Notice

本项目将通过使用Git管理小组大作业并通过`git log`体现工作量，需要遵循规范的分支管理、提交流程和协作方式。以下是详细步骤和操作指南，确保每个人的工作可追溯、可量化：

一、前期准备：搭建Git仓库结构

1. 组长创建仓库（核心仓库）

- 我在学校的Gitlab平台创建了一个**公开仓库**（命名为`aiiiii group project`），作为项目的核心仓库。
- 我已经对仓库进行了初始化：添加了基础文件（如`README.md`说明项目、`Allocation.md`说明项目、`notice.md`说明项目），完成第一次提交（`initial commit`）。

2. 组员Fork仓库（分支开发基础）

- 每个组员访问我的仓库页面，点击右上角**Fork**按钮，将仓库复制到自己的账号下（如组员A的仓库为`A/aiiiii group project`，组员B的为`B/aiiiii group project`）。
- 每个组员将自己Fork的仓库克隆到本地：

```
git clone https://git.tsinghua.edu.cn/自己的账号/aiiiii group project.git
cd aiiiii group project
```

3. 关联上游仓库（同步核心更新）

- 为了同步我的仓库的最新代码，每个组员需要将我的仓库设置为“上游仓库”：

```
git remote add upstream https://git.tsinghua.edu.cn/组长账号/aiiiii group project.git
```

- 后续可通过以下命令拉取我的仓库的更新（避免代码冲突）：

```
git fetch upstream  # 获取上游仓库最新代码
git merge upstream/main  # 合并到本地当前分支（假设主分支为main）
```

二、分支管理规范：清晰追踪每个人的工作

采用**功能分支开发模式**，每个功能/任务对应一个独立分支，避免直接在`main`分支上修改。

1. 分支命名规则

格式：`[功能模块]-[开发者]-[简要描述]`，例如：

- 成员A开发正常聊天功能：`chat-base-A`
- 成员B开发网络搜索：`search-B`
- 修复流式传输bug：`stream-fix-A`

2. 分支创建流程

- 每次开发新功能前，确保本地`main`分支与上游同步：

```
git checkout main  # 切换到主分支
git pull upstream main  # 拉取上游最新代码
```

- 创建并切换到自己的功能分支：

```
git checkout -b chat-stream-A  # 创建并切换到“流式传输-A”分支
```

三、提交规范：通过`git log`体现工作量

提交记录（`git commit`）是展示工作量的核心依据，需清晰、规范，让助教通过日志即可了解你做了什么。

1. 提交信息格式

遵循`[类型]: [具体内容]`，类型包括：

- `feat`：新功能（如`feat: 实现chat.py基础聊天函数`）
- `fix`：修复bug（如`fix: 解决流式传输中断问题`）
- `refactor`：代码重构（如`refactor: 优化messages变量更新逻辑`）
- `docs`：文档更新（如`docs: 添加搜索功能使用说明`）
- `style`：代码格式调整（不影响逻辑，如`style: 修正缩进`）

2. 提交频率与粒度

- **小步提交**：完成一个独立子任务就提交一次（如“实现search.py的SerpApi调用”→“处理搜索结果的snippet提取”→“对接app.py的/search指令”，分3次提交），避免一次性提交大量代码（难以追溯）。

- **信息具体**：避免模糊描述（如“更新了代码”），应说明“改了什么文件、实现了什么逻辑”。
  ✕ 错误示例：`git commit -m "完善聊天功能"`
  ✓ 正确示例：`git commit -m "feat: chat.py中实现流式生成器，支持逐句返回结果"`

四、协作与代码合并：确保工作可整合

当一个功能开发完成后，需通过**Pull Request（PR）** 合并到我的核心仓库，过程如下：

1. 推送本地分支到自己的Fork仓库

```
git push origin chat-stream-A  # 将本地分支推送到自己的Fork仓库（origin对应自己的仓库）
```

2. 创建Pull Request（PR）

- 在自己的Fork仓库页面，点击“Compare & pull request”，目标分支选择组长仓库的`main`分支，填写PR描述（说明实现的功能、测试情况）。
- 组长或指定成员审核代码（检查是否符合要求、有无冲突），通过后合并到`main`分支。

3. 处理代码冲突

如果多人修改了同一文件，合并时可能出现冲突，需在本地解决：

```
# 切换到自己的功能分支
git checkout chat-stream-A
# 拉取上游main分支的最新代码
git pull upstream main
# 手动修改冲突文件（找到<<<<<<< HEAD标记的部分，保留正确代码）
# 解决后提交
git add .
git commit -m "fix: 解决与main分支的代码冲突"
git push origin chat-stream-A  # 推送更新后的分支，PR会自动同步
```

五、工作量体现技巧

1. **完整的提交链**：每个功能从设计到实现的每一步都有提交记录（如“设计接口”→“实现核心逻辑”→“添加异常处理”→“优化用户体验”），形成连贯的开发轨迹。
2. **代码质量辅助**：通过注释（解释复杂逻辑）、测试代码（如为search.py添加单元测试）体

现工作深度，这些也会在仓库中被看到。

3. **文档同步更新**：在`docs/`目录下记录自己负责功能的设计思路、实现步骤，每次更新文档也提交（`docs: 更新流式传输实现说明`），体现完整性。

六、示例：完整开发流程（以成员A实现流式传输为例）

1. 同步上游代码并创建分支：

```
git checkout main
git pull upstream main
git checkout -b stream-A
```

2. 编写代码：修改`chat.py`，将聊天函数改为生成器，支持流式输出。
3. 第一次提交：

```
git add chat.py
git commit -m "feat: chat.py中定义stream_chat生成器，逐块返回模型输出"
```

4. 继续开发：在`app.py`中对接流式函数，更新`history`变量。
5. 第二次提交：

```
git add app.py
git commit -m "feat: app.py中处理流式输出，实时更新聊天界面"
```

6. 测试并修复bug：发现流式传输偶尔中断，修改异常处理。
7. 第三次提交：

```
git add chat.py
git commit -m "fix: 修复流式传输中网络中断导致的生成器停止问题"
```

8. 推送分支并创建PR：

```
git push origin stream-A
```

在GitHub上创建PR，描述“实现流式传输功能，包含chat.py生成器和app.py界面更新逻辑，已测试通过”。

通过以上规范，你的`git log`会清晰展示“做了什么、怎么做的、改了什么”，助教可直观评估工作量，同时团队协作也更高效。核心原则：**让每一次提交都有意义，让日志成为你工作的“自**

证”。

如有任何问题，请随时联系我（whs24@mails.tsinghua.edu.cn）
当然微信私聊我也行