****

大作业设计报告书

**题目：基于大模型API的AI对话APP**

**学 院 计算机科学与工程学院**

**专 业 网络工程**

**组长姓名学号 黄梓恩**

**202130440705**

**成员姓名学号 黄梓恩**

**202130440705**

**指导教师 张晶**

**起始日期 2024.5.14-2024.5.27**

华智对话SmartMate

1. 选题背景 ：

在当今AI技术日新月异的时代背景下，大型语言模型以其无与伦比的灵活性与智能化水平，正引领着软件行业的革新风潮。传统智能客服系统的局限性——诸如互动体验的机械性与响应内容的单一化——日益凸显，与此同时，人力成本伴随客服团队扩张而不断攀升，成为众多企业亟待解决的痛点。鉴于此，"华智对话SmartMate" 应运而生，旨在探索AI技术与移动应用深度融合的新路径。

本项目的核心目标是对接当前最为先进的通义千问与文心一言等大型语言模型API，开创性地在单一Android应用程序内实现多模型集成。这一设计不仅极大地方便了用户，使他们无需频繁切换不同应用即可享受多元化的AI服务体验，还通过技术创新，巧妙运用适配器模式，确保了用户在同一交互界面下能无缝切换至最适宜其需求的AI模型。此举不仅优化了资源利用，更展现了技术集成的高度灵活性与未来前瞻性，为AI应用的开发开辟了崭新的视野。

二**、**方案论证(设计理念)

该项目采纳了现代软件架构中的MVVM（Model-View-ViewModel）设计模式，精心构筑了一个高效、可维护的项目框架。具体而言，该架构将整个应用逻辑划分为三个核心层次：

Model（模型层）：作为数据与业务逻辑的基石，Model负责封装和管理所有的数据操作与后端交互。在此项目中，Model层不仅对接通义千问与文心一言等大模型API，处理复杂的请求与响应，还实现了数据的持久化存储与同步，确保信息的准确无误与实时更新。这一设计使得数据处理逻辑清晰独立，易于测试与维护。

View（视图层）：聚焦于用户体验的直接呈现，View负责应用的用户界面设计与交互反馈。利用Android丰富的UI组件，我们精心设计了直观易用的界面，让用户在享受流畅操作的同时，能够直观感受到AI大模型带来的智能服务。View层与数据逻辑解耦，保证了界面的灵活调整与视觉效果的优化，不被后台逻辑所牵制。

ViewModel（视图模型层）：作为连接Model与View的桥梁，ViewModel扮演着数据与UI绑定的核心角色。它从Model获取处理后的数据，并将其转换为适合View使用的形态，同时监听用户的交互事件，指挥Model进行相应的数据操作。这一层的存在，有效避免了直接在View中处理业务逻辑的弊端，提升了代码的可读性和可复用性。特别是在本项目中，ViewModel还承担了协调多个AI模型切换逻辑的重任，确保了用户在不同大模型间平滑过渡的高级体验。

此架构的选择基于其对视图与数据逻辑分离的高效实现。通过ViewModel作为中介，我们可以轻松实现数据绑定与事件处理的解耦，这不仅简化了UI更新逻辑，还使得开发者能够专注于业务逻辑的优化与创新，而不必过多考虑UI层面的细节。对于集成多个AI模型的应用而言，这样的设计大大提高了代码的模块化程度，便于后续的模型添加与功能升级。

面对不同AI大模型API的差异性，我们采用了适配器模式来统一接口，确保每个模型可以无缝融入应用中。这一方案降低了系统间的耦合度，提高了代码的灵活性与可重用性，使得切换模型如同更换插头一般简单直接，从而满足了用户多样化的需求和期望。

MVVM架构确保了各层之间的低耦合，使得修改或添加功能时的影响范围最小化。清晰的分层结构便于软件测试，每个部分都可以独立测试与优化，加速了开发流程。适配器模式支持快速接入更多AI模型，为应用的持续进化与服务丰富性提供了坚实基础。同时，通过ViewModel的高效数据管理与即时UI更新，用户能够享受到流畅无阻的交互体验，以及根据上下文选择最适合的AI模型带来的智能服务。

