团队第二篇报告

1.源代码操作

本次练习旨在掌握源代码管理的基本操作，包括在github上进行源代码的迁入、代码的复审、代码的合并，因为是第一次操作，所以上传github的源代码极为简单，只是为了熟悉流程操作而使用，下面是我的本次体验。

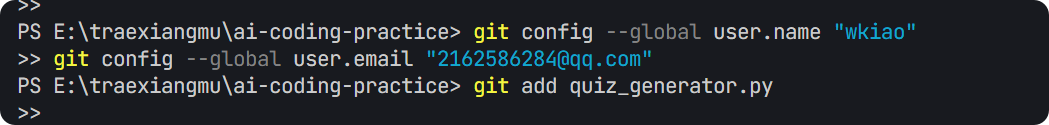
首先在本地新建项目文件夹，使用 git init 初始化仓库。

然后我配置了github的用户信息：

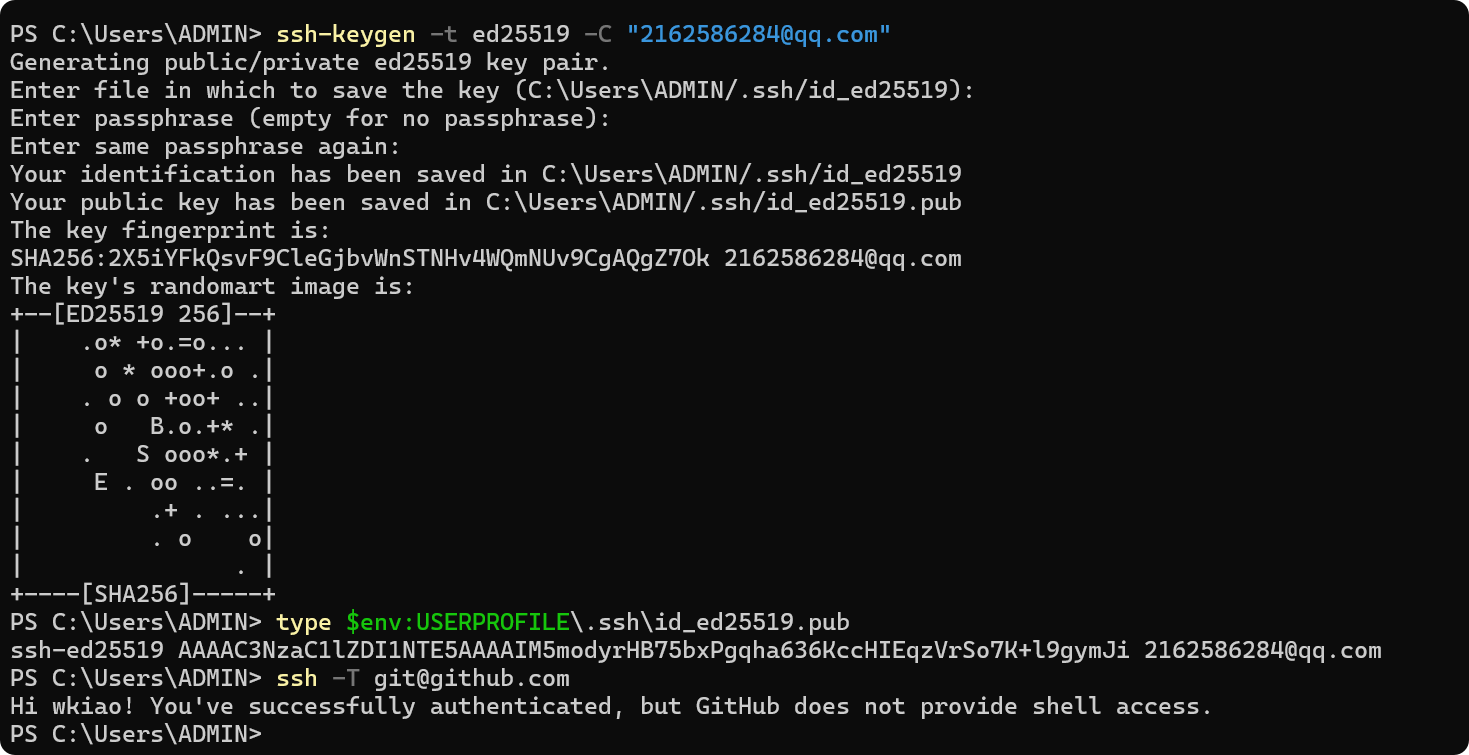
git config --global user.name "wkiao"

git config --global user.email "2162586284@qq.com"

截图如下：



然后将项目关联到 GitHub 远程仓库，通过 SSH 连接配置公钥，解决权限问题。



操作也很简单，直接把生成的一串长密钥复制到[https://github.com/settings/ssh/new](https://github.com/settings/ssh/new" \t "_new)页面下的key即可，然后title随便起一个（这里我随便起了一个mykey），然后点击绿色按钮 Add SSH key。最后我测试一下现实最后一行的内容，说明成功了。

最后，我编写一个很简单的示例代码：quiz\_generator.py，并进行了提交。

git add quiz\_generator.py

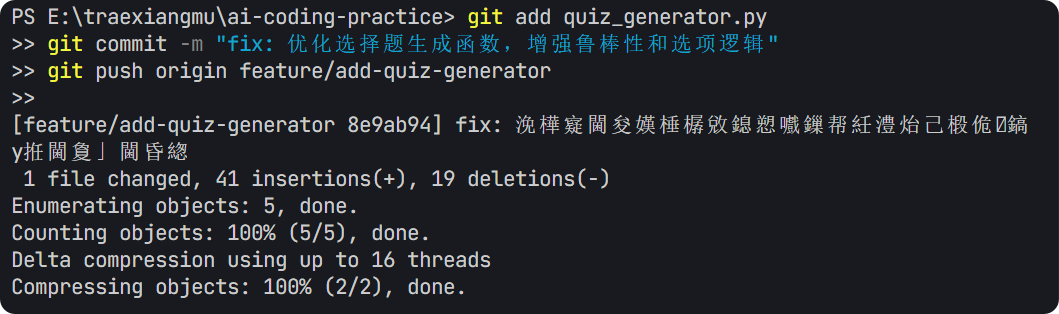
git commit -m "feat: 添加选择题生成函数"

