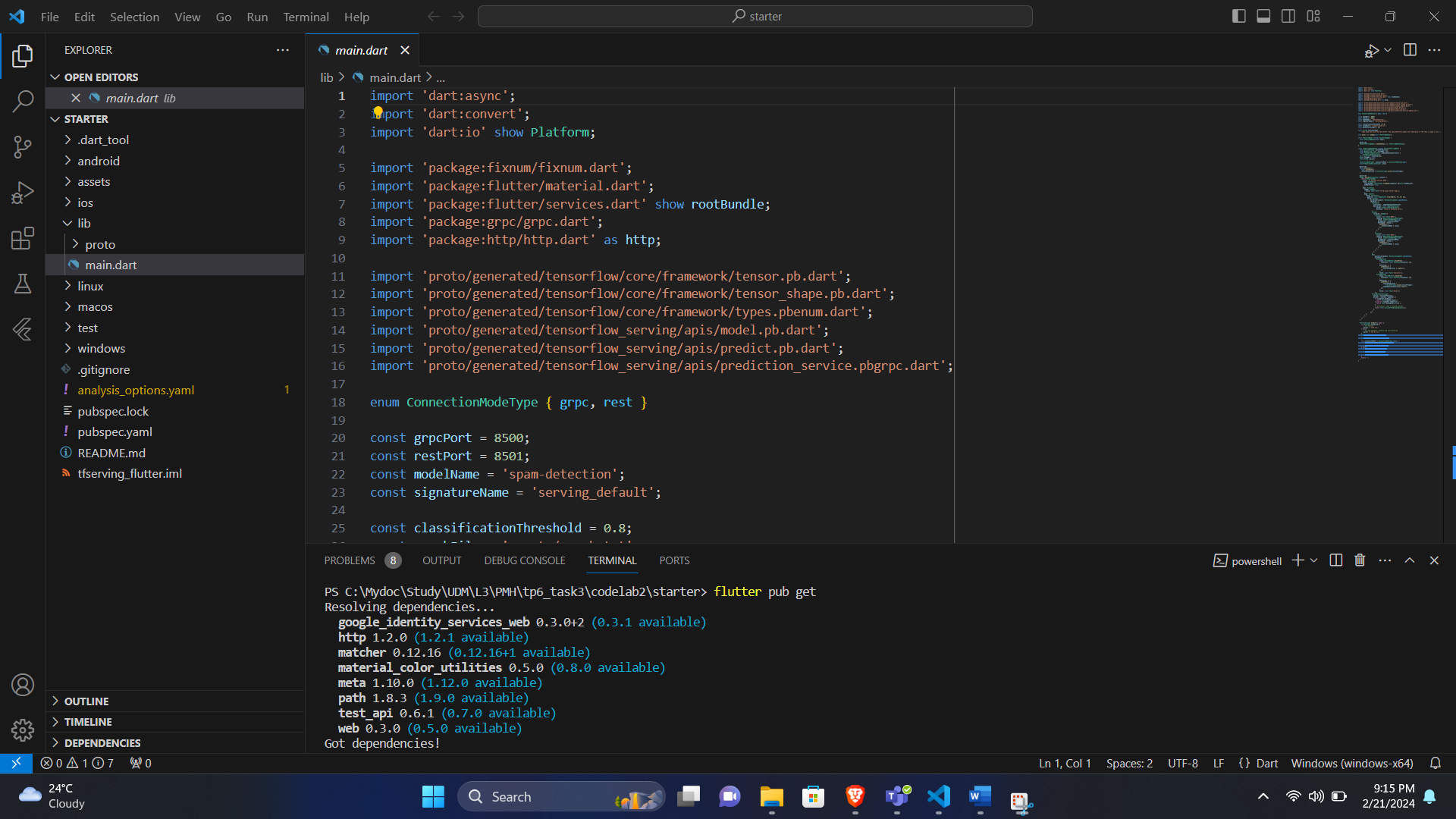
In this codelab, you will exclusively utilize the files located in the "tfserving-flutter/codelab2" subdirectory within the repository. This subdirectory comprises two main folders:

* **Starter Folder:** This contains the initial code that serves as the foundation for your work in this codelab.
* **Finished Folder:** Here, you can find the fully developed code representing the completed sample app.