三**、**过程论述

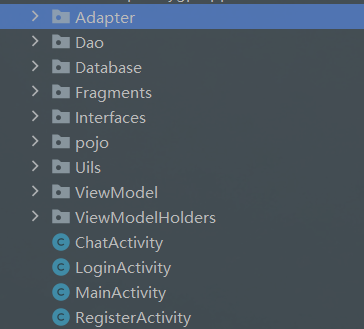


图1.项目结构

项目结构如下：

1. 数据模型（Model）：这部分主要包括数据实体类（如UserEntity、ChatInfo、Message等）、数据访问对象（DAO，如UserDao）以及数据库（如AppDatabase）。数据实体类定义了应用中使用的数据结构，DAO则提供了一种方便的方式来操作数据库中的数据。AppDatabase则是数据库的入口点，用于实例化各个表的DAO。在本项目中，绝大部分数据存储在后端主机的服务器上，因此这里的SQLite只有一个User表，用于持久化存储Token以及一些用户数据，避免反复的网络请求降低性能。
2. 用户界面（View）：这部分主要包括各种Fragment（如ChatFragment、ChatInfoFragment、ImageFragment和ToolFragment），它们定义了应用中的用户界面。此外，还包括一些工具类，如HashUtils和JsonUtil，用于处理特定的任务，如哈希和JSON解析。
3. 视图模型（ViewModel）：这部分主要包括各种ViewModel（如ChatViewModel、LoginViewModel和MessageViewModel），它们负责处理应用的业务逻辑，并提供数据给UI。ViewModel与Model和View之间进行交互，更新UI并处理用户的操作。
4. 适配器（Adapter）：这部分主要包括各种适配器类（如ChatAdapter和MsgAdapter），它们用于在RecyclerView中显示数据。
5. 接口（Interface）：这部分主要包括各种接口类（如ChatLIstItemClickListener），它们用于在不同的组件之间传递回调。
6. ViewModelHolders：这部分主要包括ViewModelHolder类，它用于在不同的组件之间共享ViewModel。ViewModel封装了各个Http请求并且持有各种信息，用于向后端服务器获取数据或者更新数据。ViewModelHolder以及各种ViewModel通过懒汉式的单例方法确保全局唯一性，使得各个不同的Activity之间通过ViewModelHolder共享同一份数据。
7. 第三方库：这个项目还使用了一些第三方库，如Glide用于加载图片，OkHttp用于网络请求，Gson用于JSON解析，Markwon用于Markdown解析等。

其中，MainActivity作为入口界面，会首先通过LoginViewModel中的autoLogin方法尝试通过数据库中的token进行自动登录，如果登录成功就会将后端返回的用户基本信息赋值给LoginView。MainActivity通过检查用户信息来判断是否成功登录，，并且导航到LoginActivity或者ChatActivity。Logi前者用于登录，又可以通过注册按钮导航至RegisterActivity；后者是程序的主界面，通过BottomNavigation来导航四个不同的Fragment，分别是ChatFragment（聊天界面）、ChatInfoFragment（聊天列表）、ImageFragment（文生图界面）和ToolFragment（用户信息）。

