

软工实验班实践 | GPT 软件开发

引言

这是软件工程实验班中对于“ChatGPT 能否真正替代人类 / 使得普通码农成为超级码农（无需相关行业知识，依靠 GPT 即可担任完成复杂、学科交叉性任务）”探索的第三部分，利用 GPT 的软件分析（OOA & OOD）结果，生成代码。

探索内容

笔者感觉，这部分是比较难得部分，最后结果还算令人满意。

主要内容：让 GPT 在人工辅助下生成一个 APP（前端 Flutter，后端 Django），然后将前面 Kaggle 比赛中训练的模型融入 APP 中（作为主要功能），最后将 GPT Agent 的能力加入 APP 中（待完成，该课程最后一部分内容）。

结果代码开源

[Kingsley-Yoimiya/EcoEye: 软工实验班 项目 植保通 \(github.com\)](https://github.com/Kingsley-Yoimiya/EcoEye)

实现思路

最开始的思路——直接让 ChatGPT 上手

生成方式

- 直接告诉 ChatGPT 软件的架构，OOD 图，强制让 GPT 遵守这些内容。
- 然后让 GPT 从头开始进行文件的生成。

效果

- 肉眼可见效果并不好。
- GPT 生成代码需要很多已经写过的或者未写过的代码的信息，而这些信息因为记忆的限制或多或少有所缺失。
- 因此 GPT 生成的内容直接出现了编译上的很大问题，出现了很多编译错误。
- 同时导致笔者非常迷惑，各种问题，各种代码，直接选择放弃。

认为最好的思路——两遍生成

生成方式 & 效果

- 先根据 OOD 和架构设计让 GPT 生成文件夹的大致情况。

- 然后用以下 prompt (后面提供 OOD 部分省略了)

你是一名软件开发师，你要根据我提供的软件架构、PlantUML 格式的类图、顺序图等等，进行软件的开发。

这是目前你应该有的文件夹结构：

前端 (Flutter)

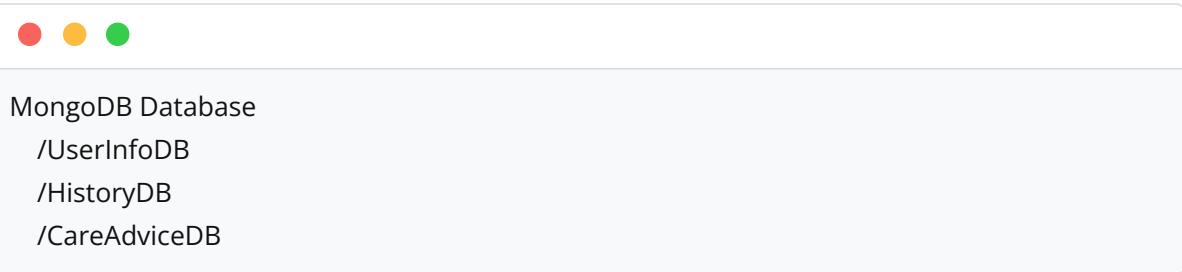
```
/flutter_app/
  /lib/
    /screens/
      main_screen.dart
      registration_screen.dart
      login_screen.dart
      upload_screen.dart
      result_display_screen.dart
      advice_display_screen.dart
      history_record_screen.dart
      share_screen.dart
    /controllers/
      user_management_controller.dart
      history_controller.dart
      upload_and_analysis_controller.dart
      share_controller.dart
    /models/
      user.dart
      record.dart
      advice.dart
    /widgets/
      input_field.dart
      button.dart
      photo_selector.dart
      result_graph.dart
    /services/
      api_service.dart
    /utils/
      constants.dart
      validators.dart
    main.dart
```

后端 (Django REST Framework)

```
/django_backend/
/app/
/settings.py
/urls.py
/wsgi.py
/asgi.py
/authentication/
/models/
    user.py
/views/
    login_view.py
    registration_view.py
/serializers/
    user_serializer.py
/urls.py
/history/
/models/
    record.py
    advice.py
/views/
    history_view.py
    advice_view.py
/serializers/
    record_serializer.py
    advice_serializer.py
/urls.py
/upload/
/views/
    upload_view.py
    analysis_view.py
/urls.py
/share/
/views/
    share_view.py
/urls.py
/manage.py
```