git push -u origin main

然后进行分支管理以及功能开发：

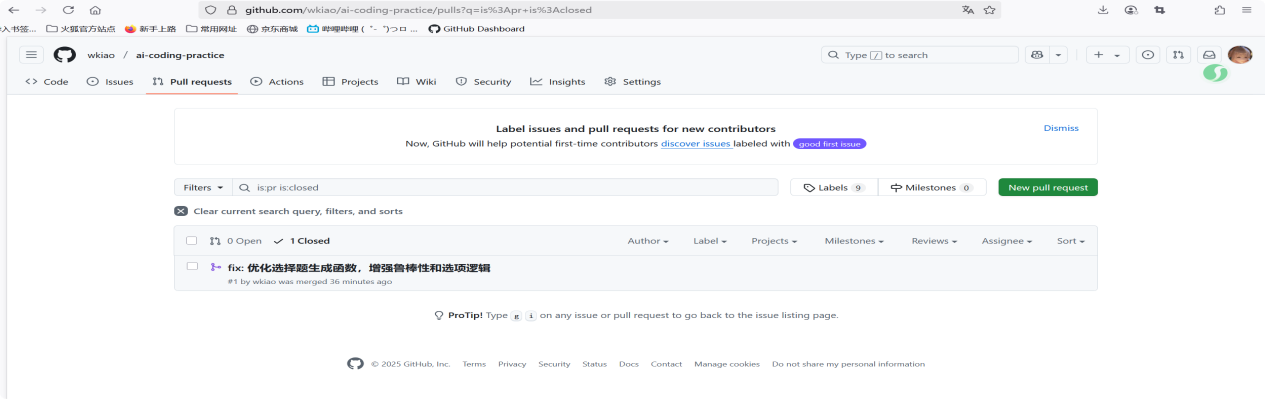
局部截取_20250709_180456

在新分支修改代码，优化选择题生成函数，并提交：

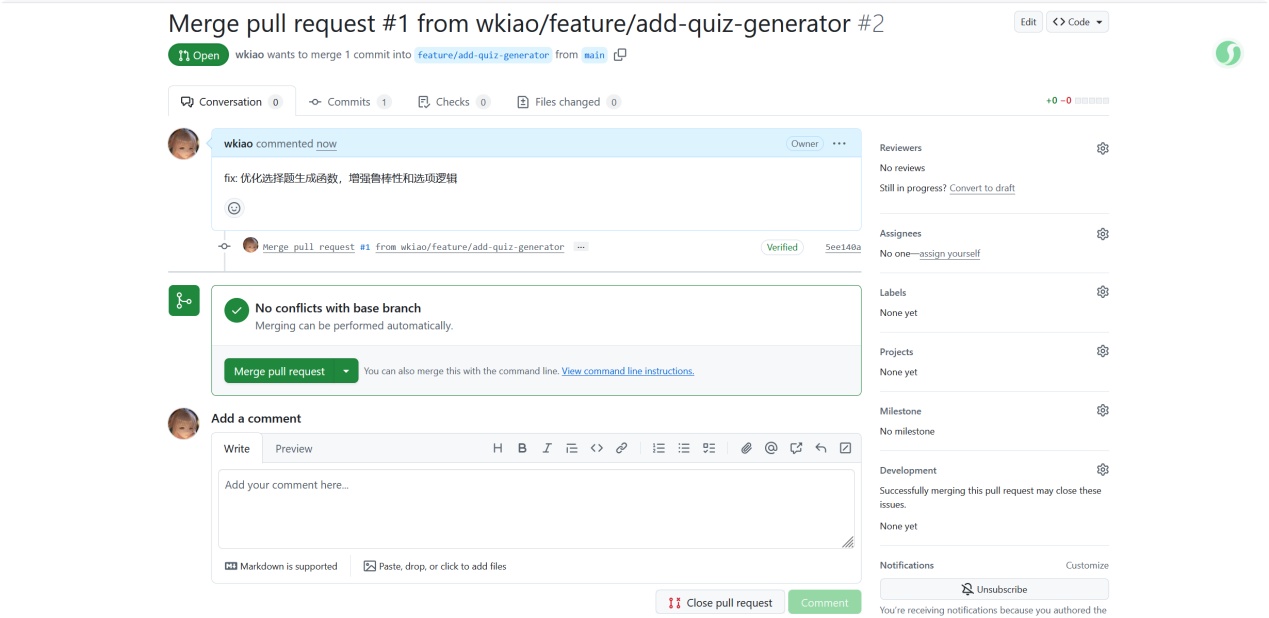


这里提交了好几次，这次应该是提交成功了。

然后通过 GitHub 发起 Pull Request，描述代码改动并邀请同伴复审。



从截图（是一整串流程完成后截的）可以看到，我们已经有可以一个closed的PR页面，我们可以在这里进行一个New PR的建立。也可以查看代码差异，进行代码复审。

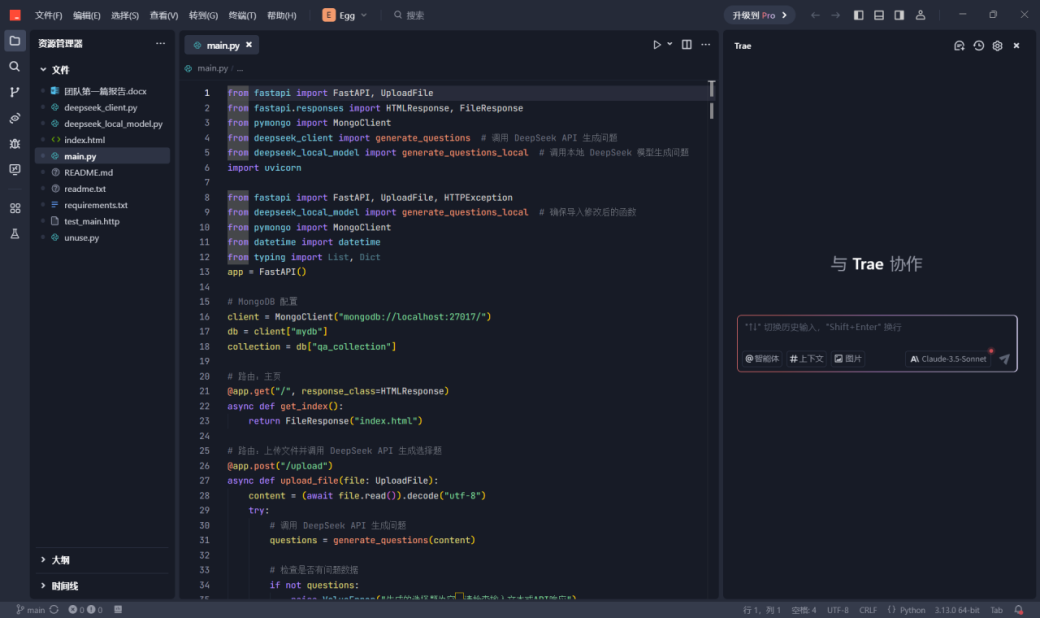


我重新建立了一个PR，可以看到，我们能在这里进行评论复审，发表自己的见解，也可以进行合并，点击merge PR，然后confirm merge即可。在完成之后也可以点击删除分支。

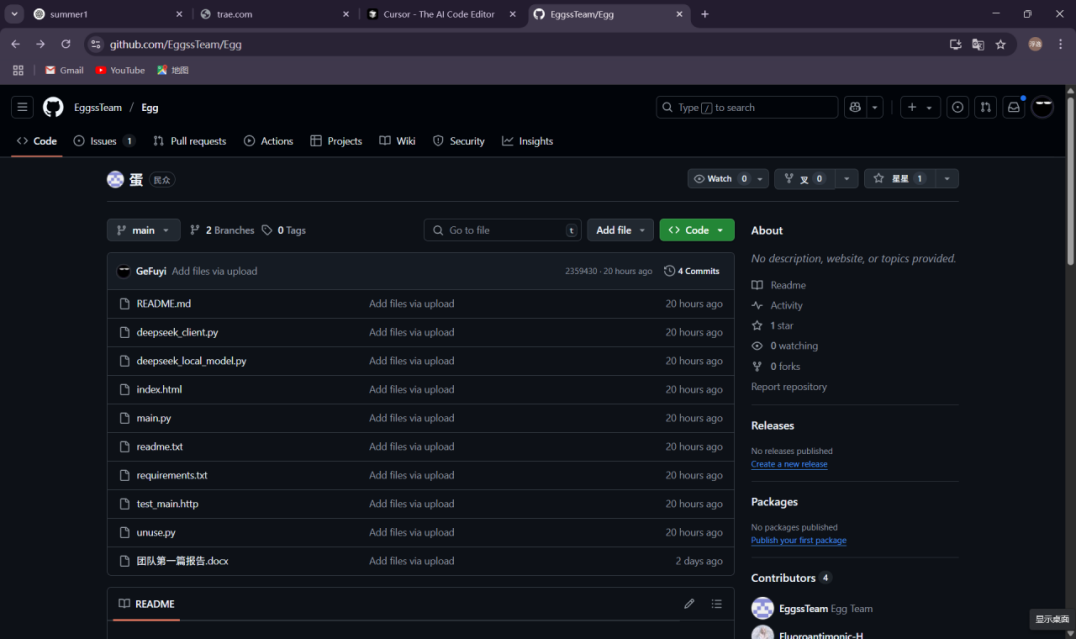
随笔：遇到的困难与如何解决

遇到了github的权限错误，我在第一次直接进行代码的提交上传时，遇到了这个错误fatal: unable to access 'https://github.com/wkiao/ai-coding-practice.git/': Recv failure: Connection was reset，即没有连接到github服务器，上网查询后发现是因为网络限制，需要一个ssh推送方式来绕开网络限制，于是便在终端进行了ssh的配置，之后这个问题便没有出现了。

