高级语言程序设计

实验报告

南开大学 计算机大类
姓名 李金璇
学号 2312418
班级 0978 张海威老师班
2025 年 5 月 6 日

目录

高级语言程序设计大作业实验报告

一. 作业题目

CurriculumHelper-AI 金融课程点读笔记本

二. 开发软件

Qt 6

Visual Studio Code

三. 课题要求

1) 面向对象。
2) 单元测试。
3) 模型部分
4) 验证

四. 设计思路

我是主修金融辅修计算机的同学，平时不太打游戏，因此想做一个很实用的工具。这是一个金融知识的“点读笔记本”，可实现“哪里不会点哪里”。我作为金融专业的学生，记的笔记中有时候存在不会的专业名词或者有疑问的地方，就可以选中，然后本程序就可以帮忙进行补充和解答，补充在笔记的下方。这样就可以实现下次看笔记的时候不需要再次查找，且这次补充笔记也不需要自己誊写。

本项目受copilot启发，当我用latex记笔记时，copilot经常可以帮我写出想要的latex代码，提高了记笔记的效率。它可以实现类似CoPilot的功能，当你记了一些笔记，但是有些不太了解、存在疑问、需要补充的内容，可以选中你想补充的部分，然后按下工具栏中的ChatGPT图标，随后智谱的ChatGLM就会被自动调用，把你选中的内容发送给大模型，并且将生成的回答补充在光标位置之后。这样可

以省去自己去搜索、誊抄或者以后再次搜索的时间。

比如，一个经济系学生，正在记笔记，当写到“供需曲线的定义”后，懒得抄书了，就可以选中这几个字，并且按下 ChatGPT 的图标。然后等待一会，关于“供需曲线的定义”便会出现在笔记本上。

本项目的亮点是用Qt实现了TCP服务器，以此和Python的ChatGLM进行通信，实现了一个简单的前后端分离。以及，通过API的方式调用了智谱的ChatGLM大模型，模仿之前GPT4 Prompt大赛冠军的设计方法，设计了Prompt，实现了简单的AI补充&解答功能，是Fine-Tuning 的简单基础，为我后续学习和研究NLP以及Agent有帮助。