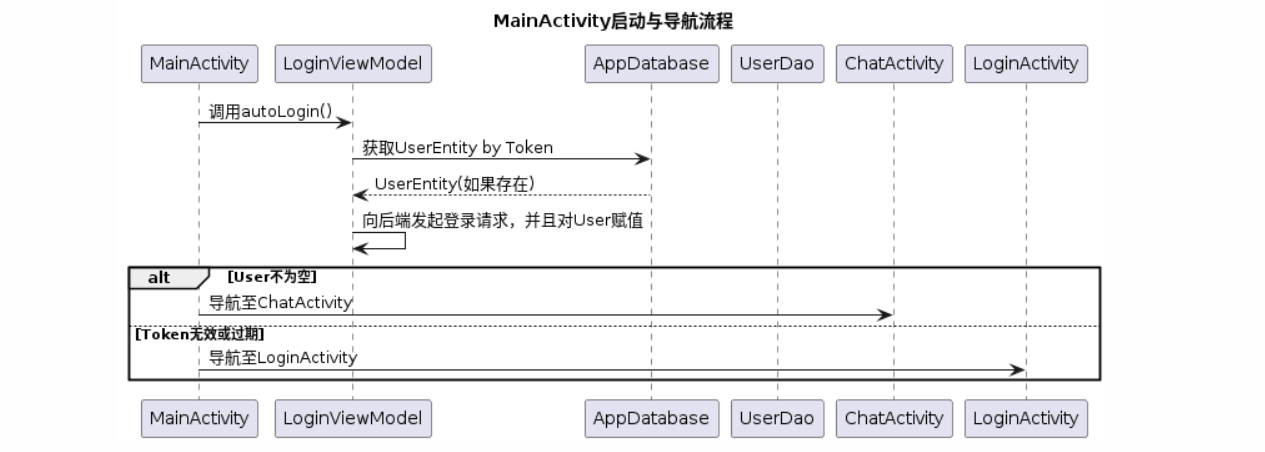


图2.启动流程

ViewModelHolder是一个精心设计的单例类，旨在全局范围内唯一地管理核心ViewModel实例，通过实现ViewModelStoreOwner接口并内置一个ViewModelStore，高效地维护这些ViewModel的状态。此设计不仅确保了ViewModel的实例在整个应用程序生命周期中的一致性和唯一性，还显著优化了内存资源的使用，避免了不必要的对象重建开销。

更进一步，ViewModelHolder所管理的每个ViewModel（如LoginViewModel、ChatViewModel及MessageViewModel）均采用懒汉式单例初始化策略，这意味着它们仅在首次访问时创建，之后则复用该实例，进一步促进了资源的有效利用。通过暴露这些ViewModel的访问方法，ViewModelHolder促进了跨组件的数据共享与通信，增强了应用的模块间协同能力。

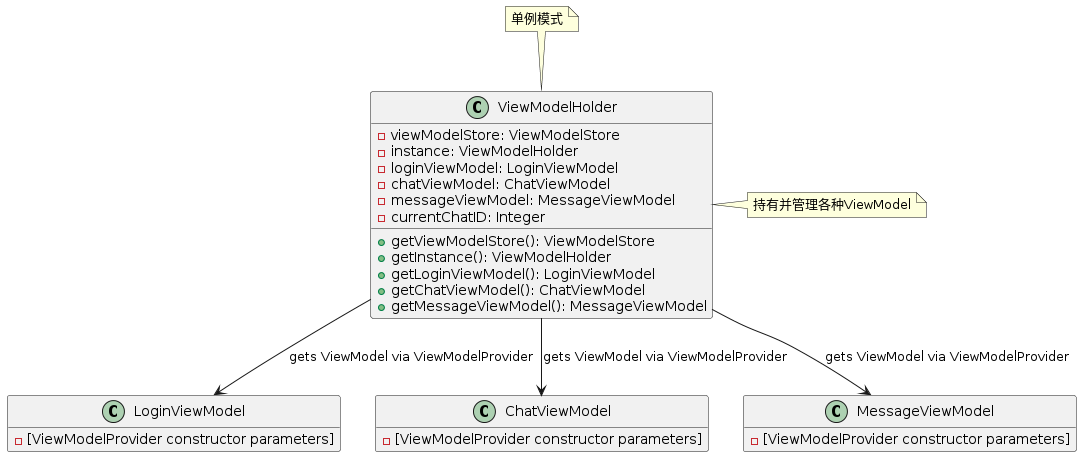
此外，ViewModel内部通常包含可观察的Mutable状态，允许UI组件通过观察者模式（如LiveData）来监听数据变化，从而自动响应式地更新界面。这一机制确保了用户界面能够及时反映出数据的任何变更，提升了应用的交互流畅度和用户体验。总之，ViewModelHolder作为架构中的核心枢纽，不仅实现了资源高效管理，还促进了应用状态管理的透明度与灵活性。

图3.ViewModel管理类图

应用程序的核心优势在于实时接收并展示来自服务器的连续数据流，这一特性在用户界面组件中得以生动体现。具体而言，在我的项目架构中，ChatActivity内嵌的ChatFragment承载了消息发送的关键功能，通过紧密集成MessageViewModel来驱动这一流程。当用户触发发送消息操作时，ChatFragment内的sendMessage()方法首先对输入内容进行合法性校验，确保消息非空，随后将控制权委托给MessageViewModel来进一步执行发送逻辑。

void sendMessage() {  
 String content = editText.getText().toString();  
 if (content.equals("")) {  
 Toast.*makeText*(getContext(), "消息不能为空", Toast.*LENGTH\_SHORT*).show();  
 return;  
 }  
 messageViewModel.sendMessage(content, getContext());  
 editText.setText("");  
}

