

信息检索设计

2211771 原敬闰

November 9, 2024

1 背景和意义

计算机的出现短短不过百年，但是已经深深的改变了我们的社会，科技飞速发展，生活生产在计算机科学的帮助下愈发便捷。而程序员的 coding，从最开始只能使用人脑记忆的机器语言，汇编语言，到后来的 c/c++ 等高级语言，再到后来 ide 飞速发展后产生的代码自动补全，代码自动格式化等新设计，每一次的科学技术发展都同样方便着代码开发人员的 coding。近几年，随着大语言模型的飞速发展，也有许多诸如 github copilot，vscode talkx 等工具帮助程序开发人员进行代码编写。而笔者也想基于这两年 coding 以及使用大模型的经历来设计一套配备有 coding copilot 的 ide 来方便程序开发。

2 现状分析

现如今的大学生使用大语言模型来方便代码编写一般有以下三种情形：

- 最基础的，同时也是最多人使用的。直接将要求详细的说明出来，并且说明要使用的语言，然后让大语言模型为我们生成，例如：

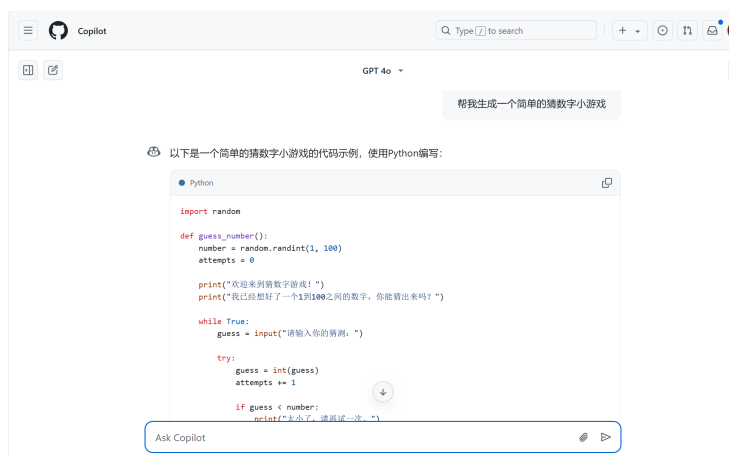
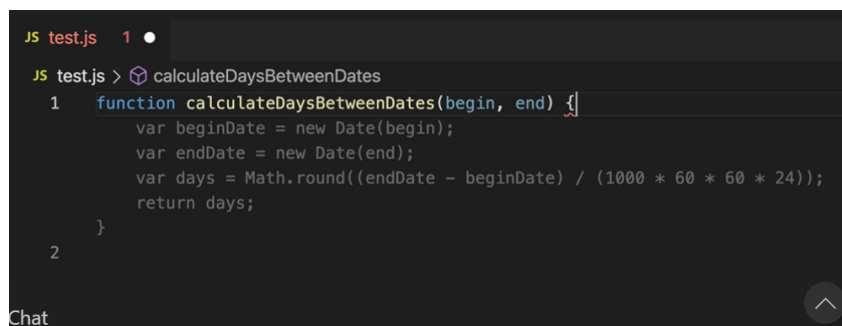


Figure 1: 要求大模型为我们生成一个简单的猜数字小游戏

但是这种方案生成的代码往往都只在简单的时候有效，而在稍微复杂一点的情况下例如让其生成一道算法题的代码就会漏洞百出，这是因为当前的大语言模型虽然可以实现和人类交互，但是逻辑能力堪忧，而代码编写却对逻辑能力要求极高。所以这种方案只适合作为简单的演示，而无法真正编写能够使用的项目代码。

- 而较为高级一点的方案是在 ide 中融合使用。例如 vscode 中使用 github copilot 插件，这个模型学习了 github 上海量的代码，能够生成更为完善逻辑更合理的代码，同时也可以 coding 的时候提供自动补全功能，如图所示：



