LLMs in Flutter

1. Introduction

Large Language Models (LLMs) are actively supported in Flutter through a plethora of plugins. Since most LLMs are very large models, the common practice is to access them from your app via API calls. In this respect, all LLM vendors (OpenAI ChatGPT & DALL-E, Google Gemini, Meta LLama, Mistral AI) require that you sign up for their service in order to obtain an API key to use the respective service. However, most API keys come with limited or no free-tier usage and most importantly, they require credit-card signup. A modest exception to the above rule is Google and its Gemini LLM.

2. Google Gemini

You may obtain a Gemini API Key that is not directly linked to a Google Billing Account and can be used up until it exhausts free tier resources. If asked, select the *Develop in your own environment option* and accept license terms. When free tier resources are exhausted, you will be getting an 429: RESOURCE_EXHAUSTED API Response.

After you create a new Flutter Application project in Visual Studio Code, you may incorporate the API key you have generated previously by creating a new file (**api-keys.json**) in the root folder of your project, with the following contents

```
{
        "GOOGLE_AI_KEY": "_your_api_key_here"
}
```

Then, you need to instruct the Visual Studio Code debugger to read your key when it launches. To do so, create a folder **.vscode** in the root folder of your project and place **launch.json** in it, with the following contents.

```
{
  "version": "0.2.0",
  "configurations": [
        {
        "name": "Launch",
        "request": "launch",
        "type": "dart",
        "program": "lib/main.dart",
        "args": [
        "--dart-define-from-file",
```

```
        "api-keys.json"
        ]
        }
  ]
}
```

The google_generative_ai plugin allows the incorporation of Google Gemini into your flutter project (via API calls to the model hosted on Google servers). In order to load it into your project, you need to add the following two lines under *dependencies* in **pubspec.yaml**

```
google_generative_ai: any
flutter_markdown: ^0.7.4+2
```

We will be reviewing two examples. In the first one, we will be implementing chatbox functionality right into our flutter application. Our code is based on the flutter example based on google gemini's GitHub repository.The supplementary file **gemini_demo.zip** contains **main.dart** which should replace the same-named file in the lib/ folder of your flutter project.

In the second example, we will use the Google Gemini LLM in order to enhance user experience (UX). In this respect, we will extend the 2D2D functionality, using an LLM for creating tasks. We will mostly follow the example code available at flutter's GitHub repository.

We will create a new flutter application using Visual Studio Code and add **api-keys.json** and **.vscode/launch.json** as before. We will add the following three lines under dependencies in **pubspec.yaml**.

```
google_generative_ai: any
flutter_markdown: ^0.7.4+2
url_launcher: ^6.2.6
```

Finally, we will upgrade gradle to version 8.4 by following the next steps:

a. Set *distributionUrl* to *https\://services.gradle.org/distributions/gradle-8.4-all.zip* in **android/gradle/wrapper/gradle-wrapper.properties**
b. Set *id "com.android.application"* version to *8.3.2* in **android/settings.gradle**
c. Set *ndkVersion = "25.1.8937393"* in **android/app/build.gradle**

The supplementary file **tasks_demo.zip** contains all the necessary code for this example, which should be placed into the **lib/** folder of your flutter application. More specifically, it contains:

1. **main.dart**: Loads the main application widget, which is the *GenerativeAISample* class and implements all functionality.
2. **widgets/message_widget.dart**: Widget to display user requests and system responses on screen

3. **text_field_decoration.dart**: Contains a function that decorates text that appears in the chat screen