之后我们团队也进行了第一个项目代码的迁入，如下图所示：



在github里面如下显示：



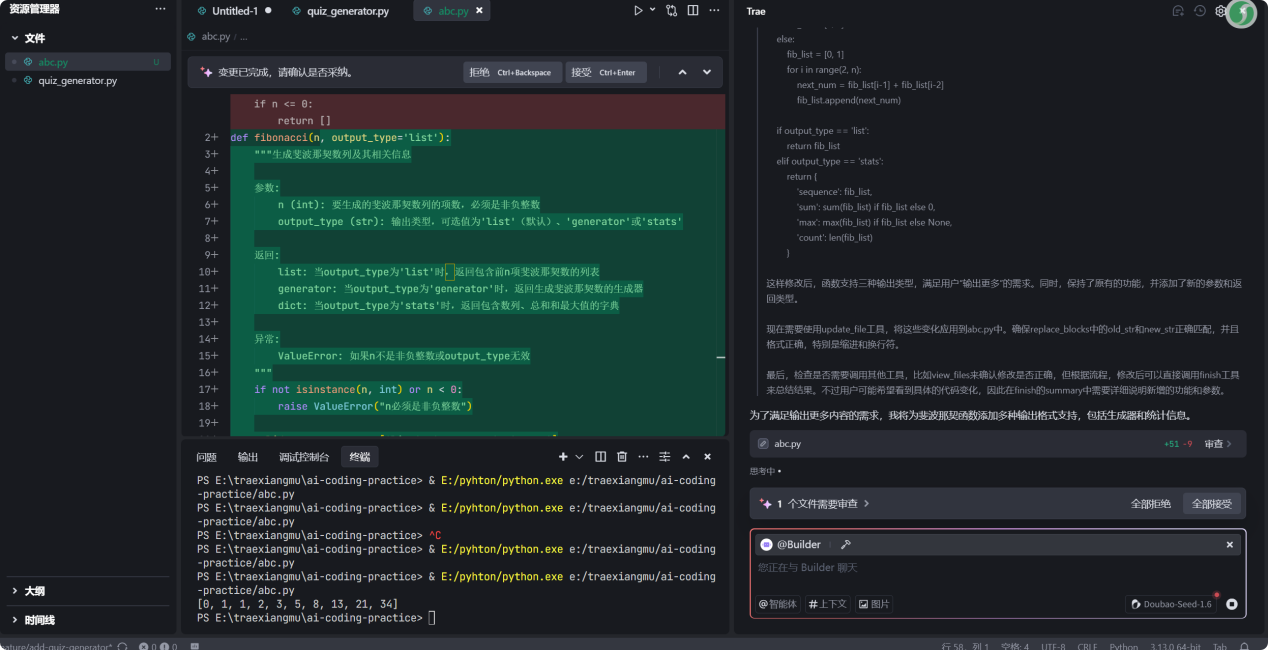
1. ai-codeing工具使用体验

团队成员：王开跃

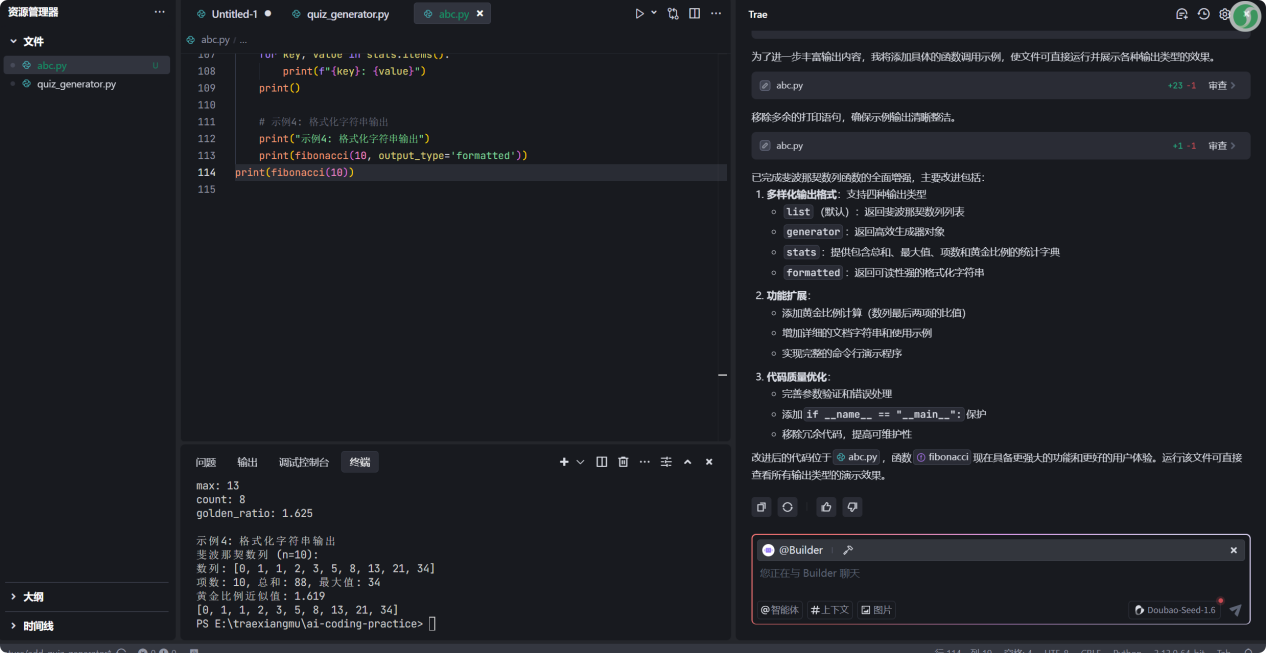
选用工具：trae VN

我使用的是trae VN进行ai-codeing体验，操作极其简单，首先在浏览器下载trae，然后进行用户登录即可，我首先选用的是手机号登陆，后面发现操作起来不是很方便，和github连接起来比较麻烦，所以后来我选择了github账号进行连接。

使用过程：我在这个软件进行了比较简单的ai-codeing体验，第一个体验就是上面quiz\_generator.py，是一个比较简单的输出问题的小代码，我给了ai一个简单的命令，他就帮我输出了这段代码，并且可以直接投入使用。第二次体验就是我让他生成了一段斐波那契数列的代码，我给了他一段注释：（# 生成前 n 项斐波那契数列，）他就帮我生成了一个很简单的代码段，然后我在小助手里面让他精进代码，他很快就给了我思考过程并且进行了修改，截图如下：