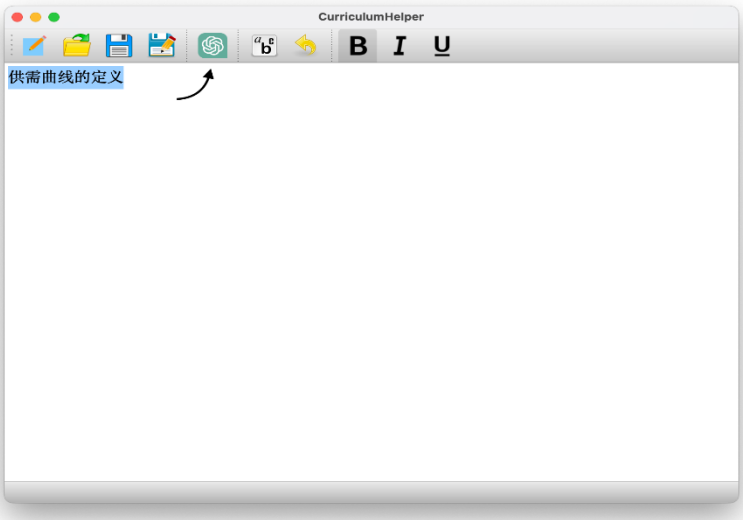


图 1：想要补充的内容

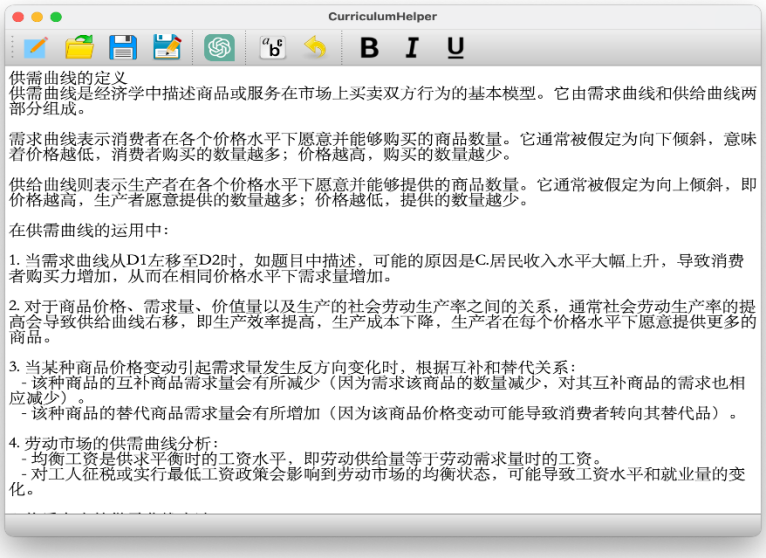


图 2：AI 解答&补充之后

五. 主要流程

1. 整体流程

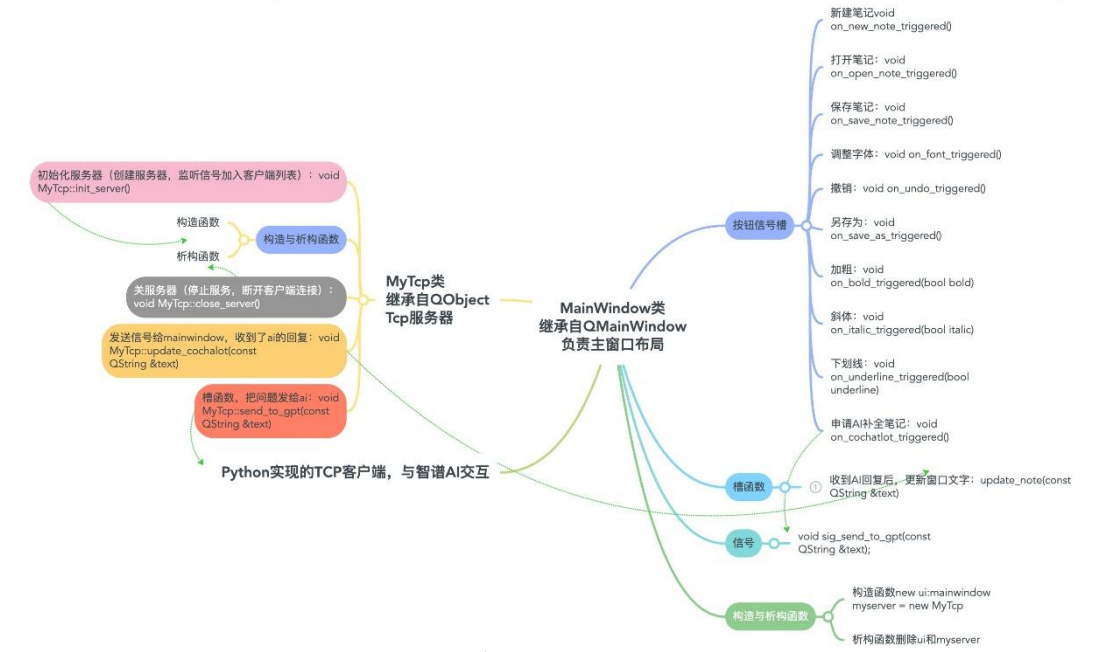


图 3: 流程图

每个对象使用后都进行删除，确保内存不泄漏。类之间使用connect信号槽进行通信，根据信号启动功能，体现了面向对象思想。MyTcp和MainWindow类继承于QT中预先封装好的类，极大简化代码，只需直接调用函数即可实现功能，体现了封装和继承。

2. 重要代码

信号槽，用于在主界面显示回答，使用了 C++ 的新特性 Lambda 函数

```
//收到数据，用update_cochalot发送信号，在mainwindow中设计槽，更新界面上的文本，
connect(socket, &QTcpSocket::readyRead, [this, socket]{
    if (socket->bytesAvailable() <= 0)
        return ;
    const QString recv_text = QString::fromUtf8(socket->readAll());
    qDebug() << "Read:" << recv_text;
    update_cochalot(recv_text);
});
```

MyTcp 类的析构函数调用的函数，逐个断开连接，最终在析构函数中删除 server，确保内存不泄漏

```
void MyTcp::close_server()
{
```

```
    //停止服务
    server->close();
    for(QTcpSocket * socket:qAsConst(client_list))
    {
        //断开与客户端的连接
        socket->disconnectFromHost();
        if(socket->state()!=QAbstractSocket::UnconnectedState){
            socket->abort();
        }
    }
}
```