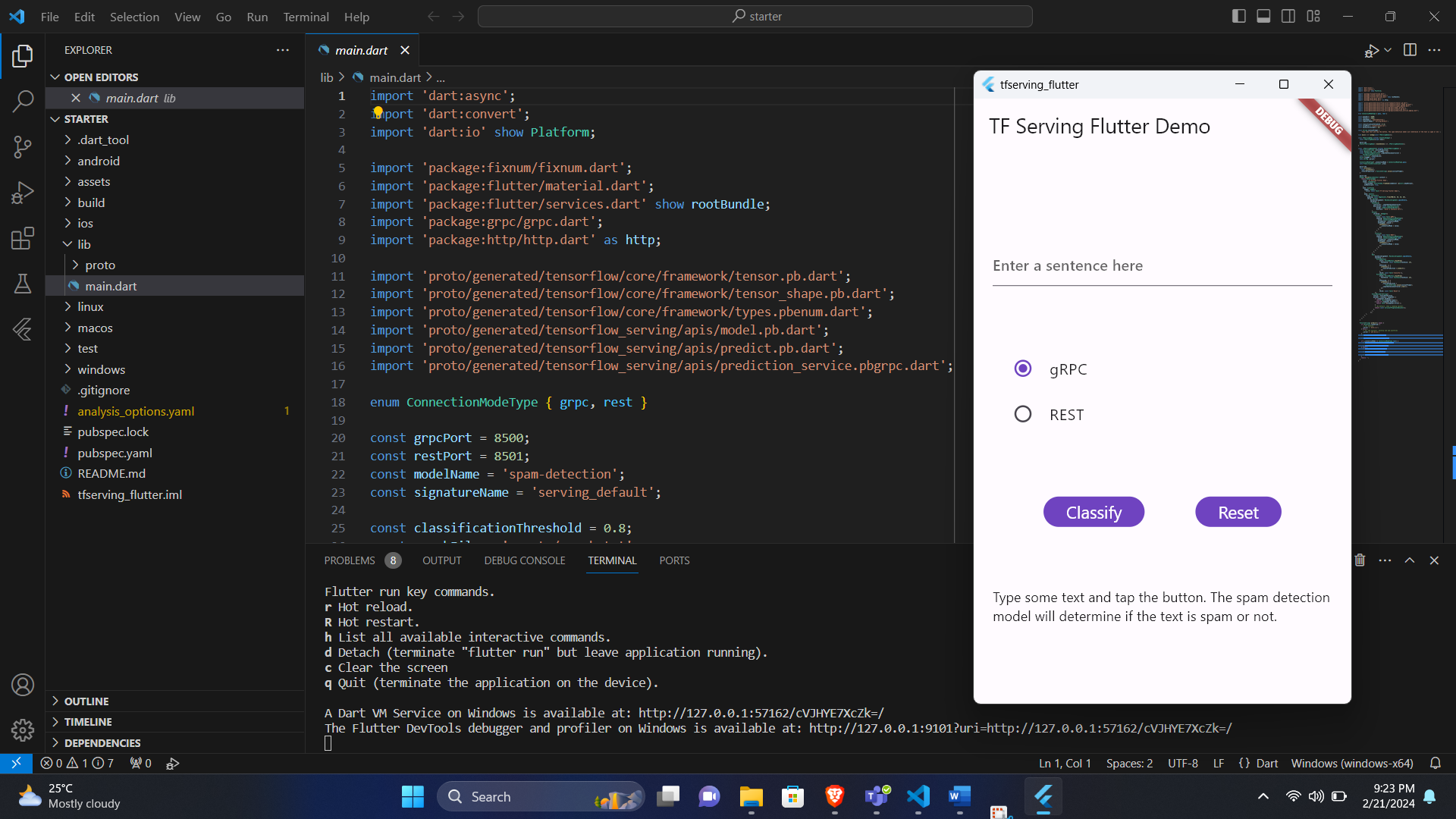
1. Retrieve the project dependencies by following these steps:

* Open Visual Studio Code (VS Code).
* Navigate to "File" > "Open Folder."
* Choose the starter folder from the previously downloaded source code. Run this command: flutter pub get



1. Execute and navigate through the application:

Once launched, the app will appear on your Android Emulator or iOS Simulator. The user interface is intuitive, featuring a text field allowing users to input text. Users have the option to transmit data to the backend using either REST or gRPC. The backend employs a TensorFlow model to conduct text classification on the processed input, subsequently relaying the classification outcome to the client app. The app, in turn, updates the UI to reflect the result.



[Deploy a text-classification model with TensorFlow Serving](https://developers.google.com/codelabs/classify-texts-flutter-tensorflow-serving#5)

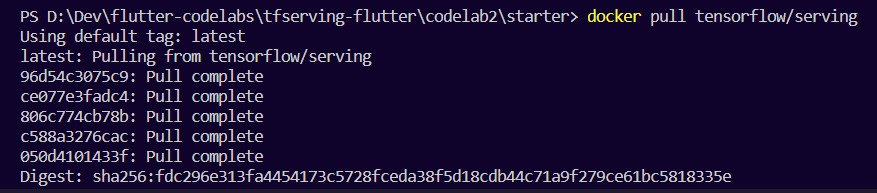
Text classification is a very common machine learning task that classifies texts into predefined categories.

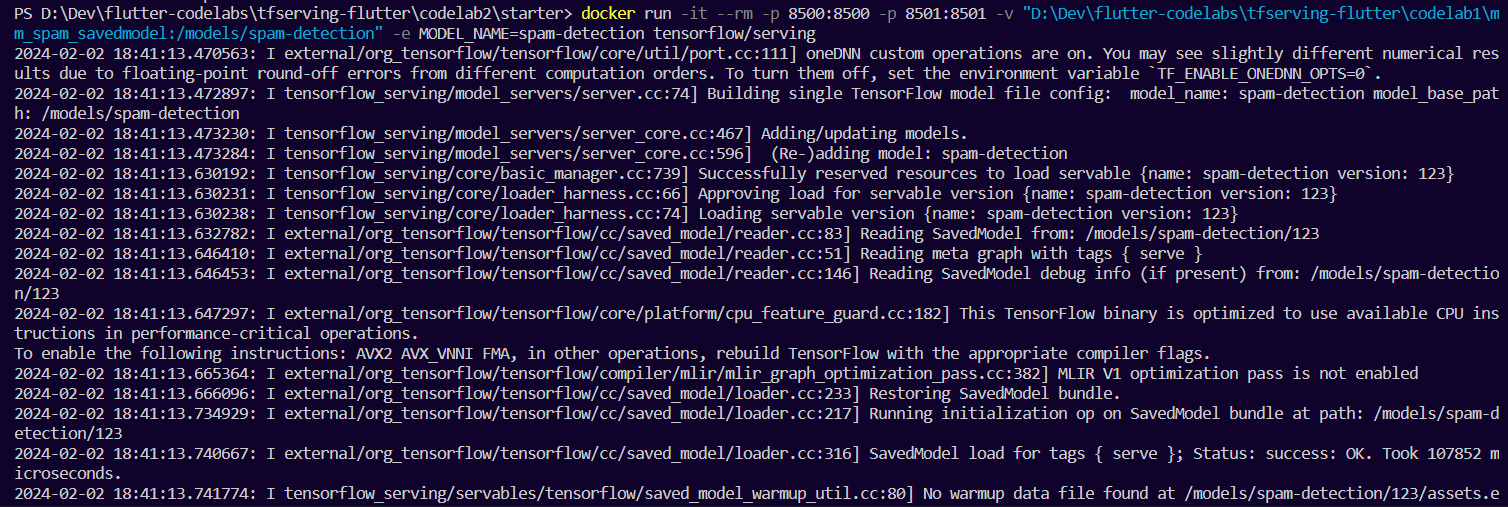
In this codelab, we will deploy a pretrained model from the [Train a comment-spam detection model with TensorFlow Lite Model Maker codelab](https://developers.google.com/codelabs/classify-text-tensorflow-serving) with TensorFlow Serving and call the backend from our Flutter frontend to classify the input text as ***spam* or *not spam***.