The screenshot shows a VS Code editor window with a file named 'test.js'. The editor is displaying a JavaScript function 'calculateDaysBetweenDates' with parameters 'begin' and 'end'. The function body is being auto-completed by GitHub Copilot, showing the following code:

```
1 function calculateDaysBetweenDates(begin, end) {  
    var beginDate = new Date(begin);  
    var endDate = new Date(end);  
    var days = Math.round((endDate - beginDate) / (1000 * 60 * 60 * 24));  
    return days;  
}
```

The line numbers 1 and 2 are visible on the left side of the editor. At the bottom left, there is a 'Chat' button, and at the bottom right, there is a scroll-up arrow.

Figure 2: 代码预测补全

然而，这种方案和第一种区别并不很大，都是使用大语言模型来编写代码，因此存在同样的问题，这种方案也只是利用了之前的程序员编写的代码，而没有帮助程序开发人员提升自身的能力，也不能帮助开发人员编写出真正能够使用的代码。

笔者愚见，如果一个开发人员不能完全的把握代码的情况，而只是用大语言模型来生成一段代码就交付使用，无疑是非常不靠谱的行为。

- 当然，前两种方式都类似于偷懒而非学习，毕竟坚持让大模型来代写代码并不能提升自身的代码能力。而第三种就是借助大模型来分析代码并且提供帮助，例如利用大模型来理解一个操作系统内核的源码；或者询问大模型某个库函数的使用，或者根据正在编写的代码给出一些参考方案。这种方式基本上相当于有了一个知识极度丰富，同时也能完全理解开发人员代码的助手来协作编程，可以极大程度的方便开发人员编写代码的同时提升 coding 能力。

当然，每个人的情况都有所不同，开发人员在使用大语言模型的时候也都是基于自己的需求来综合使用以上三种方法，而笔者比较推荐的方法是将第三种进行改造升级，因此后文中基于笔者日常使用较多的 vscode+github copilot 来进行方案设计。