然后我点击变更，他就增加了很多新功能，截图如下：



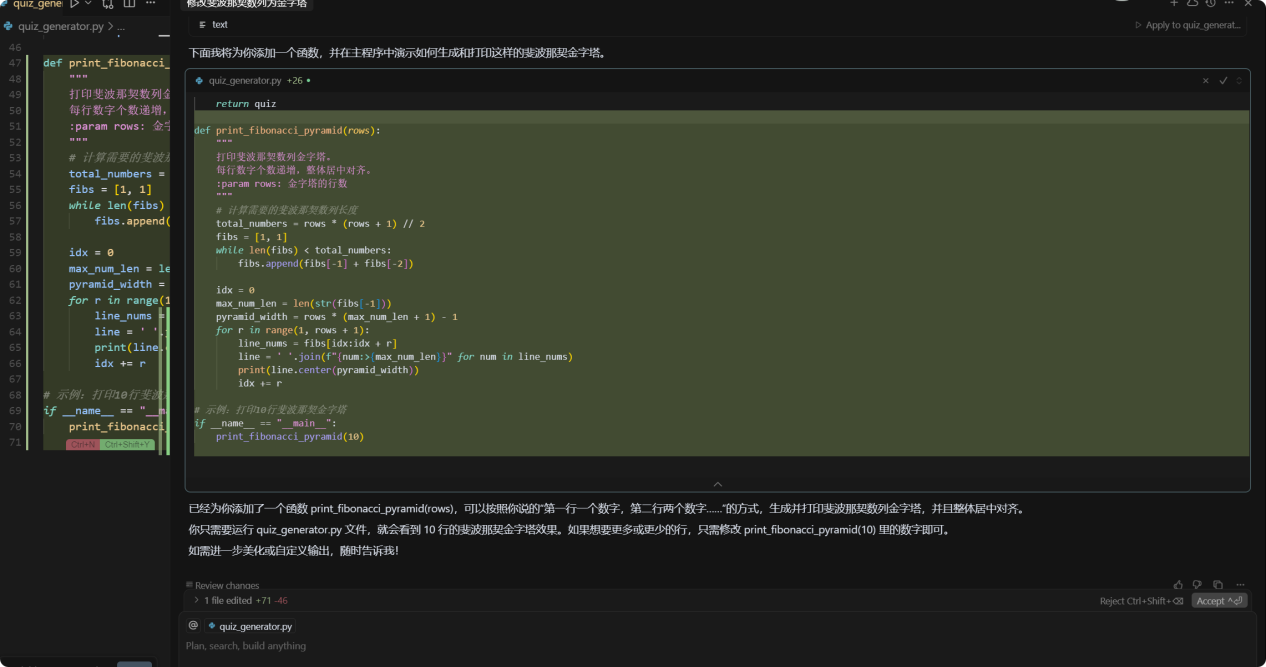
包括什么多个示例与更加细致的输出，总体上来说感觉还是很不错的。

一些个人体验感受：这个trae用起来还是比较顺手的，使用起来操作非常简单，ai的功能也比较强大，你给了ai一段注释，他就可以根据这段注释进行代码的编写，生成的很快并且精准度也可以，也可以根据你的代码进行自动补全，进行一些函数的优化，提升了自己的代码编写速度，而且生成的代码结构也比较清晰，容易理解；但是缺点也是很明显，首先生成的代码有可能不按照你的想法，比如说我只让他精进一下数列，但是他擅自给我增加了很多新的功能，比方说数列的个数，总和等，增大了我的审核压力，另外，对于复杂代码，他的能力就比较有限了，他会思考很久然后进行尝试，但是最终给出的结果并不总是准确。

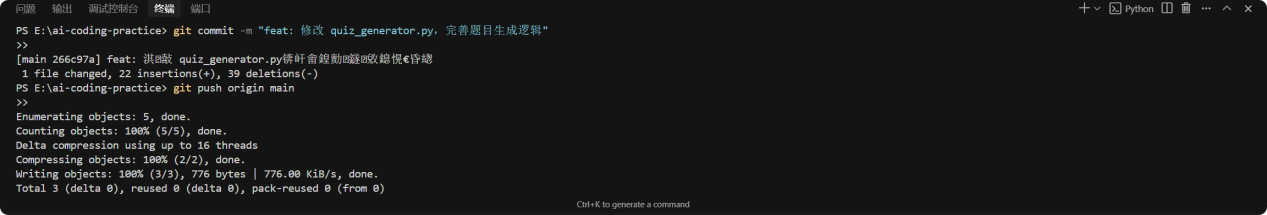
选用工具：cursor编辑器

这个软件的用法也比较简单，我用谷歌账户登陆之后，它提示我未连接github，连接操作非常简单，他给你一个密钥你直接在github的对应网址输入就行了。

使用过程我觉得和trae基本没有区别。都是你生成一段代码放上去，ai帮你修改，我认为总体上是大差不差的。他在听不懂你的要求时会继续思考并发出提问知道你给他解释明白，这点我觉得还是很不错的。截图如下：



