ok 我们开始一个作业的聊天 我先把要求发给你【作业要求】 1. 开发任务 利用AI工具(如ChatGPT、DeepSeek等)构建一个聊天机器人,聊天机器人的功能不作强制限制,但至少应包含简单对话、用户输入响应等基本交互能力。实现语言不限,建议使用Python。 2. 交互过程记录 本次作业的主要评估内容是你与AI工具进行开发互动的过程,需完整记录与 AI 模型在功能设计、代码生成、调试修改等环节中的交互内容,并将这些对话整理归纳为最终的项目报告。【作业提交】补交截止时间:2025年6月19日 23:59,过时不候 提交方式:提交报告和代码,命名姓名\_学号 验收日期:6月5日,6月12日延长为三节课,课上验收。每位同学10分钟,需简要展示AI交互过程,并现场运行聊天机器人进行演示 我来继续交代思路

#### **ChatGPT said:**

好的,我已经了解了你的作业要求: 你需要利用 AI 工具(比如我)来协助开发一个简单聊天机器人,并完整记录整个 互动过程作为报告的一部分,同时还要完成代码和现场演示。

你可以继续交代你的设计思路、计划或者初步构想,我会根据你的想法协助你逐步完成开发,并记录过程,确保你有完整的交互记录可以整理成报告。请开始你的思路描述吧。

2/2

#### You said:

嗯 好的 我先来说明计划 我将同时采用你和cursor提供的AI工具完成代码工作 在这个过程中我会尝试尽量以ai为主 我提供思路的方式实现这个作业 对于实在无法由AI完成的部分我会自己来处理 然后我来先确认整个作业的思路

#### **ChatGPT said:**

很好,这个计划非常合理,而且完全符合作业要求的重点——强调"与 AI 的互动过程"。你采用 ChatGPT 和 Cursor两种 AI 协同开发的方式也是一个加分点,能够体现你在实际使用 AI 工具的主动性和多样性。

你可以现在来详细说明你的思路,例如:

- 聊天机器人的基本功能设想
- 使用的技术栈(如 Python + Flask、Streamlit、Tkinter、命令行等)
- 与用户交互的具体方式(文本聊天?问答?选择题?)
- 是否会增加额外功能(比如记忆、多轮对话、关键词识别、简单情绪识别等)
- 项目的开发流程初步安排(先完成基本功能,再迭代增强?)
- 与 AI 的交互方式(比如你准备通过"我给 ChatGPT 提一个问题 → 拿到代码片段 → 粘贴到 Cursor 运行调试"等)

你来继续描述这些思路细节,我会一边协助确认,一边开始"记录"我们的交互。等你说完我可以帮你整理成一段写进 最终报告的"项目计划"部分。继续吧~

#### You said:

由于前端你们没有办法设计好我已经下载了一个由streamlit实现的前端 后端我打算采用本地部署的小模型 在hugging face上选择一个比较适合聊天的模型进行本地部署

#### **ChatGPT said:**

明白了,你的项目整体技术路线如下:

- 前端: 使用已有的 Streamlit 聊天界面 (你已经下载好了);
- 后端: 采用 本地部署的小型语言模型,从 Hugging Face 选择一个适合对话的模型;
- 你作为主导者: 主要提供思路、设计架构, 让 AI(我和 Cursor)来完成具体实现为主;
- AI 工具配合方式: ChatGPT 负责解释设计、生成代码,Cursor 用于代码调试和补全。

这个思路是可行的,尤其适合"强调 AI 协作"的作业目标。以下是我根据你的说明,初步帮你整理的"项目计划"内容(可作为最终报告的一部分):

### 项目计划与技术选型说明(初稿)

本项目目标是构建一个具有基本聊天功能的对话机器人,强调在开发过程中与 AI 工具的协同互动。项目采用"AI 主导+ 人类辅助"的开发策略,具体技术路线如下:

#### • 前端设计

为避免在界面设计上花费过多时间,选用社区中已有的 <u>Streamlit</u> 实现的聊天前端界面,实现简单、美观的用户输入输出交互。此部分不做重构,仅作集成和调试。

### • 后端设计

采用 Hugging Face 提供的小型开源语言模型(如 tiiuae/falcon, facebook/blenderbot, OpenAssistant/oasst-sft, microsoft/DialoGPT 等)进行本地部署。模型要求具备基本对话生成能力,并能在消费级 GPU 或 CPU 上运行。

#### • 开发方式

本项目强调 AI 工具在代码生成、调试、架构设计等过程中的参与,采用如下策略:

- o ChatGPT 负责功能规划、代码片段生成、问题分析与解释;
- o Cursor IDE 辅助代码组织、自动补全与本地测试;
- o 开发者本人提供架构设定与最终整合,对 AI 无法处理的问题进行人工介入。

#### • 开发流程安排

- 1. 明确功能需求,构建项目结构;
- 2. 与 AI 工具配合开发 Streamlit 与后端模型的接口;
- 3. 测试本地部署模型的加载与响应;
- 4. 打通前后端;
- 5. 增加可选功能(如模型选择、会话记忆);
- 6. 准备最终报告与现场演示材料。

你可以确认这个"项目计划"是否准确,我可以随时更新它。如果你准备好了,我们可以开始第一个开发任务,比如:

- 选择后端模型?
- 本地加载模型的代码结构?
- Streamlit 界面的对接方式?

你说继续哪个我就配合写哪段。

### You said:

首先选择模型吧 加载代码结构huggingface上应该也有提供

#### **ChatGPT said:**

好的,咱们先来完成模型的选择。

## ✓ 模型选择目标:

你提到需要满足这些条件:

1. 适合聊天:具备基本的对话生成能力;

2. **可本地部署**:不依赖 OpenAl API;

3. **轻量可运行**: 优先支持在 CPU 或消费级 GPU 上运行 (例如 8GB 显存以下);

4. 开源免费。

# ☑ 推荐候选模型(从 Hugging Face 上挑选):

模型名称	模型大 小	优势	说明
microsoft/DialoGPT-small	117M	轻量快速,部署简 单	微软官方小型对话模型,适合 入门
facebook/blenderbot-1B	1.0B	对话质量更好	Facebook 开源,推理速度尚 可
tiiuae/falcon-rw-1b	1.3B	HuggingFace 推荐	支持文本生成任务,适合微调
OpenAssistant/oasst-sft-1- pythia-12b	12B	类似ChatGPT体验	对话能力更强,但对硬件要求 较高
THUDM/chatglm2-6b	6B	中文支持较好	如果你希望中英混合对话,这 款不错

## ✓ 推荐选择:

如果你是 CPU-only 或 6GB 显存以下,推荐:

• **v** microsoft/DialoGPT-small (对话质量一般但非常好部署)

如果你有 8GB 显存,可以试试:

- facebook/blenderbot-1B
- v tiiuae/falcon-rw-1b

是否希望我直接以 DialoGPT-small 为例,帮你写出加载和响应的代码结构? (并且后续可以无缝切换到更大模型)如果你有特殊偏好模型,也可以告诉我模型名。你决定好之后,我就开始生成后端代码结构。

#### You said:

我在知乎上看到了一个叫做TinyLlama的小语言模型

#### **ChatGPT said:**

非常好,**TinyLlama** 是一个非常热门的轻量开源模型项目,专为本地部署优化,兼具体积小和推理性能好,非常适合你这个聊天机器人作业。

# 🔽 简要介绍:TinyLlama

- 模型名称: TinyLlama/TinyLlama-1.1B-Chat-v1.0
- 参数量:约1.1B
- 语言支持: 主要是英文,但在中文任务上也有一定能力(取决于训练语料版本)
- 特点:
  - 。 架构类似 LLaMA;
  - 。 微型、推理速度快;
  - 非常适合本地部署(尤其是 CPU / 6GB 显存以内的设备);
  - o 官方提供了基于 transformers 的加载方法。