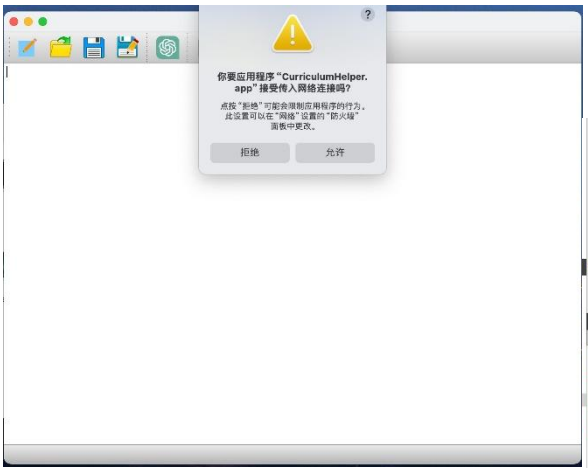
服务器可以接受多个客户端连接，使用多个 Lambda 函数，减少代码量

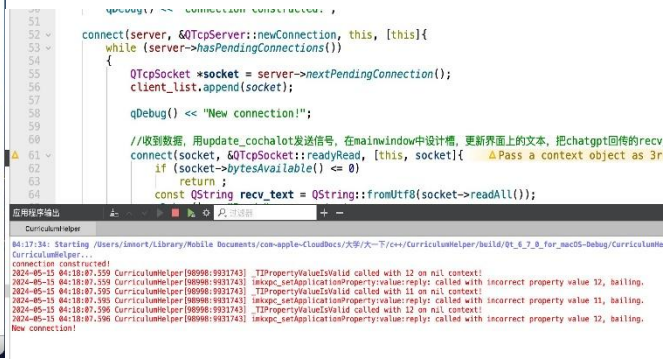
```
connect(server, &QTcpServer::newConnection, this, [this]{
    while (server->hasPendingConnections())
    {
        QTcpSocket *socket = server->nextPendingConnection();
        client_list.append(socket);

        qDebug() << "New connection!";

        //收到数据，用update_cochalot发送信号，在mainwindow中设计槽，
        //更新界面上的文本，把chatgpt回传的recv_text显示在窗口
        connect(socket, &QTcpSocket::readyRead, [this, socket]{
            if (socket->bytesAvailable() <= 0)
                return;
            const QString recv_text = QString::fromUtf8(socket->readAll());
            qDebug() << "Read:" << recv_text;
            update_cochalot(recv_text);
        });

        //如果出现错误，显示错误信息
        connect(socket,&QAbstractSocket::errorOccurred,[this,socket]
        (QAbstractSocket::SocketError){
            qDebug() << (QString("[%1:%2] Soket Error:%3")
                         .arg(socket->peerAddress().toString())
                         .arg(socket->peerPort())
                         .arg(socket->errorString()));
        });

        //client断开连接时，销毁socket
        connect(socket,&QTcpSocket::disconnected,[this,socket]{
            socket->deleteLater();
```

```
            client_list.removeOne(socket);
            qDebug() << (QString("[%1:%2] Soket Disonnected")
                                .arg(socket->peerAddress().toString())
                                .arg(socket->peerPort()));
        });
    }
});
```