然后把他上传到github：



感觉这个ai用起来比之前那个顺手一点，至少他对我的要求比较重视，不会做无谓的修改，这是我比较喜欢的。

团队成员：吴志超

AI编程工具实践报告：Copilot与Cursor使用体验

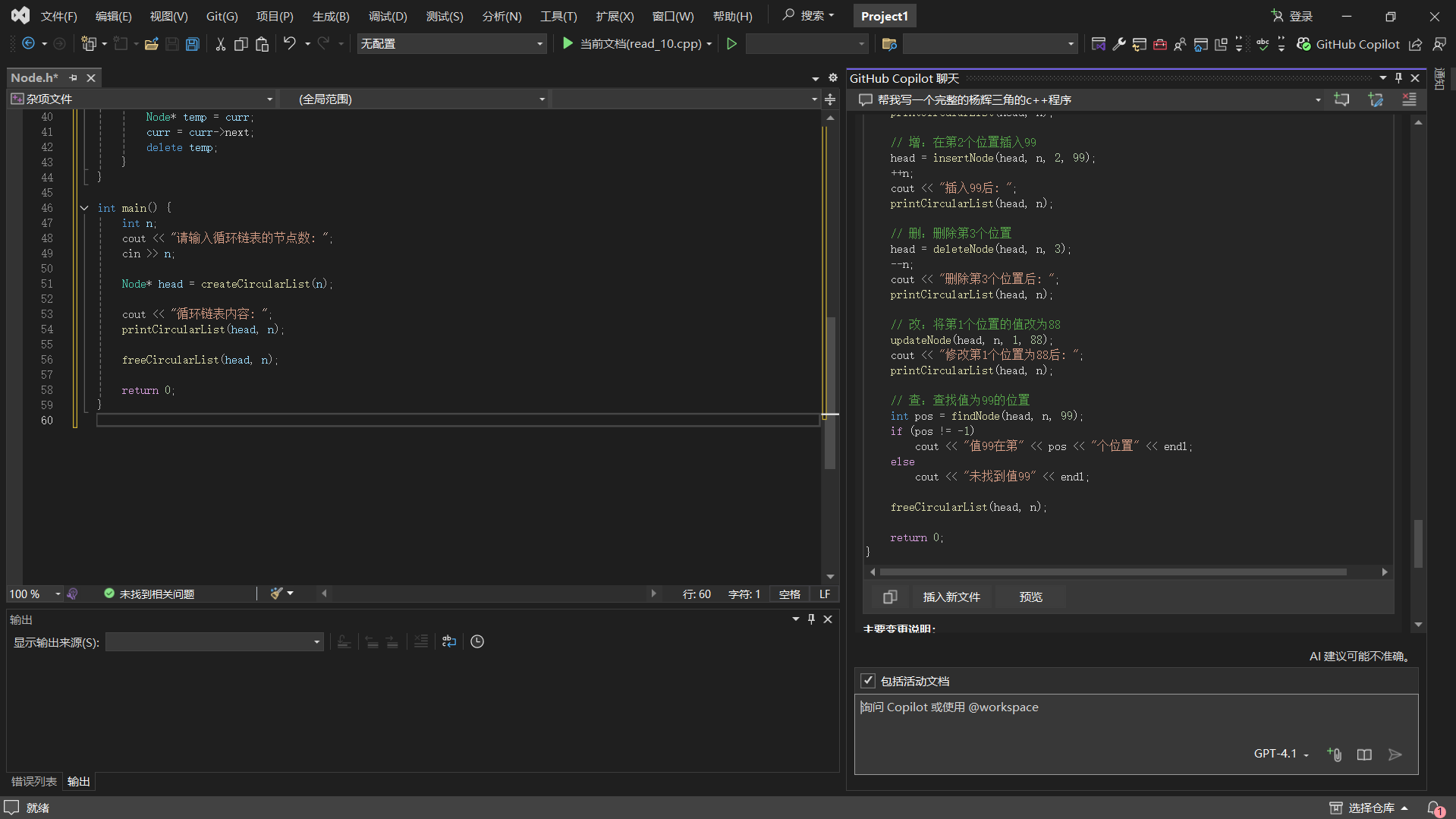
一、Visual Studio Copilot 使用体验

1. 杨辉三角代码生成

通过Copilot成功生成了杨辉三角的Python实现代码。整个过程非常流畅，只需输入简单的注释说明，AI就能理解需求并生成完整可运行的代码。

2. 循环链表实现与功能扩展

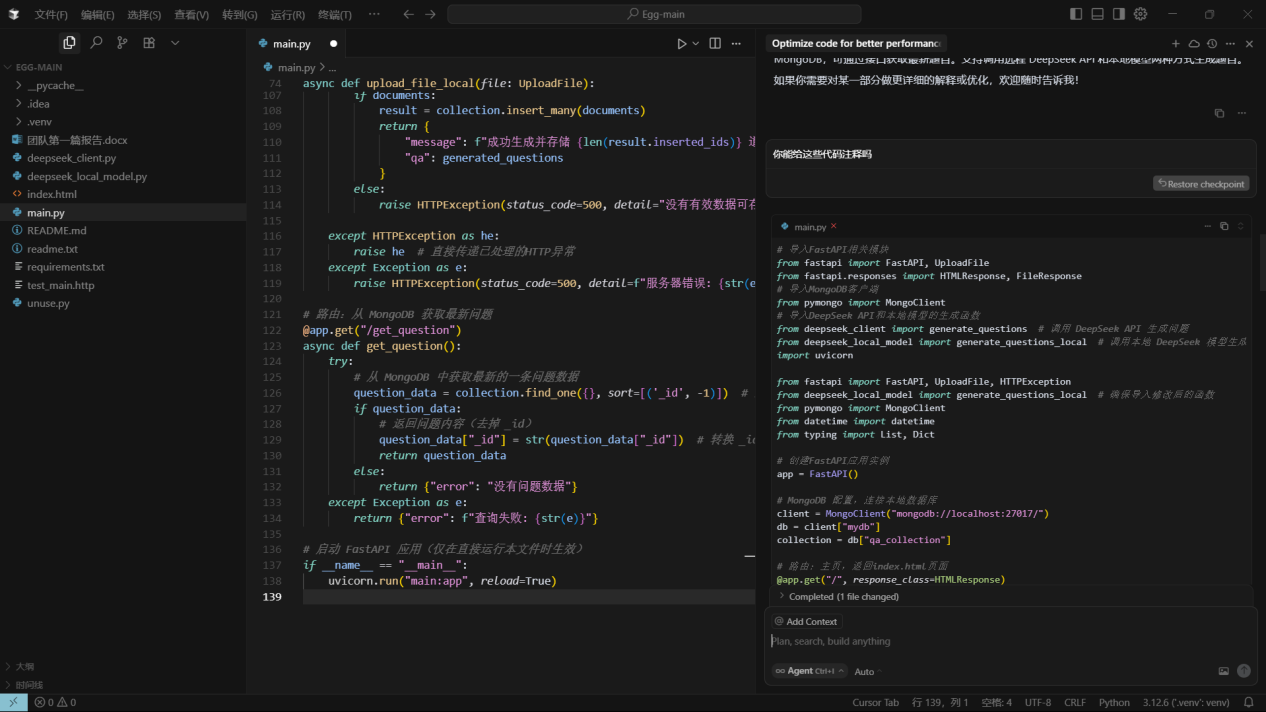
随后我尝试删除原有代码，让Copilot生成一个循环链表的基础结构。在此基础上，我通过逐步添加需求提示，让AI为链表增加了增删改查等操作功能。



