分工方案

针对四人小组:

本人结合**任务关联性、难度梯度和前后依赖关系**设计了以下的分工方案，供大家参考，确保高效协作且责任清晰：

核心分工原则

1. **按功能模块拆分**：将6个核心功能（+1个附加功能）划分为4个主要模块，每人负责1-2个强关联功能，减少跨模块沟通成本。
2. **考虑前后依赖**：先实现基础功能（如正常聊天、流式传输），再扩展高级功能（如搜索、文件处理），避免阻塞。
3. **平衡任务难度**：每个成员兼顾基础功能和复杂功能，避免任务分配不均。

具体分工方案

成员	负责功能模块	核心任务	技术要点	优先级
成员A	基础聊天框架（3.1+3.2）	1. 实现 `chat.py` 基础聊天函数（调用GPT-3.5-turbo） 2. 改造为流式传输生成器 3. 维护 `app.py` 中聊天记录（`messages` / `history` 更新逻辑）	1. OpenAI SDK调用 2. 生成器（generator） 3. Gradio组件状态管理	最高
成员B	网络交互（3.3+3.4）	1. 实现 `search.py`（SerpApi调用+结果处理） 2. 实现 `fetch.py`（网页爬取+HTML解析） 3. 在 `app.py` 中对接指令（`/search` / `/fetch`）	1. SerpApi接口 2. `requests` 爬取 3. BeautifulSoup解析HTML 4. 指令正则匹配	高
成员C	多媒体处理	1. 实现 `image_generate.py`	1. 图片生成API调用	中

成员	负责功能模块	核心任务	技术要点	优先级
	（3.5+3.7）	（Stable Diffusion API调用） 2. 实现`mnist.py`（LeNet模型部署+图片分类） 3. 在`app.py`中对接指令（`/image`+图片上传）	2. PyTorch模型加载 3. Gradio图片组件显示	
成员D	文件处理（3.6）+整合测试	1. 实现`pdf.py`的3个函数（摘要生成/问题构建/文本补全+流式） 2. 在`app.py`中对接文件上传和`/file`指令 3. 负责整体测试、文档编写和录屏	1. 文件IO 2. 文本处理 3. 流式输出适配 4. Git版本管理 5. 功能联调	中高

分工逻辑说明

1. **成员A：基础框架核心**
   优先实现正常聊天和流式传输，这是所有功能的基础（后续功能均依赖聊天记录管理）。成员A需确保`app.py`中的`messages`和`history`变量在基础聊天和流式场景下正确更新，为其他功能提供稳定的底层支持。
2. **成员B：网络数据获取**
   网络搜索（3.3）和网页总结（3.4）均依赖网络请求和数据解析，技术栈相似（API调用+文本处理）。成员B需重点处理指令解析（如`/search`/`/fetch`的正则匹配）和HTML中`<p>`标签提取逻辑，确保数据格式符合模型输入要求。
3. **成员C：多媒体交互**
   图片生成（3.5）和图片分类（3.7）均涉及多媒体文件处理，成员C需熟悉Gradio的图片上传组件和显示逻辑。其中，图片分类依赖第二次小作业的LeNet模型，需注意模型加载和输入预处理（如MNIST图片尺寸适配）。
4. **成员D：文件处理与整合**
   文件聊天（3.6）是最复杂的功能之一（涉及文件上传、文本处理、流式输出），成员D需

协调`pdf.py`与`app.py`的对接，确保上传TXT文件后能自动生成摘要，并支持`/file`指令的交互。同时负责最终测试，确保各功能模块兼容。

协作流程

1. **阶段一（基础搭建）**：成员A先完成3.1（正常聊天），并提交基础代码框架（`app.py`/`chat.py`），其他成员基于此框架开发。
2. **阶段二（功能并行开发）**：成员A实现3.2（流式传输），成员B开发3.3/3.4，成员C开发3.5/3.7，成员D同步设计`pdf.py`的函数接口。
3. **阶段三（整合测试）**：所有功能完成后，成员D主导联调，检查指令冲突（如`/search`与`/file`的优先级）、变量状态一致性（如`messages`是否正确更新），并编写文档和录屏。

优势

- **低耦合**：每人负责的功能模块内部关联性强，减少跨成员依赖。
- **高并行**：在基础框架完成后，成员B、C、D可并行开发，缩短整体开发周期。
- **易扩展**：若某功能延迟，可由其他成员协助（如成员D熟悉流式传输，可协助成员A调试）。

通过此分工，四人可高效协作，确保所有功能按要求实现，同时平衡工作量和技术挑战。只是初步分工方案，大家可以一起再讨论一下