在MessageViewModel的设计中，sendMessage方法接收两个参数：待发送的消息内容以及应用的上下文环境。上下文扮演着至关重要的角色，它不仅用来实例化AppDatabase，实现本地数据的便捷访问，还确保了数据操作远离主线程，遵循Android的线程安全原则，防止UI阻塞。通过在独立线程中执行数据库查询，既保护了用户体验，也保证了数据获取与请求发起的恰当顺序，避免了因并发操作可能导致的数据不一致问题。

public void sendMessage(String content, Context context) {  
 AppDatabase db = Room.*databaseBuilder*(context, AppDatabase.class, "user").build();  
 Thread t = new Thread(() -> {  
 UserEntity user = ViewModelHolder.*getInstance*().getLoginViewModel().getUser().getValue();  
 String token = db.userDao().getToken();  
 chatInfo.setChatID(ViewModelHolder.*currentChatID*.toString());  
 if(chatInfo.getChatID().equals("-1")) {  
 chatInfo.setSystem(user.getDefaultPrompt());  
 }  
  
 msgList.getValue().add(new Message(chatInfo.getSystem(), content, "user"));  
  
 OkHttpClient client = new OkHttpClient();  
 Request request = new Request.Builder()  
 .url("http://47.99.133.237:8080/aiChat?inputValue=" + content +  
 "&chatID=" + chatInfo.getChatID()+  
 "&chatModel=" + user.getDefaultModel() +  
 "&temperature=0.85" +  
 "&top\_p=0.8" +  
 "&system=" + chatInfo.getSystem())  
 .get()  
 .addHeader("cookie", token)  
 .build();  
  
  
 EventSourceListener listener = new EventSourceListener() {  
 int[] count = {0};  
 boolean ifClose = false;  
 @Override  
 public void onOpen(EventSource eventSource, okhttp3.Response response) {  
 super.onOpen(eventSource, response);  
 // 连接已打开  
 }  
  
 @Override  
 public void onEvent(EventSource eventSource, String id, String type, String data) {  
 super.onEvent(eventSource, id, type, data);  
 if(ifClose) {  
 // 获取ID  
 ViewModelHolder.*currentChatID* = Integer.*valueOf*(data);  
 return;  
 }  
  
 // 收到事件  
 if (data.equals("CHAT COMPLETED!")) {  
 ifClose = true;  
 return;  
 }  
 if (count[0] == 0) {  
 msgList.getValue().add(new Message(chatInfo.getSystem(), data, "assistant"));  
 count[0]++;  
 } else {  
 msgList.getValue().get(msgList.getValue().size() - 1).setContent(msgList.getValue().get(msgList.getValue().size() - 1).getContent() + data);  
 }  
 msgList.postValue(msgList.getValue());  
 }  
  
 @Override  
 public void onClosed(EventSource eventSource) {  
 super.onClosed(eventSource);  
 // 连接已关闭  
 }  
  
 @Override  
 public void onFailure(EventSource eventSource, Throwable t, okhttp3.Response response) {  
 super.onFailure(eventSource, t, response);  
 // 连接失败  
 }  
 };  
  
 EventSource eventSource = EventSources.*createFactory*(client).newEventSource(request, listener);  
 });  
  
 t.start();  
  
}

鉴于后端服务采用Server-Sent Events（SSE）技术来持续推送响应，因此可以利用EventSource建立持久连接，实时监听服务端的动态数据流。通过重写EventSource的各类回调方法，其中，onEvent回调尤为关键。每当服务端有新数据片段送达，onEvent即被激活，依据双方约定的协议解析数据，从而管理聊天对话的连续性，还能在特定标记（如“CHAT COMPLETED!”）出现时，准确无误地更新当前会话的chatID，确保用户对话上下文的连贯与精准。