二、Cursor 工具使用体验

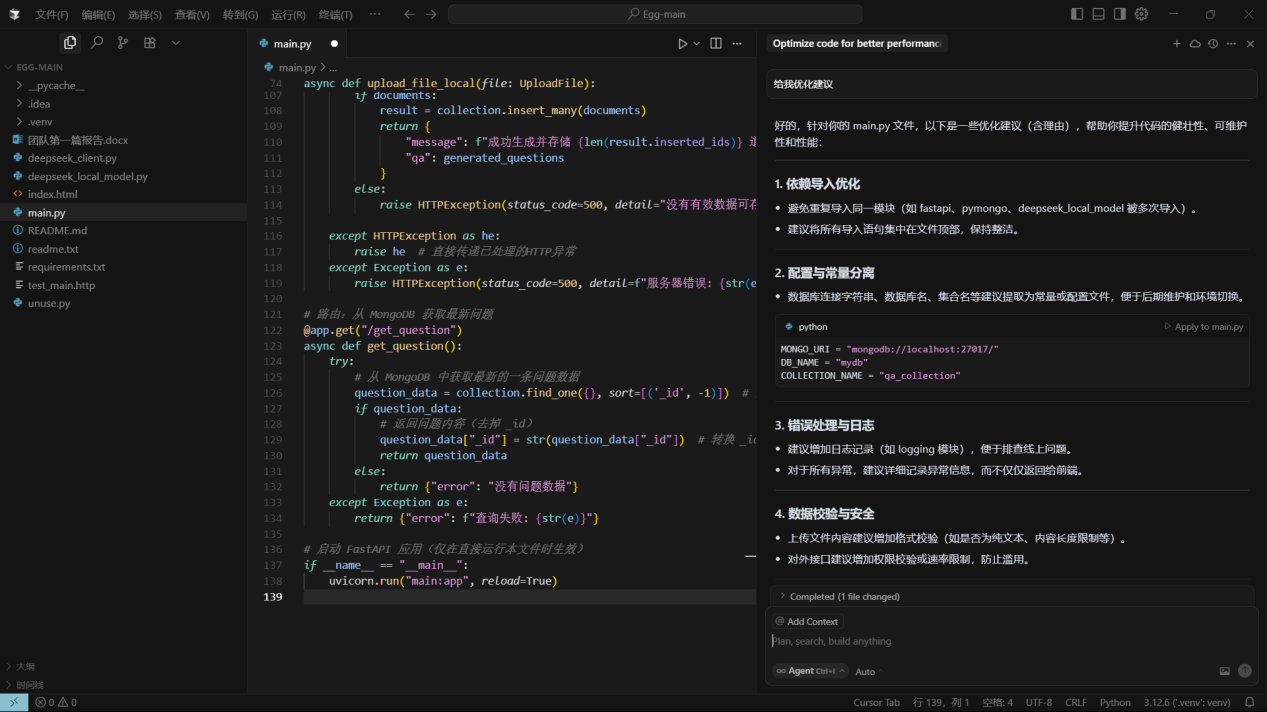
1. 代码分析与注释

将已有的代码导入Cursor后，使用其分析功能对代码结构进行了解读。AI不仅准确解释了代码逻辑，还为关键部分添加了清晰的注释。



2. 功能优化建议

Cursor还针对代码提出了几项优化建议，包括性能改进和可读性提升等方面。



三、AI编程工具在实际工作中的深度体会

通过这次实践，我对AI编程工具在实际开发中的应用有了更深刻的认识：

1. 开发效率的显著提升

- 快速原型构建：Copilot能在几分钟内完成基础代码框架，过去需要数小时的手写工作

- 减少重复劳动：如链表操作等常见模式，AI能准确生成模板代码

- 上下文感知：工具能根据已有代码风格保持一致性

2. 学习与开发的新模式

- 交互式学习：通过修改AI生成的代码，更直观理解算法实现

- 即时文档：Cursor的注释功能相当于随时可用的代码导师

- 最佳实践获取：AI常能提供我尚未掌握的优化技巧

3. 实际工作中的注意事项

- 验证必要性：

- 生成的代码需严格测试边界条件

- 特别是算法类代码，必须验证正确性

- 提示词工程：

- 明确的需求描述能得到更好结果

- 分步骤提示比一次性复杂需求更有效

- 代码所有权：

- 企业项目中需注意AI生成代码的合规性

- 关键业务逻辑仍需自主实现

4. 团队协作中的价值

- 代码审查辅助：AI能快速发现基础问题

- 知识传递：新成员通过AI注释更快理解代码

- 风格统一：自动格式化保持团队一致性

四、体会

AI编程工具已经从"可有可无"发展为"不可或缺"的开发助手。在实际工作中：

- 将Copilot用于40%的常规编码任务

- 使用Cursor处理30%的复杂逻辑和代码优化

- 建立"生成-验证-优化"的标准工作流程

这些工具不仅提升了我的个人效率，更改变了解决问题的方式和思维模式。随着技术的进步，AI编程助手必将成为开发者技术栈的核心组成部分。

团队成员：葛东豪

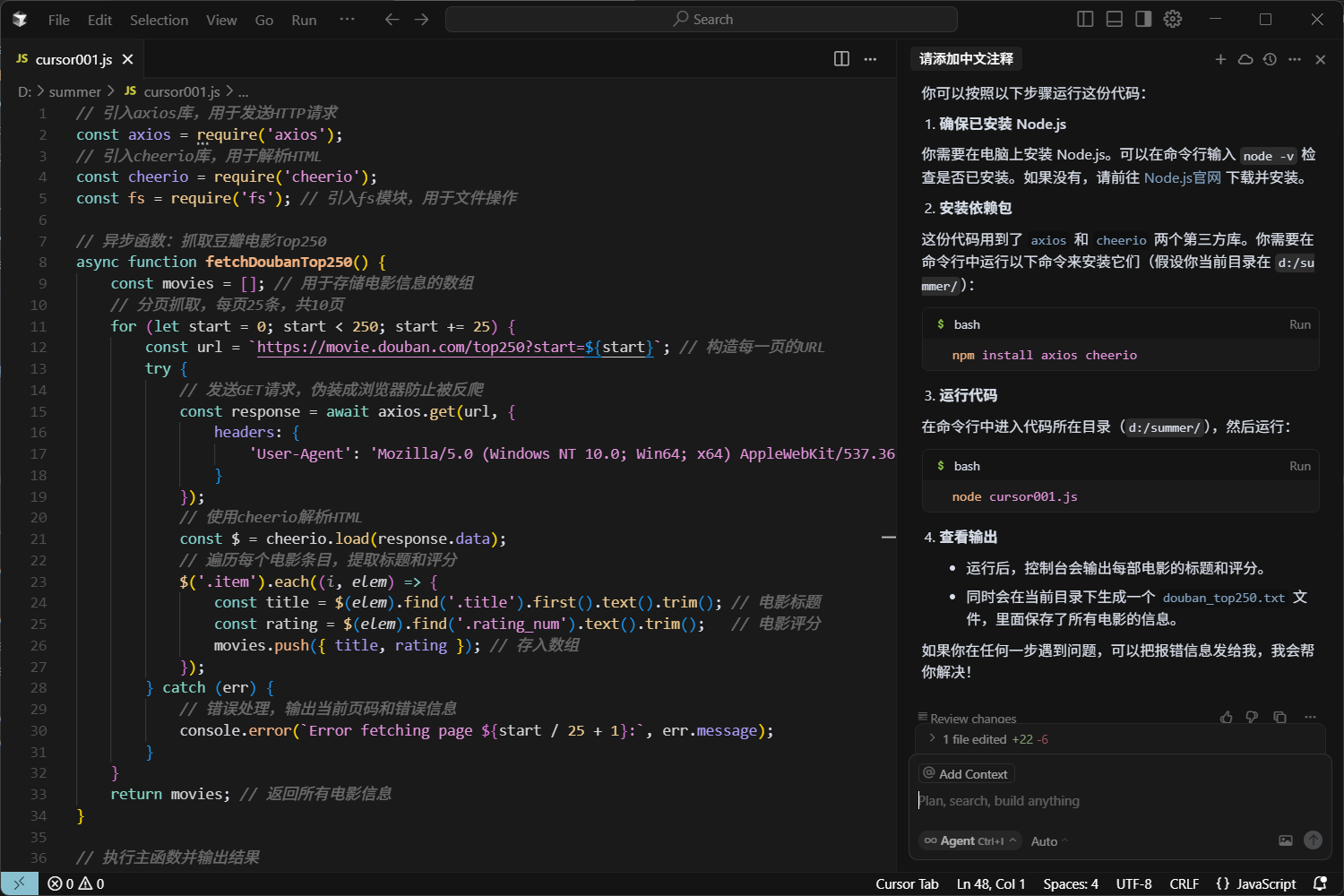
AI-coding 的经验

本次练习中，我下载并体验了以下两款AI编程辅助工具：

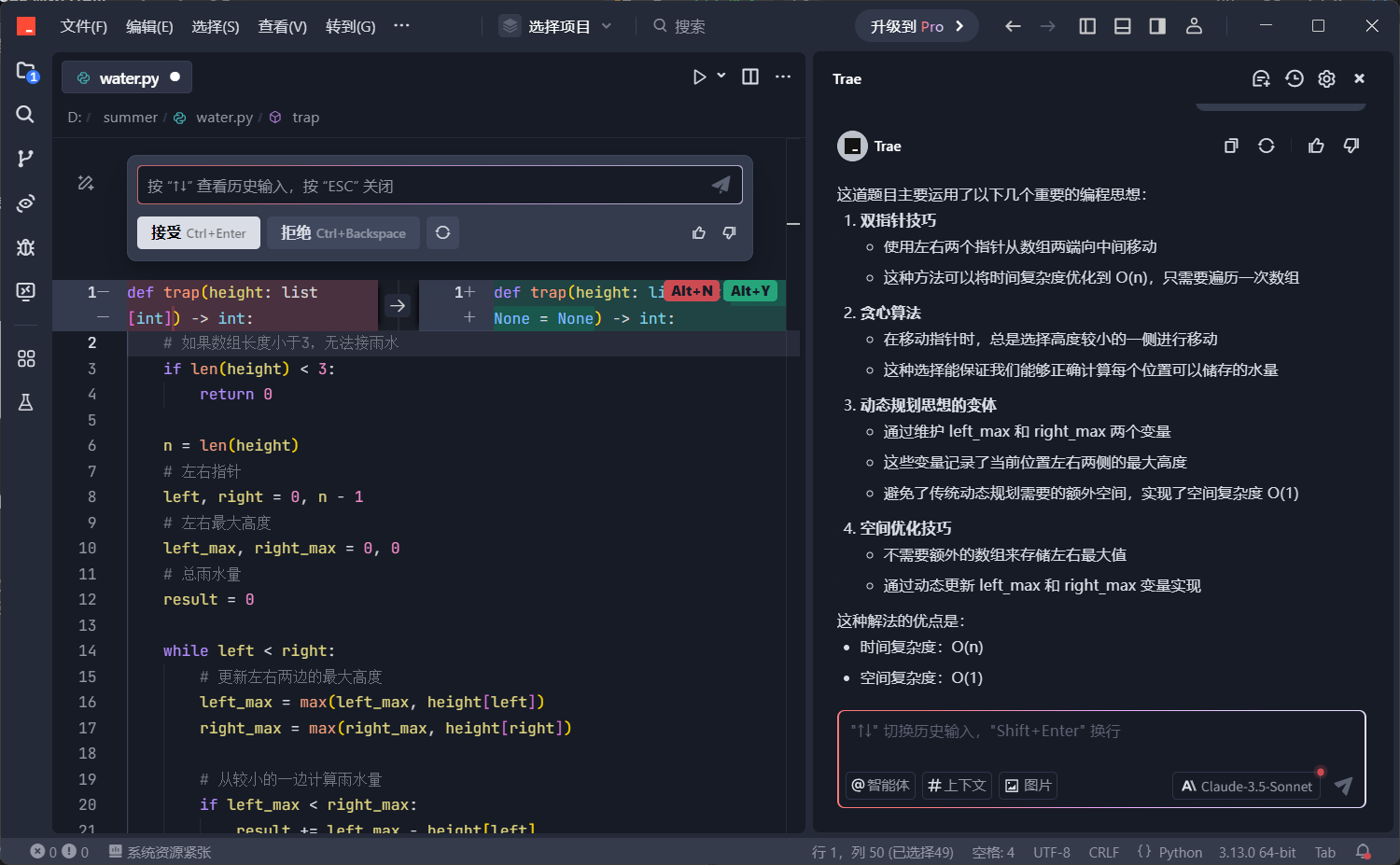
在Cursor AI里我提示AI给我写了一个python爬虫，用于爬取豆瓣top250的电影，然后又提示AI输出到一个txt中。