为了设计功能，笔者认为还是需要简要的介绍一下 vscode+github copilot 以及其优缺点。其界面如下图所示：

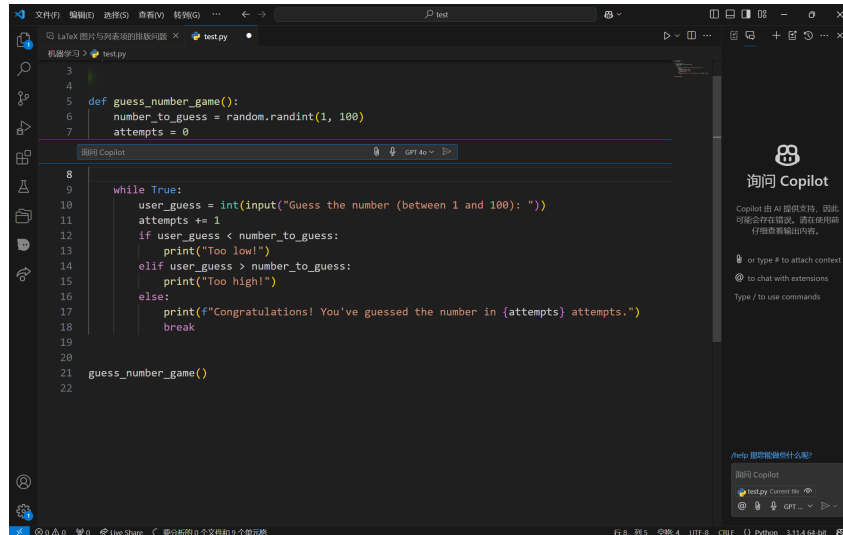


Figure 3: vscode+GitHub copilot

- 优点：页面简洁且融合度较高，主要功能有两个。
 - 在工作区域中的内联聊天窗口，用户可以通过一些命令来让其工作，例如 `/fix` 来修改代码中的报错；`/explain` 来解释当前选中的代码；`/doc` 为当前的代码编写文档等。
 - 在页面右侧的聊天窗口，用户可以直接与其进行交流，功能基本和 `chatgpt` 等主流的大语言模型的功能类似，可以附加文件来询问交流
- 缺点：虽然相对于从前需要上网搜索读各种博客的情况简单了许多，但是仍然相当的麻烦，毕竟从原本的需要打开许多窗口的麻烦变成了需要绞尽脑汁让逻辑能力较为欠缺的大语言模型来理解用户的命令需求，并且还需要面对在我无数次指出它的问题并改变我的话术之后，它还是选择说抱歉之后把之前的输出重新粘出来折磨我，显然这样的情况并不是我们的初衷。当然，这种情况的一部分原因是由于大语言模型本身存在的问题，但是笔者想要指出的关键在于，我们并不需要让工厂里的每一个螺丝都会说话毕竟那样着实有点恐怖，也不需要让 `copilot` 需要和我们像一个人一样极其灵活的交互，因为大部分时候我们和现实中的人合作都很难默契的完成任务，更何况是一个“模拟人”。

3 方案设计

经过上一个阶段的论述，我们已经明确了在 `vscode` 中嵌入 `copilot` 所存在的问题，即过于的强调其交互性从而使得问题更加的复杂，更具体的说，作为一个助手，当然笔者更希望将其工具化，`copilot` 将太多的功能都塞给了聊天界面也就是右侧的聊天，导致我们无论需要什么都必须和人机队友进行交互。

因此笔者的设计中将其拆分成一个个更具体的功能单元，而这一个个功能单元仍然是通过大语言模型进行实现。只是减少了一些交互性以提高效率。

具体功能设计模块有：改进后的报错板块，库及库函数解释板块，保留的经过缩减后的交互聊天板块，以及扩充功能的内联聊天板块。

3.1 报错板块

有过 coding 学习经历的人应该都理解代码报错的痛苦，莫过于**没写两行就满屏飘红，没报错但是运行不了以及能运行但是莫名其妙崩了**。传统的报错板块是由编译器（C/C++ 和 Java 等）或解释器（python 和 JavaScript 等）以及静态分析工具来进行的，这些工具提供的报错信息通常让人摸不着头脑，甚至经常出现塞满整个终端的报错信息，让人头都大了。

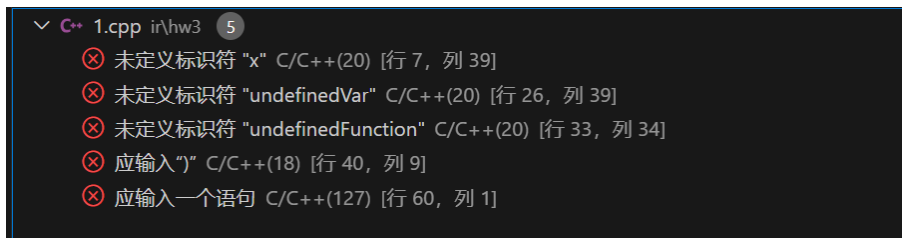


Figure 4: 传统的报错信息

对于水平没有到特别高的开发人员，这些旨在帮助调试错误的报错信息的唯一作用就是复制后粘到浏览器中回车，然后看到博客中的**相似报错**被博主完美解决，但是自己却焦头烂额，近年来有了大语言模型后，开发人员可以将报错信息和代码一同发送给大语言模型让其帮忙定位错误，但是如果文件过大就没办法了，所以笔者觉得可以将这个功能一同集成到 IDE 中，改进后的报错板块应该可以由两个选项，可以选择原本的报错信息，也可以选择经过大语言模型经过分析代码和原本的报错信息后得到的分析结果，帮助开发人员定位到错误位置。例如：你将一个 Car 类型的对象赋值给了一个 Baoma 类型的指针，这样是不被允许的，因为不能将一个父类的对象赋值给其子类的指针。