**Note:** A pretrained SavedModel along with the vocabulary and label files is provided in the*tfserving-flutter/codelab1/mm\_spam\_savedmodel*folder.

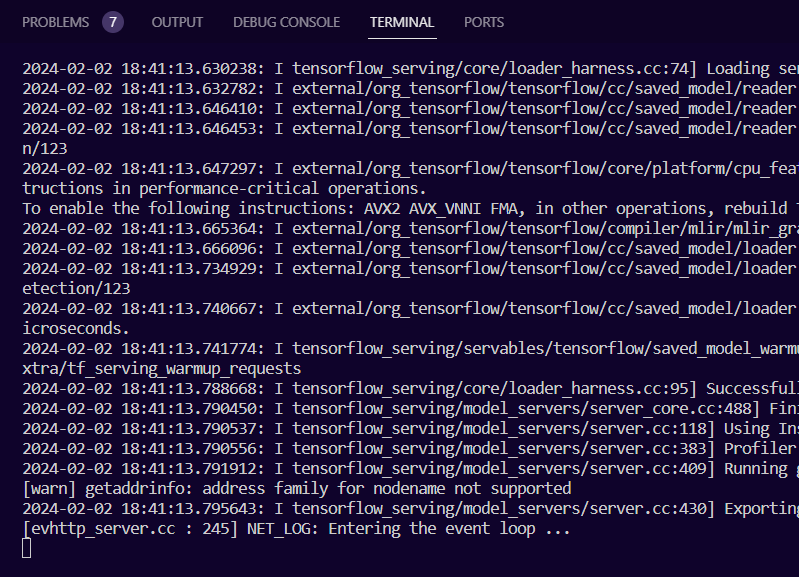
Start TensorFlow Serving

* In a terminal, start TensorFlow Serving with Docker, but replace the *PATH/TO/SAVEDMODEL* placeholder with the absolute path of the mm\_spam\_savedmodel folder on the computer.

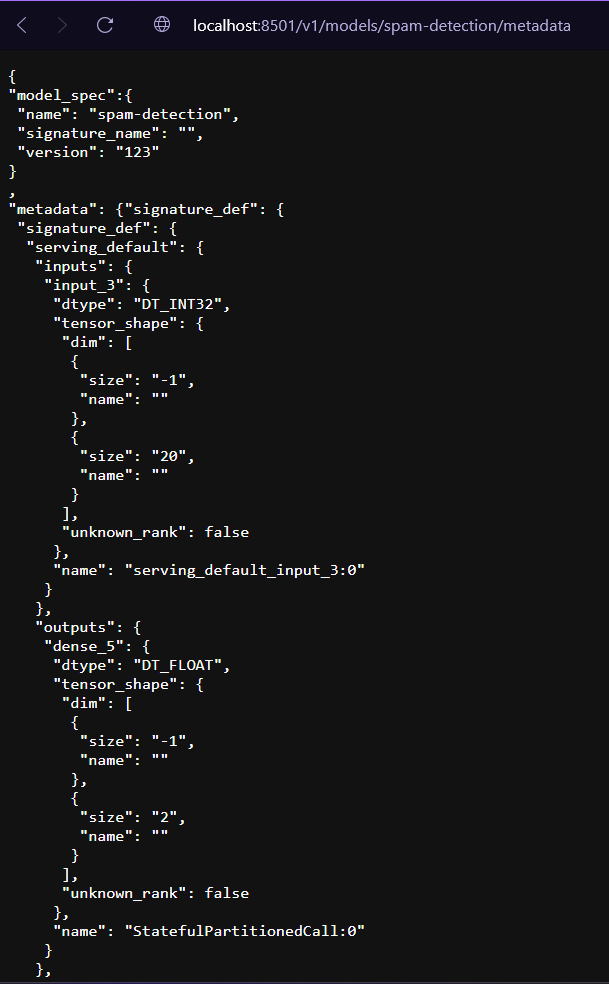




Docker automatically downloads the TensorFlow Serving image first, which takes a minute. Afterward, TensorFlow Serving should start. The log should look like this code snippet:



After starting the TensorFlow Serving docker image, we can visit *http://localhost:8501/v1/models/ spam-detection/metadata* to inspect the details of the input and output tensors.