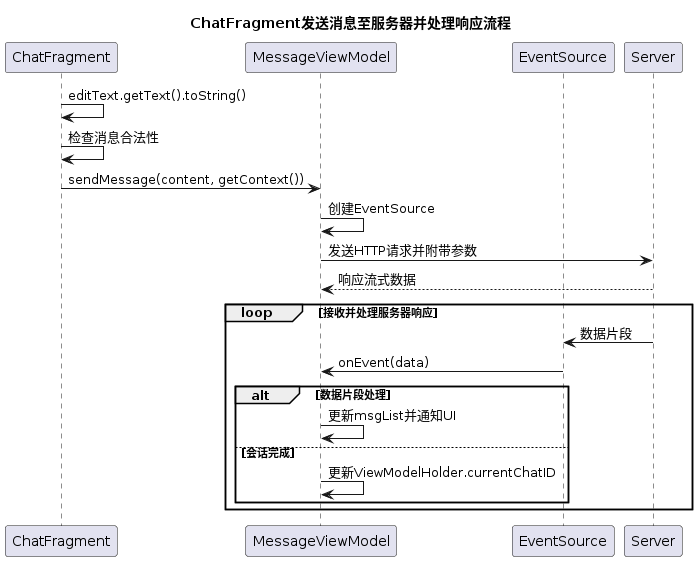


图4.消息发送与处理流程

应用程序的起始点为主活动（MainActivity），它根据用户登录状态智能导航至相应功能模块，确保用户旅程的顺畅过渡。登录状态的判断机制有效引导用户至登录页面（LoginActivity）或直接进入核心业务处理模块（ChatActivity），这种设计策略提升了用户体验的连续性。ChatActivity作为应用的核心模块，承载了用户交互的主要功能。它通过集成四个功能各异的Fragment（聊天界面、聊天信息列表、图片处理界面和工具设置界面），实现了多功能合一的综合体验。这一设计不仅丰富了应用的使用场景，也展现了高度模块化的架构理念，便于维护与扩展。

应用程序采用了一套高效的数据管理模式，核心在于ViewModelHolder的设计与实施。ViewModelHolder作为数据管理的中心枢纽，通过懒汉式单例模式确保了全局范围内ViewModel实例的唯一性，有效地促进了数据在各组件间的共享与同步。每个ViewModel封装了与后端服务器的交互逻辑及相关的数据模型，从而实现业务逻辑的集中处理与分发。

数据持久化方案依赖于Room Database框架，利用DataBase、DAO（数据访问对象）和Entity类协同工作，实现了用户Token及常用数据（如UserEntity、Message等）的本地存储。这一策略在减少网络请求频次的同时，提升了应用的响应速度与用户体验。此外，通过Pojo（Plain Old Java Object）类映射数据库表结构，进一步简化了数据模型的设计与管理。

为优化用户界面的展示效果，应用采用了Adapter模式来驱动RecyclerView，灵活高效地展示了消息列表和聊天内容。此外，自定义的Utils类集成了常用功能逻辑，如哈希处理、JSON解析等，这不仅减少了代码重复，还提高了代码的复用率和维护性。