在Trae AI中我让AI帮我解决一道LeetCode上的Hard题目，AI准确地给出了结果，随后我又让AI给我分析代码里蕴含的算法思想。

Cursor AI

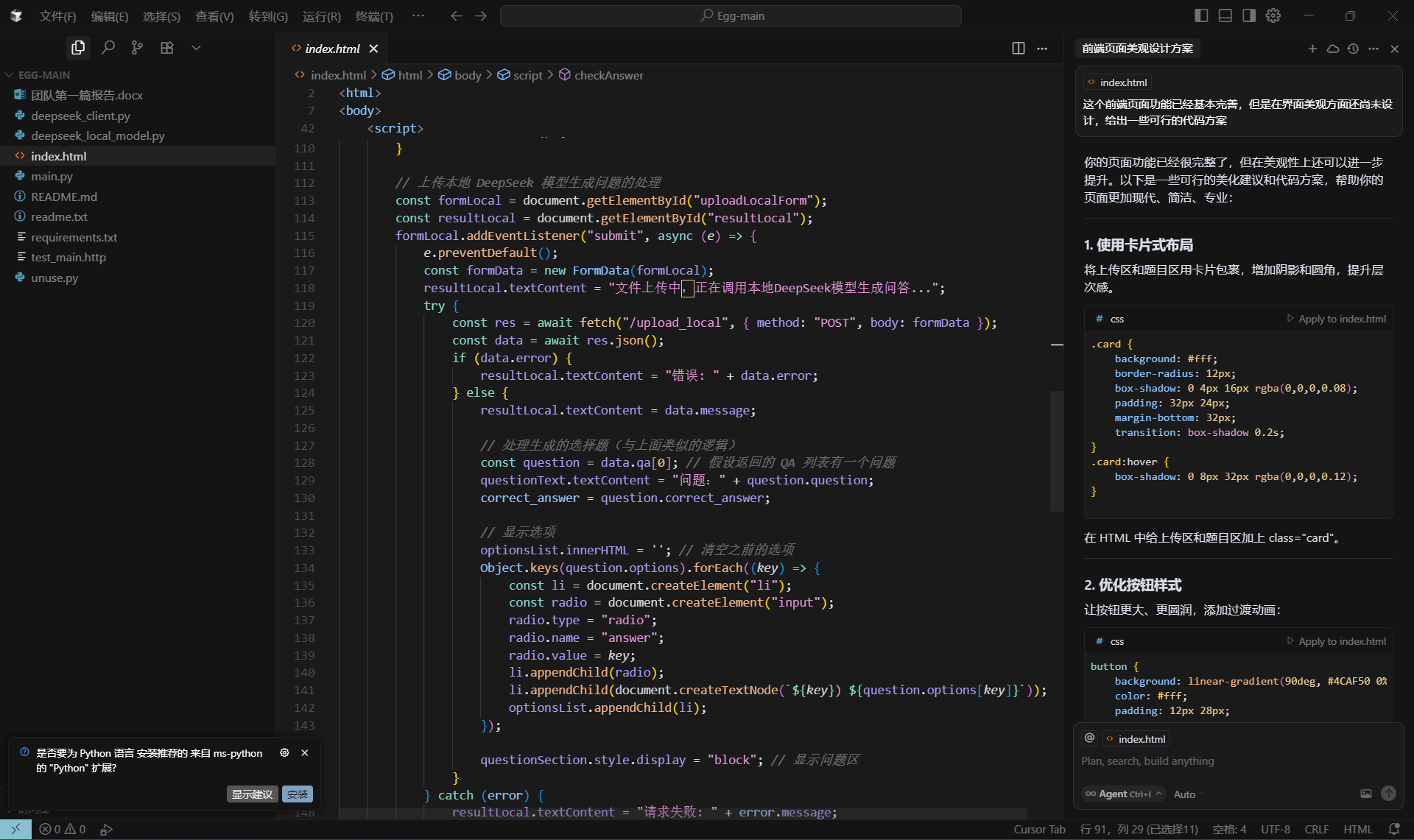


Trae AI



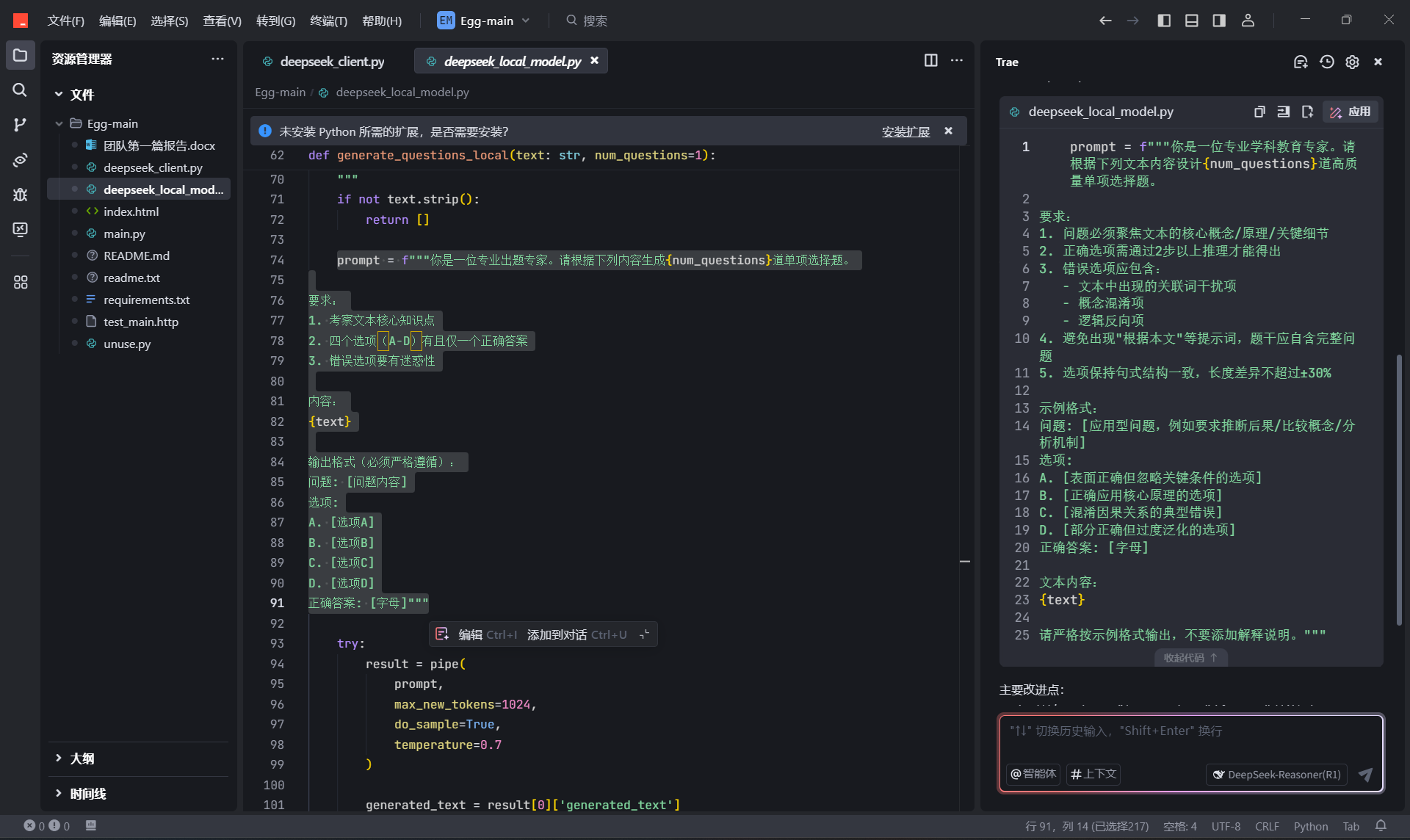
团队成员：贺子泰

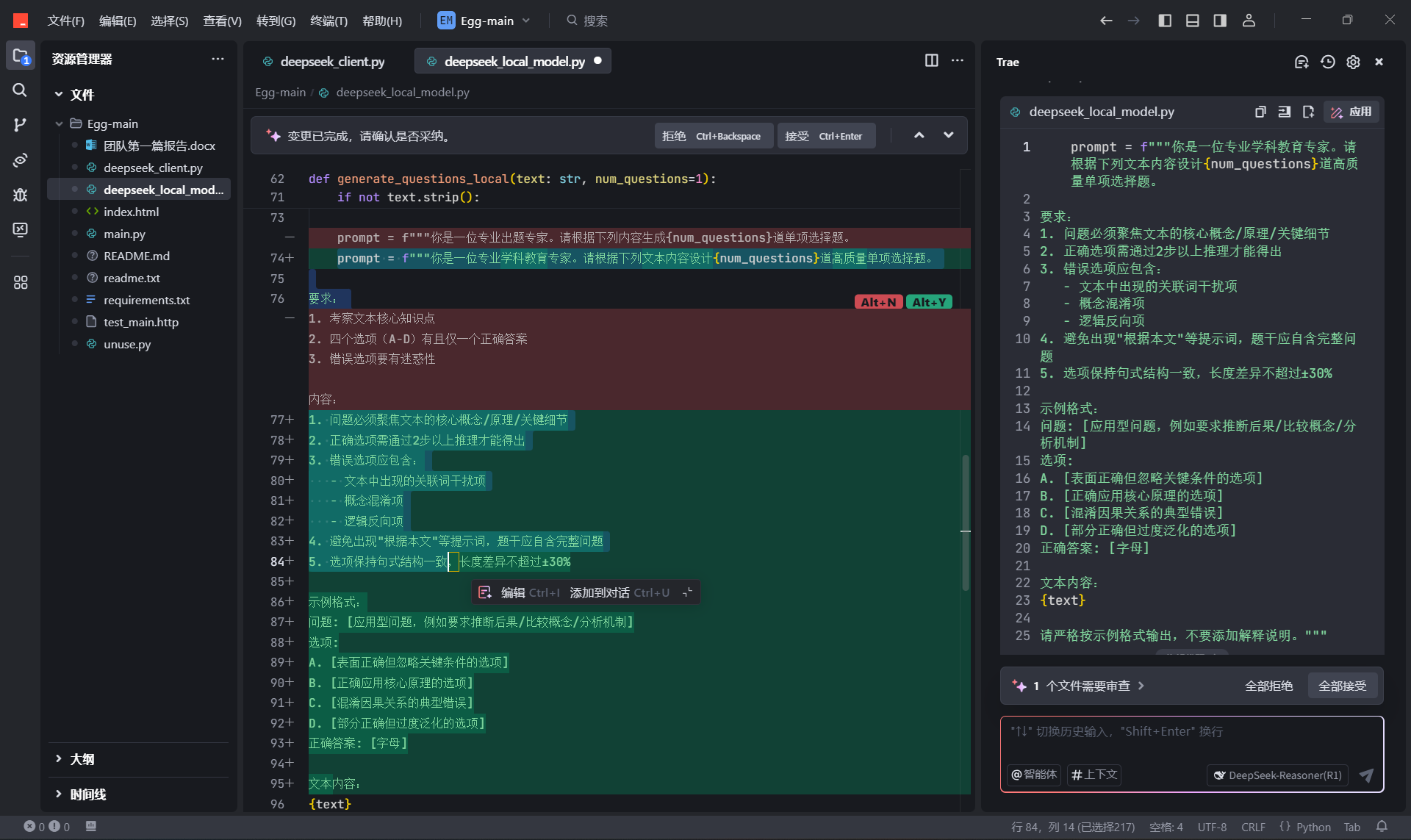
可以用如Cursor等工具来对前端页面进行改进。编程人员一般会更注重功能的编写与实现，但是对于界面UI美观度有所忽略，可以使用AI-Coding来对界面美观度进行进一步调整，例如使用更加多样化的UI，调节模块之间的大小以及相对位置，提升用户视觉体验，增加使用流畅度等。





可以用如Trae等工具来对功能代码进行编写，可以让其实现某一特定功能，或者对于某个已经实现的功能进行进一步改进。拿prompt engineering举例，使用AI生成的prompt也许可以更好的引导本地的预训练语言大模型，生成更接近我们预想的答案。也可以用AI-Coding对代码进行debug，或者进行算法上的优化，降低方法的时间、空间复杂度等。





1. 对构建之法相关章节的随笔

第二章：个人技术与流程

本章主要讲的就是关于个人的软件开发流程。首先讲解的单元测试就是让自己的软件保质保量，包括以何种方式测试自己的软件正确性与功能性，如何做一个完整的有用的单元测试；如何介绍了几种测试的方式，以及其中最重要的psp个人开发流程（它强调开发者在每一个环节（计划、设计、编码、测试、复审等）中都进行量化记录和规范操作，通过自我约束和数据反思来不断优化效率）。