[Tokenize input sentence](https://developers.google.com/codelabs/classify-texts-flutter-tensorflow-serving#6)

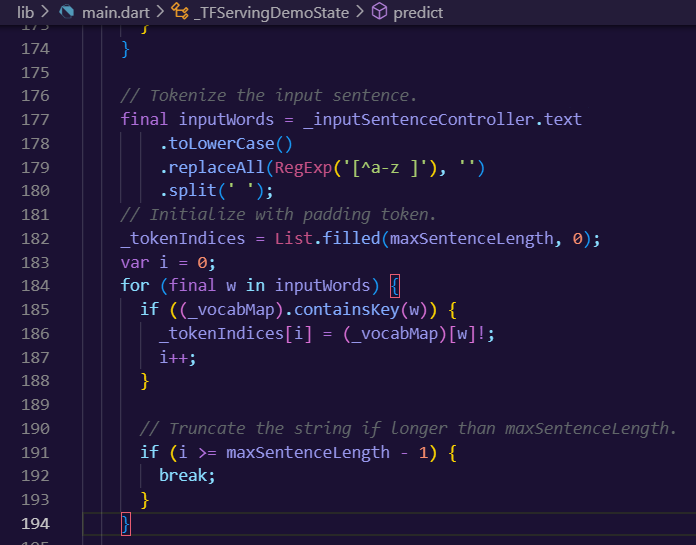
After preparing the backend, the next step is to tokenize the input sentence, enabling the transmission of client requests to TensorFlow Serving. By examining the input tensor of the model, it becomes apparent that it anticipates a list of 20 integer numbers instead of raw strings.

Tokenization involves mapping individual words entered in the app to a list of integers based on a predefined vocabulary dictionary before forwarding them to the backend for classification.

To implement this, add the following code to the **predict()** method in the **lib/main.dart** file to construct the **\_vocabMap** vocabulary dictionary.



Immediately after the previous code snippet, add this code to implement tokenization:



This code lowercases the sentence string, removes non-alphabet characters, and maps the words to 20 integer indices based on the vocabulary table.

[Connect the Flutter app with TensorFlow Serving through REST](https://developers.google.com/codelabs/classify-texts-flutter-tensorflow-serving#7)

There are two ways to send requests to TensorFlow Serving:

* REST
* gRPC

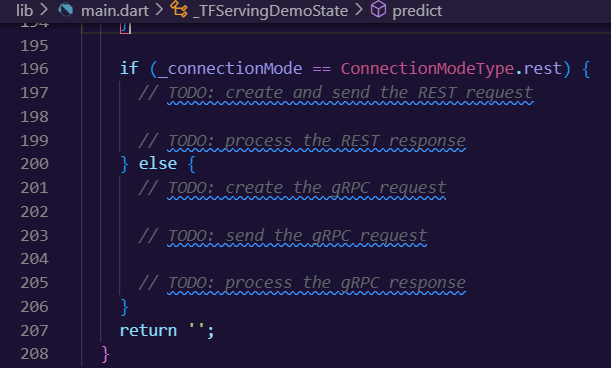
Send requests and receive responses through REST

There are three simple steps to send requests and receive responses through REST:

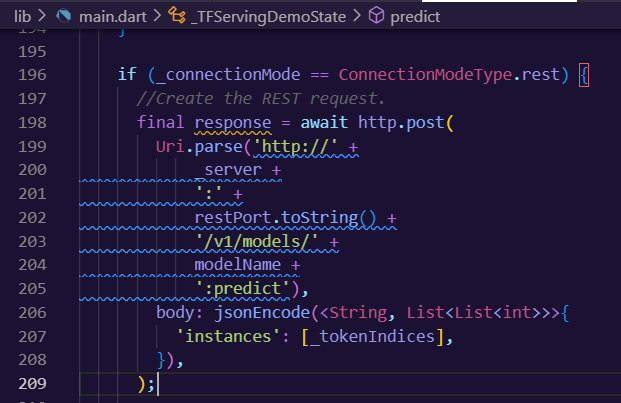
1. Create the REST request.
2. Send the REST request to TensorFlow Serving.
3. Extract the predicted result from the REST response and render the UI.

*Create and send the REST request to TensorFlow Serving*

1. Right now, the predict() function doesn't send the REST request to TensorFlow Serving. You need to implement the REST branch to create a REST request:

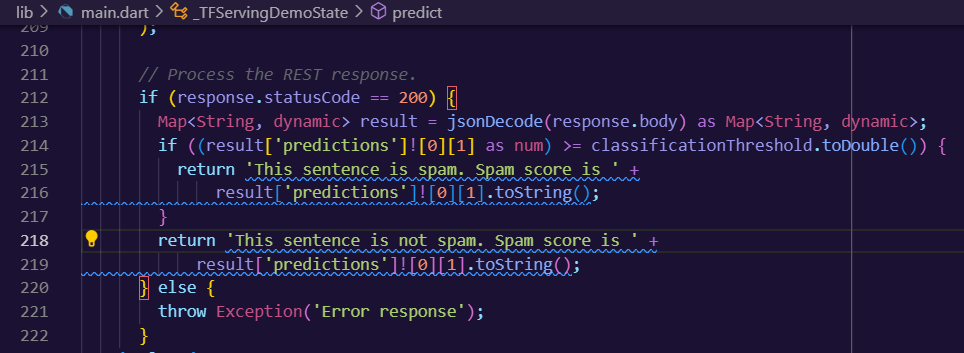


1. Add this code to the REST branch:



*Process the REST response from TensorFlow Serving*

* Add this code right after the previous code snippet to handle the REST response:



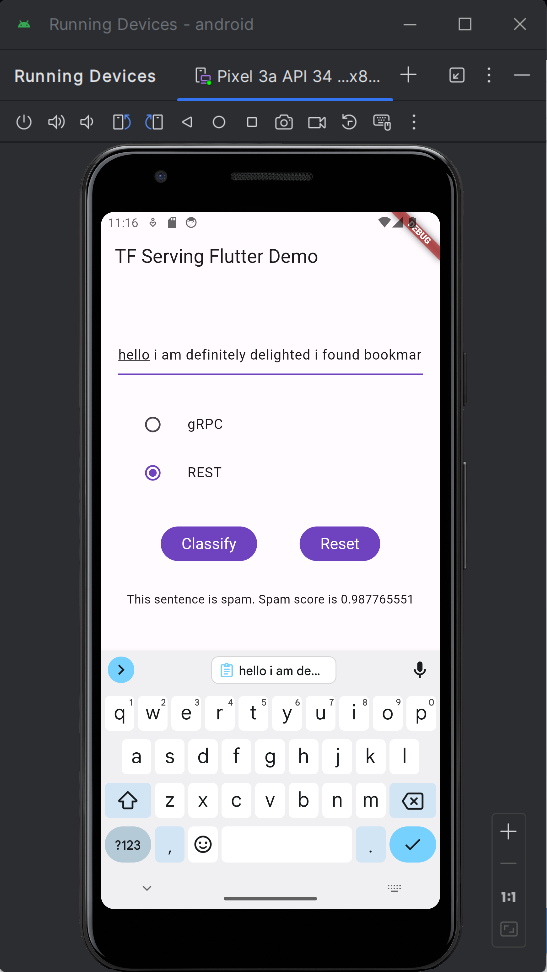
The postprocessing code extracts the probability that the input sentence is a spam message from the response and displays the classification result in the UI.