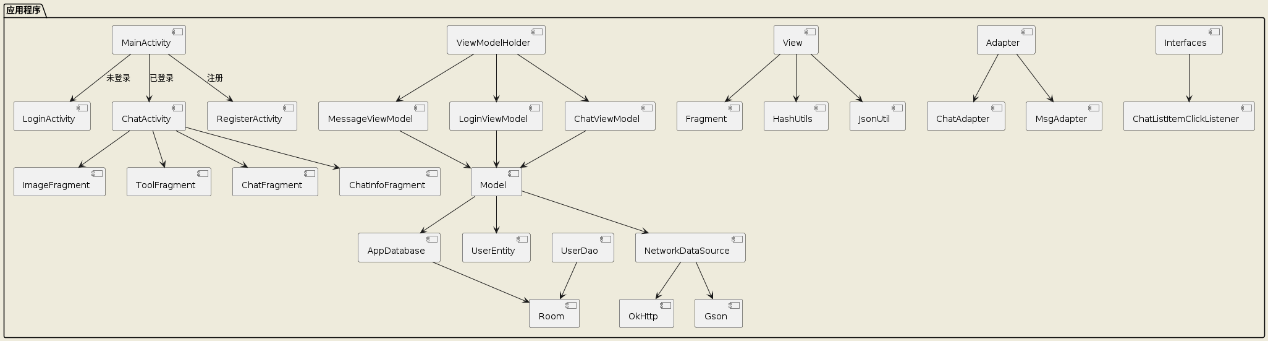


图5.架构图1

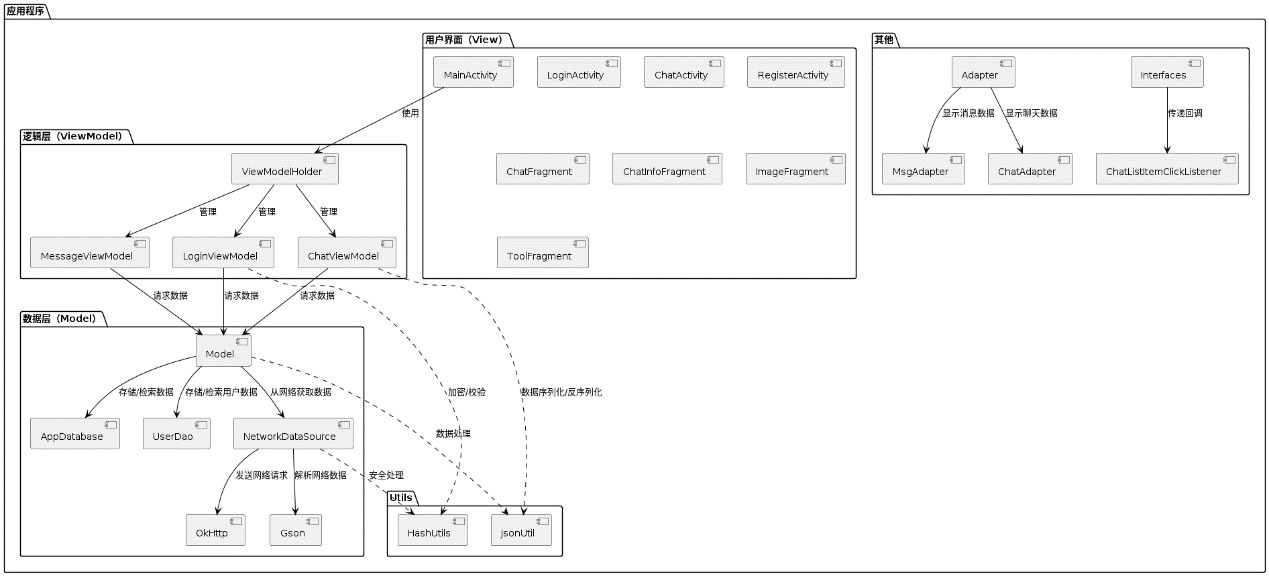


图5.架构图1

四**、**结果分析



图6.登录界面



图7.注册界面

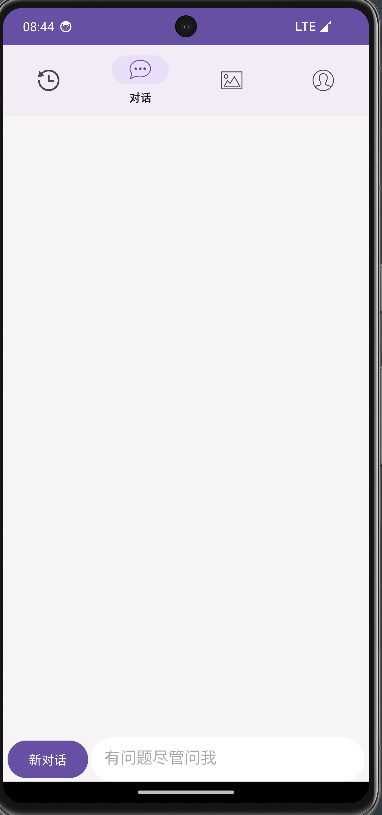


图8.登录成功后的界面

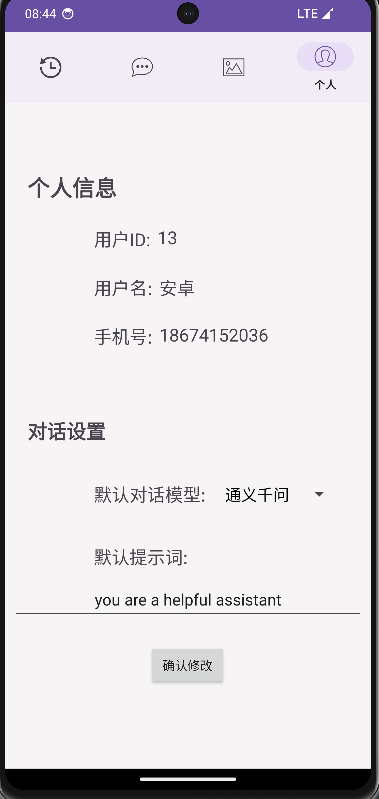


图9.个人信息界面

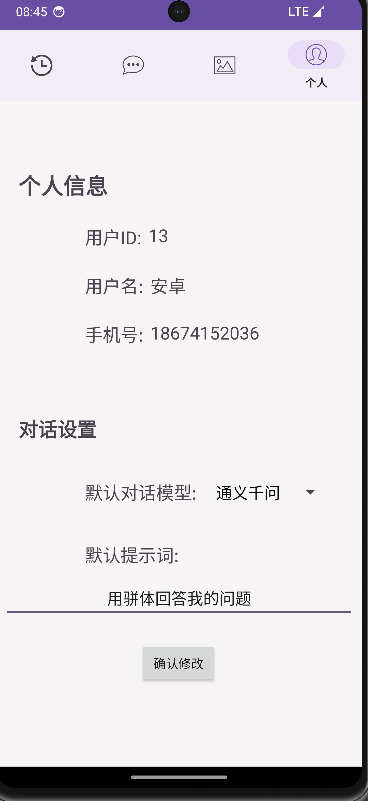


图10.修改个人信息

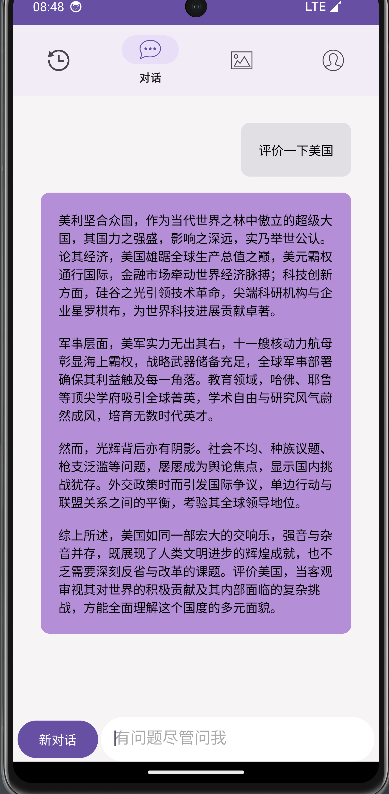


图11.对话

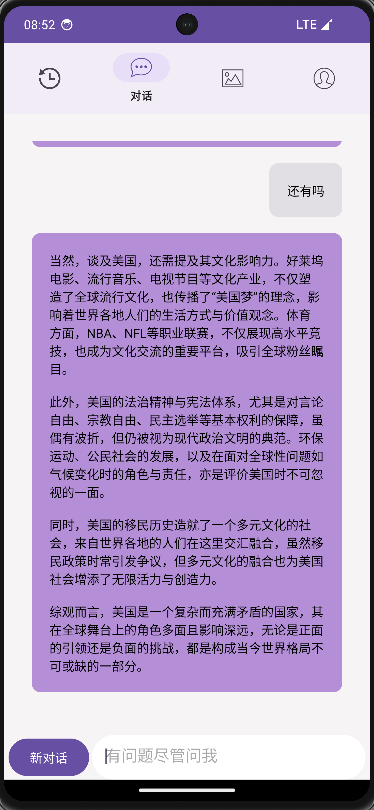


图12.多轮对话

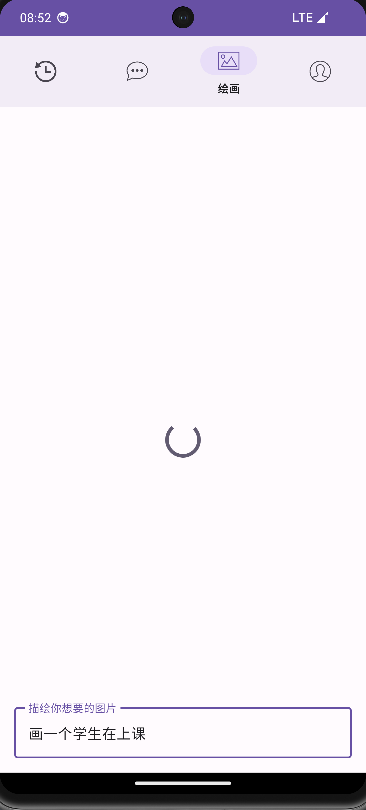


图13.等待作画



图14.作画完成

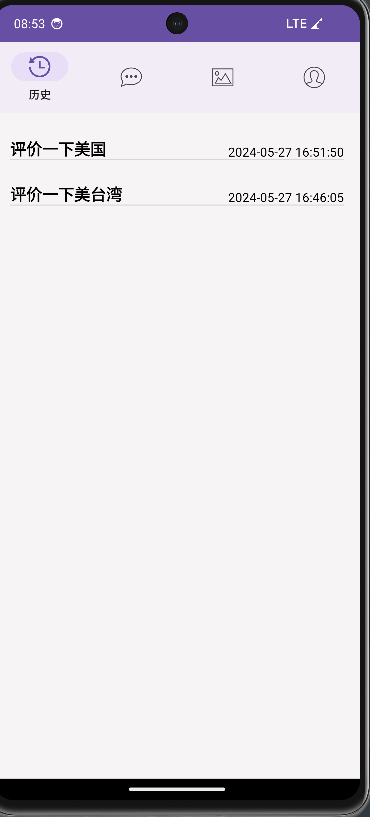