个人感觉，这个psp流程还是比较有用的，因为我之前看一些团队的冲刺报告，他们都习惯用这种psp来记录自己工作的进行进度，我觉得很实用，也很必要。

第三章：软件工程师的成长

我认为本章是比较有实际意义的一章吧，看完之后对我的感触也挺深的。第一点个人能力的衡量与发展：我认为伴随着一个人发展的一生，包括积累专业知识、对软件工程理解的加深、提升职业成果等；以及演变成为具体的从小公司到大公司、证书考级等，这些都是个人发展的必经之路，同时，我觉得作者也借助”玩魔方“这一例子，给我们学生一些观点与想法：一步一步来，从大往小开始构思，找到自己能够完成的层次，弄清自己的定位，然后在一步一步的精进自己，从只会一面到恢复整个魔方。

第四章：俩人合作

本章比较浅显易懂，首先讲述了有关代码的东西，比如说代码的风格与规范；个人、团队要有符合自己风格的代码，要能够被众人所理解，要有一定的规范，并且在以后的设计中最好一直沿用；后续就是我今天进行的有关代码审查的文本。

很直白的一个问题：我们为什么要进行代码审查，代码审查有什么好处？

我觉得：提前发现 Bug，减少后期代价，提前发现了bug，后续工作就非常简单了，就不用那么费力了。

保持代码风格一致，提升可读性，大家的代码都经过审查了，长得都一样了，读起来自然舒服了。

促进知识共享和团队成长，代码审查并不是一个人能够完成的，这是我认为的，多人的合作能让团队成长的更加紧密与强大，能保证团队成员之间的互相扶持。

总结来说：代码审查是高质量代码背后的保障，是团队合作中的桥梁，是程序员成长的加速器。

最后作者提了一嘴俩人合作，包括有哪些阶段以及技巧，因为我们是一个四人团队，我就没有仔细阅读了，不过应用到四人团队应该也是一样的，磨合、影响、共同成长、接受反思、反思复盘。