3.2 库及库函数的解释板块

传统的开发人员在需要使用一些函数或一些库时，通常要去网上搜集功能文档和讲解博客，大语言模型流行后笔者通常将要使用的库或库函数发送过去要求解释，这个功能同样可以集成在 ide 中，来提升开发的效率。

具体而言，ide 应该维护一片区域，从程序中分析出使用的库和库函数，让大语言模型进行学习其文档以及一些教学博客当然很可能未来就不需要了，然后提供给开发人员，最好能够在工作区中插入一段示例代码帮助理解。库函数解释板块的界面应该分为两个标签页

- 用来动态的分析当前用户使用的库和其中的库函数。
- 是用户自主查询页面，包含两个功能
 - 输入某个库中的某个库函数即可得到大语言模型对其的讲解以及演示代码。
 - 根据用户输入的功能描述，跨多个程序库向用户推荐相关的 API 方法。[\[1\]](#)

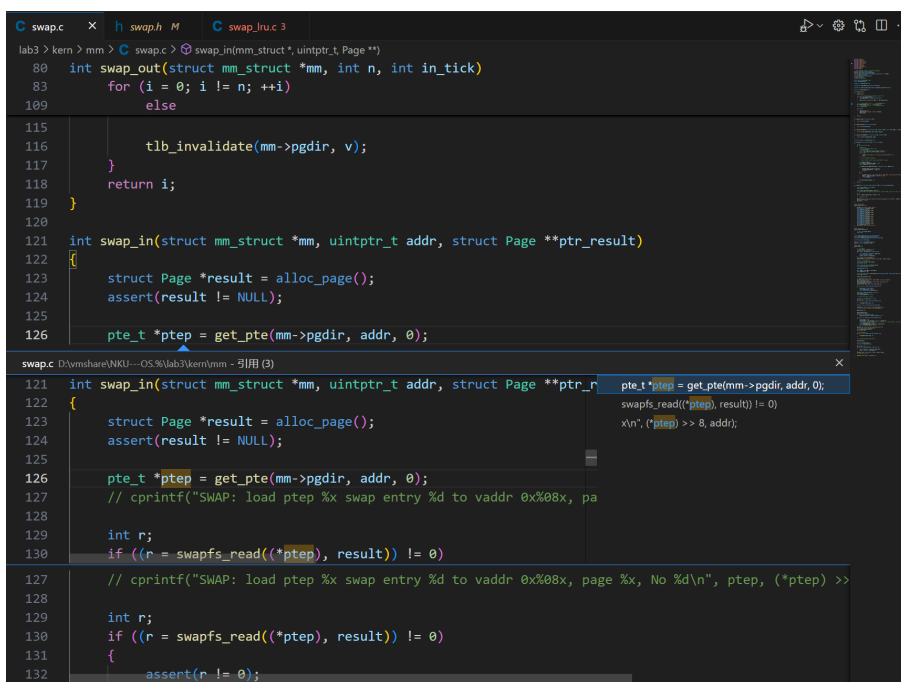


Figure 5: 示例代码插入位置样式（请忽略实际内容）

而且由于用户在不断成长，所以许多函数或库一开始还需要提示讲解，长时间后就不需要了，所以这里的推荐算法需要使用动态信息检索推荐算法 [2]，并且需要动态的记录近期内用户查询这个函数 or 库的次数，用户使用这个函数 or 库的次数来判断是否将其剔除。

另外，也需要在此处引入 cache 机制，毕竟有许多的库和函数是常用的，如果每次使用都让大语言模型重新分析一次未免效率过于低下且造成算力浪费。

3.3 内联聊天板块

当前的内联聊天板块支持了多种功能包括 explain（解释代码），fix（修复代码），doc（添加文档）等功能。

笔者建议添加优化板块，即评价已经编写的代码的优劣，并提出优化建议及思路 [3]，笔者提出的设想是在代码编写的右侧给出划分出一部分区域来给出优化建议（类似上文中提到的补全代码功能的效果）。