数据库 (MongoDB)

对于MongoDB，不需要一个具体的文件结构，因为它是非关系型的，我们主要通过后端代码定义数据模型。但是，为了概念上的组织，我们可以想象以下的集合结构：



- 接着 GPT 会进行询问，自动从头开始一个一个文件进行生成，这时候的文件情况基本上没有太大的编译方面的问题了，如果出现了，直接将报错信息丢给 GPT 几乎都能分析出原因。
- 但是这个时候的界面有点简陋，而且前端的功能没有衔接。
- 在接下来就是人工根据返回的信息，不断丢给 GPT（同时附上目前需要修改文件的代码和参考代码，通常是两份文件，最多是三份~四份文件）
- 这个时候需要自己进行代码大致框架理解或者让 GPT 进行解释，接着按照自己的需求——定位负责的代码，告诉 GPT 需要的效果，然后 GPT 进行修改。见感想-GPT 代码生成工作方式部分。

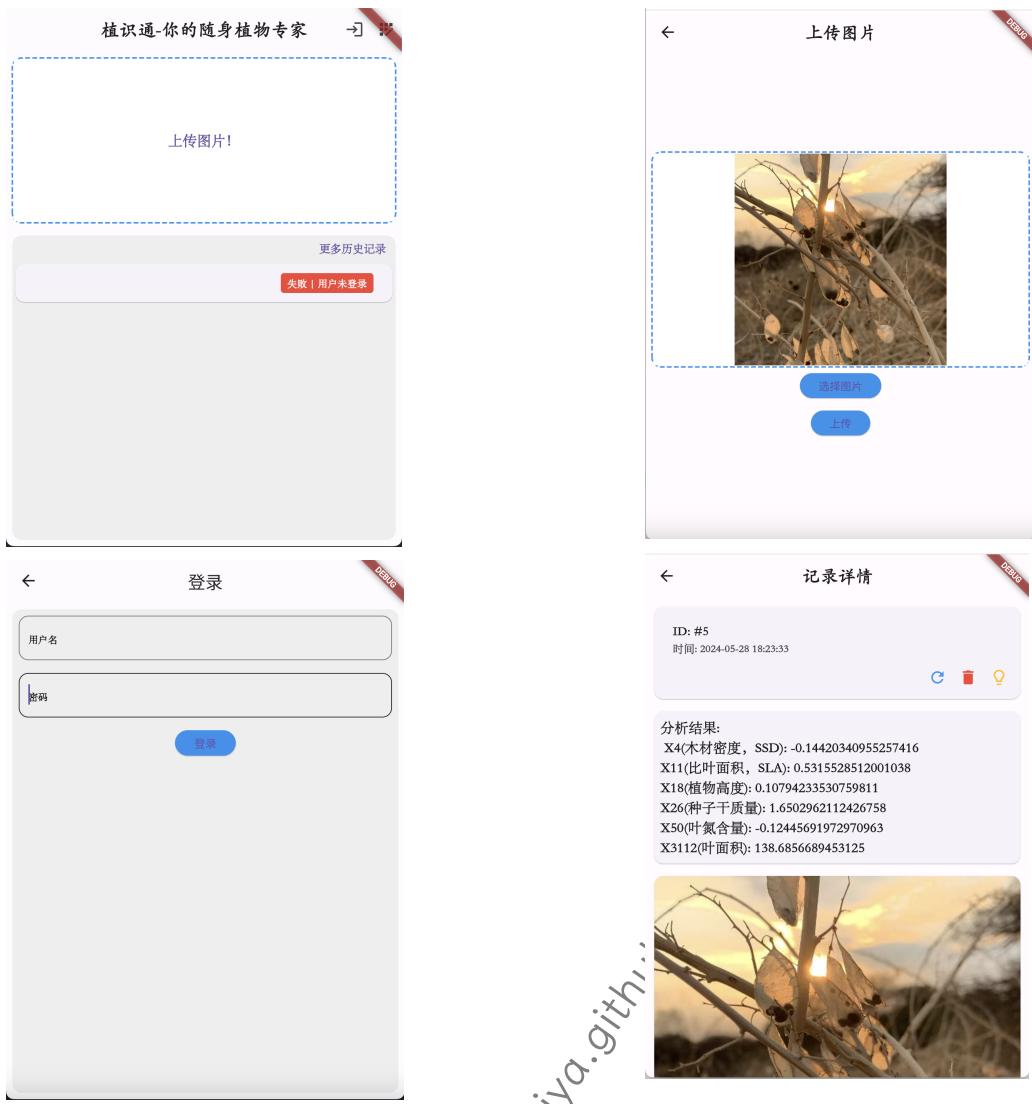
最终效果

生成代码长度

Language	files	blank	comment	code
Dart	23	119	23	1606
XML	22	5	50	845
Python	33	223	172	713
C++	8	121	62	443
CMake	7	82	110	346
JSON	4	0	0	248
C/C++ Header	8	51	65	103
Gradle	3	17	5	89
Windows Resource File	1	23	29	69
Swift	6	14	7	54
HTML	1	8	13	38
YAML	2	16	82	38
Markdown	2	8	0	13
Properties	2	0	0	8
Kotlin	1	2	0	3
SUM:	123	689	618	4616

注意，可以只考虑 Dart、Python 的代码长度，大概 2300 行代码。

应用概况



人类参与情况

- 几乎是提问。
- 大概最多 3% 的代码是人工写完的。
- 耗时：未统计，20 个小时应该是有的。

感想

GPT 代码生成工作方式