**Run it**

1. Type flutter run in a new terminal. Make sure docker is running on the same time.



1. Enter some text and then select **REST > Classify**.



**Note:** Depending on your training procedure and the input sentence, the model may predict a different spam score. For example, "hello i am definitely delighted i found bookmarking site www winbig com very profitable" is an example sentence that generally receives a high spam score.

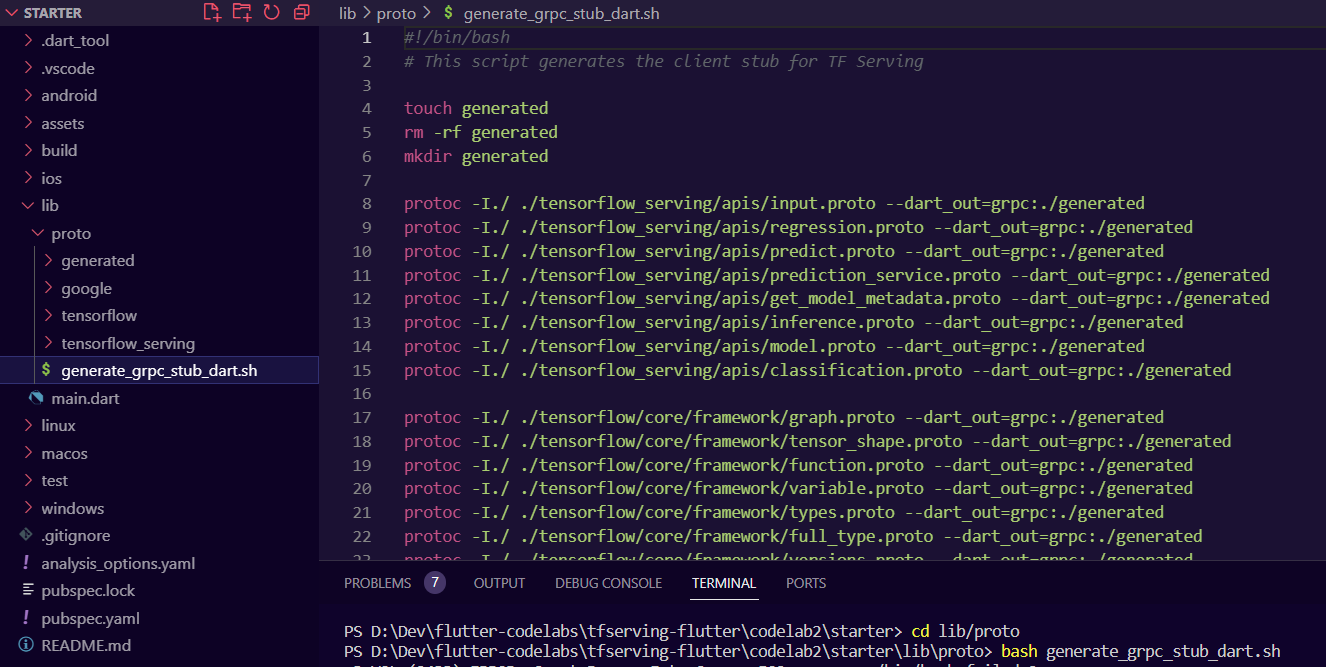
[Connect the Flutter app with TensorFlow Serving through gRPC](https://developers.google.com/codelabs/classify-texts-flutter-tensorflow-serving#8)

In addition to REST, TensorFlow Serving also supports [gRPC](https://grpc.io/).

gRPC is a modern, open source, high-performance Remote Procedure Call (RPC) framework that can run in any environment. It can efficiently connect services in, and across, data centers with pluggable support for load balancing, tracing, health checking, and authentication. It's been observed that gRPC is more performant than REST in practice.