图15.浏览对话记录

五**、**软件优缺点与对未来的展望

**优点总结**

* **模型切换灵活性高**：软件设计允许用户无缝切换至不同的人工智能模型，这一特性极大地增强了用户体验，满足了针对特定领域或任务需求选择最适宜模型的个性化要求。
* **跨平台数据同步**：实现了网页端与软件客户端数据的完全同步，确保了用户在不同设备或环境下都能延续之前的对话历史，享受无缝的交互体验，提升了便捷性和用户黏性。

**缺点分析**

* **界面美观度与交互性不足**：当前软件界面设计相对朴素，缺乏吸引人的视觉元素和动态效果，这可能影响用户的初次印象及长期使用的满意度。
* **个性化缺失**：尚未提供用户自定义设置选项，如界面主题、字体大小、聊天背景等，以及模型参数调整，限制了用户的个性化定制空间。

**未来发展展望**

1. **增强用户界面与体验**：计划跟进网页端的设计更新，引入更加现代化的UI元素，增加平滑的动画和过渡效果，提升整体的视觉吸引力和使用流畅度。
2. **聊天记录管理功能**：引入聊天记录的管理工具，允许用户直接在应用内删除不再需要的对话历史，或对重要对话进行重命名归档，增强数据管理的便捷性。
3. **个性化参数配置**：集成如temperature和top\_p等高级参数设置，让高级用户能够根据个人偏好微调模型输出的随机性和确定性，增强对话的多样性和可控性。
4. **内置对话模板丰富化**：预设更多场景化的内置对话模板，无需用户手动编写prompt即可快速启动特定主题的对话，降低使用门槛，提升效率和趣味性。
5. **快捷短语功能**：开发快捷短语或常用语句功能，用户可自定义常用回复，一键发送，提高日常沟通效率。
6. **代码复制与文生图集成**：实现代码块的一键复制功能，方便开发者用户直接使用；同时，探索将文本到图像生成技术集成到对话中，为用户提供更丰富的多媒体交互形式。

综上所述，软件的发展策略着重于提升用户体验、增强个性化定制能力，并引入创新功能以拓宽应用场景，旨在打造一个更强大、更个性、更高效的交互平台。

六**、**项目成员工作内容分配

本项目由黄梓恩（202130440705）独立完成，借助先前已经搭建的后端服务器，通过API文档，只需要着眼于界面与基本处理逻辑。