- 笔者认为，人类在GPT代码生成中，更像一个懂技术的产品经理的角色。（懂技术，但是只懂大概，不会全貌）
- 而前面的强行使用 OOA / OOD 方式不太能获得一个好的代码，要实现一个好的应用，OOA / OOD 最多只能作 GPT 参考，真正的贯彻实行必须人类指导，而 OOA / OOD 的作用就是帮助人类理清思路。
- 最后的工作方式就是：实时地运行项目，然后人类找出不满意的地方，告诉 GPT，同时人类手动定位相关代码，作为辅助信息给 GPT，如果遇到了 BUG，就将报错信息给 GPT，GPT 通常会快速解决，遇到难搞的 BUG 就需要人类少量理解（可以问 GPT），然后根据 GPT 信息手动 / 给出修改思路。

有了 GPT，人人都能做出应用？

- 笔者认为，GPT 的能力上来看，这个还是有点难，但是要想做出应用，对于人的综合素质，尤其是毅力、提问能力有很大考验，而人类的已有计算机相关知识，可以减少相关的要求水平。
- 提出该结论的原因是因为：这个项目，前端是 Flutter（笔者完全不会），后端是 Django（Python 会，但是这个不了解），而笔者一步一步提问后感触最深的就是耐心、细心、洞察力以及信心，中间出现的有的问题，其解决办法如果不是笔者一路拷打，可能会变得无法解决。
- 就拿最后将 Kaggle 模型集成到应用中为例，这里出现了 Python 环境配置问题，具体情况可以[看这里（但是只有前半部分，聊天记录太多找不到了）](#)。问题主要出在了模型的权重是 Kaggle Notebook 上的，而本地（macOS）环境出现不一致，于是一致在找解决办法，这里一直对话了很多轮，当时 ddl 接近，还没有好的结果（云端重新训练有点慢），最后一次询问中，笔者强行让 GPT 令本地机器安装上 Kaggle 远程环境，还是有问题，这时笔者提出了一个很好地角度“能不能忽略冲突限制，强制安装”，然后成功解决了这个问题。
- 从这里可以看出，GPT 会如实回答问题，但是在这种地方，还是需要人的作用的，如果笔者没有灵机一动，给出了一个好的角度，那么不会有如期完成 DDL 的展示。