3.4 交互聊天板块

将众多的功能从聊天板块中分化出去后，笔者依然保留了交互聊天板块，因为笔者认为聊天功能可以起到一个管家的功能，可以执行一些更高权限的命令，例如将编译项目，保存、创建、删除文件的功能都交给聊天功能，它也可以保留一些接口，方便未来进行扩展。

另外，虽然笔者认为将大量的功能交给聊天直接解决不大必要，但是显然许多事情交给 chatgpt 来处理是一件很不错的的事情，因此保留这个板块，以备不时之需。

3.5 页面布局控制

虽然在上述设计中，分离出了许多界面，但是并不会杂乱，采用标签页的设计应该能够让页面整洁很多夫不子就多分几级标签页。



Figure 6: vscode 的终端窗口设计就使用了标签页

4 技术路线

下表汇总了上一部分方案设计中提出的设想和它们对应的实现技术路线。

模块名称	功能描述	技术路线
改进后的报错板块	提供原始报错信息和经过大语言模型分析后的报错信息	编译器/解释器（如 GCC、Clang、Python 等）、大语言模型 API（如 OpenAI GPT-4）
改进后的报错板块	帮助开发人员定位错误位置，并提供详细的错误解释和解决方案	编译器/解释器（如 GCC、Clang、Python 等）、大语言模型 API（如 OpenAI GPT-4）
库及库函数解释板块	动态分析使用的库和库函数，提供文档和示例代码	静态代码分析工具、大语言模型 API（如 OpenAI GPT-4）、缓存机制（如 Redis）
库及库函数解释板块	提供用户自主查询功能，包含库函数讲解和跨库 API 推荐	静态代码分析工具、大语言模型 API（如 OpenAI GPT-4）、缓存机制（如 Redis）
内联聊天板块	提供代码解释功能	大语言模型 API（如 OpenAI GPT-4）、IDE 插件开发框架（如 VS Code Extension API）
内联聊天板块	提供代码修复功能	大语言模型 API（如 OpenAI GPT-4）、IDE 插件开发框架（如 VS Code Extension API）
内联聊天板块	提供文档添加功能	大语言模型 API（如 OpenAI GPT-4）、IDE 插件开发框架（如 VS Code Extension API）
内联聊天板块	提供代码优化建议功能	大语言模型 API（如 OpenAI GPT-4）、IDE 插件开发框架（如 VS Code Extension API）
交互聊天板块	执行高权限命令，如编译项目、保存、创建、删除文件等	大语言模型 API（如 OpenAI GPT-4）、IDE 插件开发框架（如 VS Code Extension API）
交互聊天板块	保留一些接口，方便未来进行扩展	大语言模型 API（如 OpenAI GPT-4）、IDE 插件开发框架（如 VS Code Extension API）
页面布局控制	采用标签页设计，保持页面整洁	IDE 插件开发框架（如 VS Code Extension API）、前端框架（如 React）

5 总结

笔者基于自己学习代码的经历，以 vscode+github copilot 为蓝本，设计了一个新版的 ide，这个集成开发环境能够更好的将大语言模型融入到了 ide 的设计中，能够更好的利用当前代码、报错信息、网络文档等资源，并将其整合出来，提升开发者的效率。

在设计过程中，笔者主要参考了初学代码时期理想的“老师”以及现在流行的物联网智能管家概念，将这个 ide 虽然目前没有一行代码实现设计成一个类似代码管家的效果，希望未来真的有可能

如此让程序员省心的软件。

References

- [1] 宋文灏. 基于信息检索的 API 推荐算法的设计与实现 [D]. 上海交通大学,2016.DOI:10.27307/d.cnki.gsjtu.2016.001187.
- [2] 杨文. 动态信息检索推荐算法研究 [J]. 信息系统工程,2021,(08):44-46.
- [3] 李真成. 计算机编程代码优劣评价系统的设计与研究 [J]. 微型电脑应用,2023,39(12):123-126.