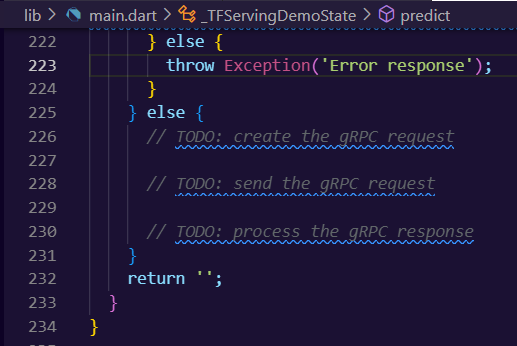
* In the terminal, navigate to the starter/lib/proto/ folder and generate the stub:

*bash generate\_grpc\_stub\_dart.sh*

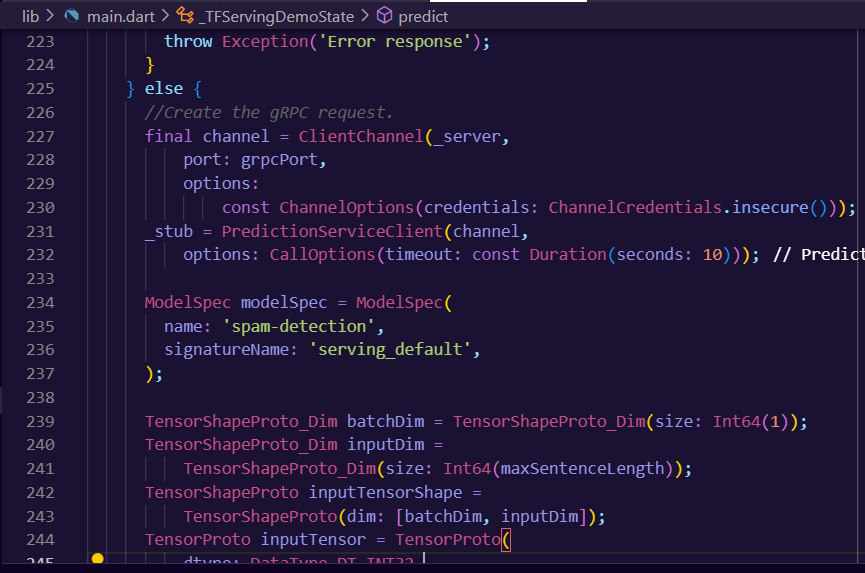


Create the gRPC request

Similar to the REST request, you create the gRPC request in the gRPC branch.



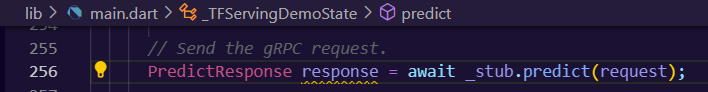
Add this code to create the gRPC request:



The input and output tensor names could differ from model to model, even if the model architectures are the same. Make sure to update them if you train your own model.

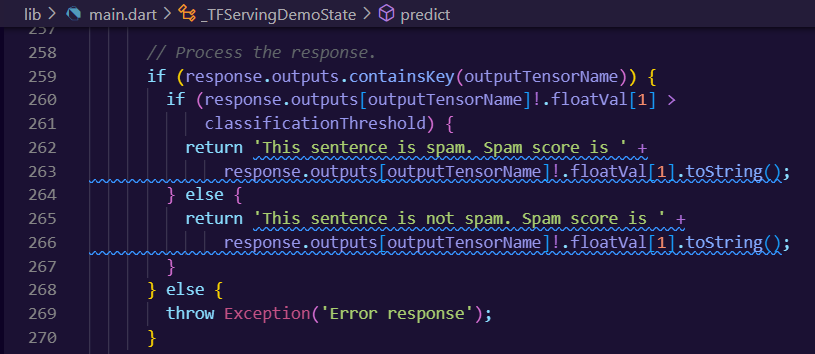
Send the gRPC request to TensorFlow Serving

Add this code after the previous code snippet to send the gRPC request to TensorFlow Serving:



Process the gRPC response from TensorFlow Serving

Add this code after the previous code snippet to implement the callback functions to handle the response:



Now the postprocessing code extracts the classification result from the response and displays it in the UI.

**Run it**

1. Type flutter run in a new terminal. Make sure docker is running on the same time.
2. Enter some text and then select **gRPC > Classify**.