六. 单元测试

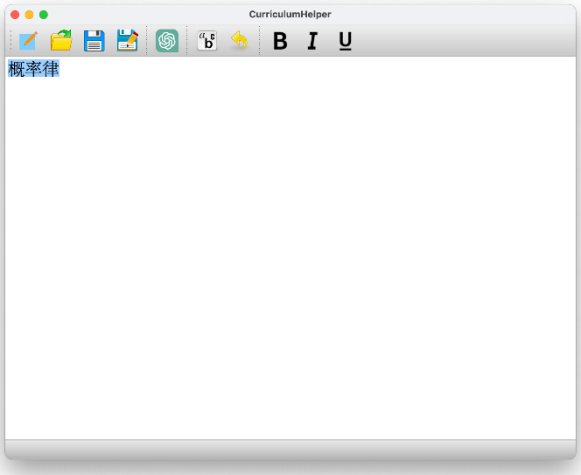
1. 服务器链接客户端测试

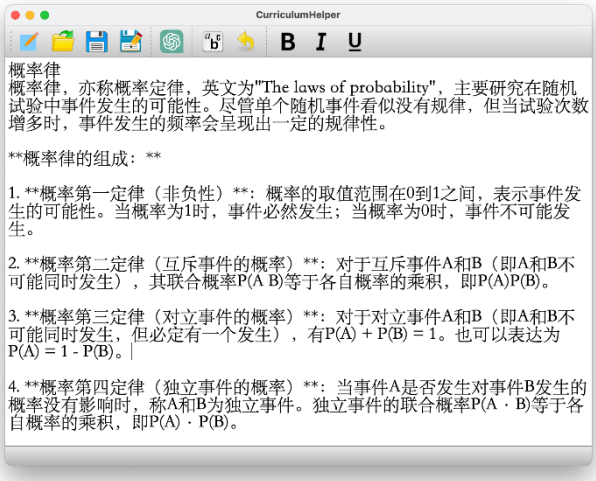




弹出窗口询问是否允许程序接入网络，并且在控制台输出了“New connection!”说明服务器成功接受了客户端的连接。

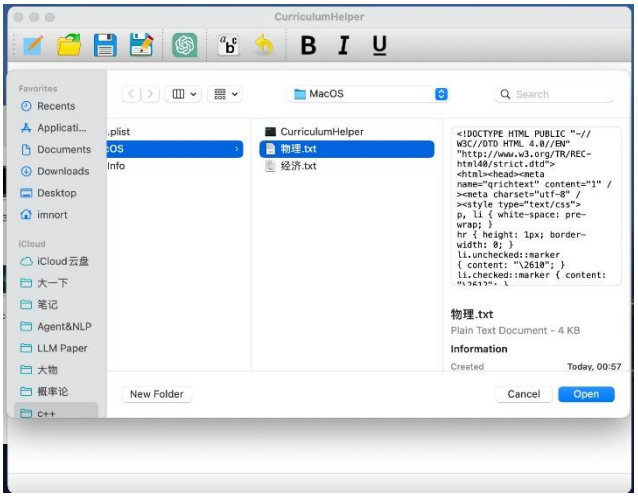
2. 调用 ChatGLM，回答问题并补全笔记

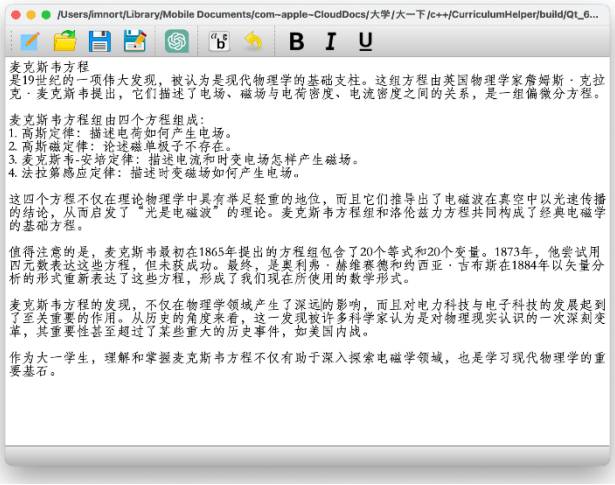




能够针对选中内容给出笔记结构的回答，说明ChatGLM成功调用。

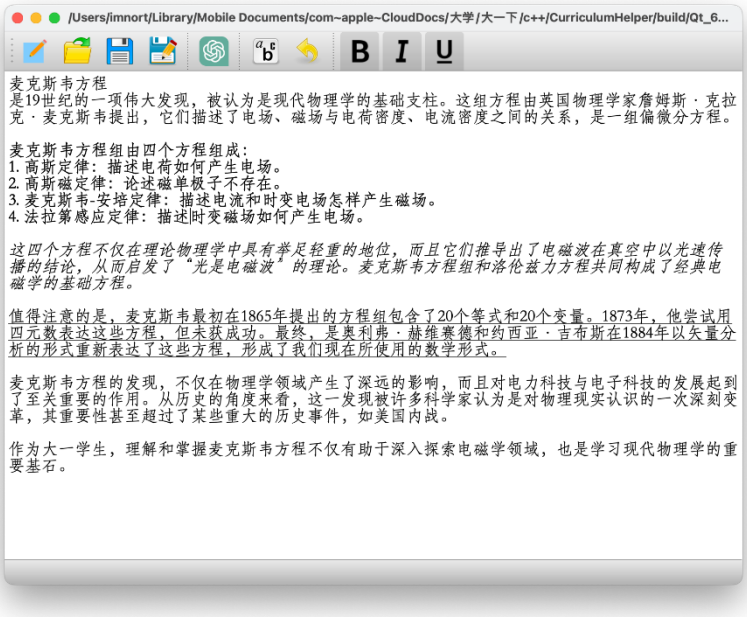
3．笔记本保存与打开笔记功能测试





能够弹出窗口，选择文件保存，打开文件，说明保存和打开功能正常。

4．笔记本加粗，下划线，斜体功能测试



由图可见，加粗，下划线，斜体功能正常。

测试结果

TCP服务器与客户端，ChatGLM的调用，笔记本的保存与打开，加粗，下划线，斜体功能均正常。

七. 收获

学会了QT的使用，感觉QT的信号槽机制非常有面向对象的思想，每个对象之间可以相互进行不同功能的耦合，方便进行非过程性的编程。

内存管理意识提升，以前做OJ题不需要关注析构函数和内存释放，但是需要用完每个变量后手动释放内存，或者借助QT的内存释放机制自动释放。

学会了如何调用ChatGLM等大模型，如何设计Prompt，如何设计一个简单的AI补全功能，对NLP有了初步了解。

同时，感受到了工程代码和平时做题的区别。工程代码需要考虑各种情况，处理并且输出错误，比如大模型无法调用，需要设计重新访问，无法链接客户端也需要报错等。

对网络有了更深入的了解，简单学习了TCP的原理，并且成功的用TCP实现了本地不同语言间的内容通信，便捷安全。

可以说这次大作业学到的内容大抵都在未来还会继续使用。