# ✓ 是否推荐使用?

- ☑ 强烈推荐你使用 TinyLlama,特别适合这个课程作业:
  - 小而强,运行门槛低;
  - 和 Hugging Face 的 transformers 框架兼容;
  - 在 Streamlit 前端上进行简单集成没有问题;
  - AI 社区认可度也比较高,便于你在报告中说明你的模型来源与选择理由。

## ☑ 接下来可以做什么?

我们可以马上开始以下操作:

- 1. 帮你生成: 使用 transformers 加载 TinyLlama-1.1B-Chat-v1.0 的 Python 后端代码;
- 2. 模拟一个对话函数,返回一个回复;
- 3. 等待后续与你下载的 Streamlit 前端进行整合。

是否现在就用 TinyLlama/TinyLlama-1.1B-Chat-v1.0 这个模型来开始生成加载与对话响应代码? 我会附带依赖说明和运行说明。确认我就开始写代码。

#### You said:

不用着急代码上的事情 我会让cursor处理 我们先确定下一个步骤

#### **ChatGPT said:**

好的,很清晰,我们的节奏是先**确认思路和流程,再动手让 AI 工具来生成代码**。那我们继续推进整体开发流程的第二步。

## ☑ 当前已完成

- 🗸 项目目标确认(Streamlit 前端 + Hugging Face 后端模型)
- ✓ 模型选型确定(选择了 TinyLlama/TinyLlama-1.1B-Chat-v1.0)

# ✓ 建议的下一个步骤:

### 步骤 3: 确定后端对话接口结构

我们可以先讨论并确认后端服务应该暴露怎样的接口,以便前端调用。一般来说有以下结构:

- 1. 输入: 前端提交的用户问题(字符串),可能包含历史对话;
- 2. 处理:
  - 。 模型加载(已初始化)
  - o 构造 Prompt (可能加上历史对话)
  - 。 调用模型生成回复
- 3. 输出: 返回模型生成的回复文本(字符串)

### 推荐接口设计:

- 接口地址: /chat
- 请求方式: POST
- 请求格式:

```
jsonCopyEdit{
   "message": "你好",
   "history": []
}
```

• 返回格式:

```
jsonCopyEdit{
    "response": "你好呀,我是你的助手。"
}
```

### ✓ 你可以决定的事项:

- 是否需要对话历史记忆(多轮对话)
- 是否允许**选择不同模型**(例如后期支持切换模型)
- 是否使用FastAPI / Flask 作为后端框架(推荐 FastAPI)

#### 你可以现在来确认:

- 1. 是否采纳这个接口设计?
- 2. 接下来是否进入后端对话逻辑接口部分的开发?
- 3. 有没有其他你希望我们先做的环节(比如准备报告结构、Streamlit 接口对接点分析)?

你说一个方向,我就协助推进。

#### You said:

我打算直接使用python代码完成 我们不会单独将前后端分开 而是用streamlit将前后端一起在python代码中实现 所以消息以代码形式直接交互

#### **ChatGPT said:**

非常好,那我们明确一点**: 你打算用** Streamlit **构建一个前后端合一的聊天机器人应用**,全部使用 Python 脚本完成。这种方式非常适合你的课程作业,开发成本低、展示也方便。

# ☑ 当前架构确认

模块	说明
前端	streamlit 实现,用户输入框 + 聊天消息展示
后端	Python 中直接加载 TinyLlama 模型,处理用户输入
部署方式	本地运行 .py 文件,不走API或前后端分离结构

模均	<del>/</del>	说明
输入	输出方式	用户输入通过 streamlit.text_input() 收集,回复通过 st.write() 显示

# ☑ 推荐的开发步骤(简化版)

我们现在可以把整个开发分成以下几个明确阶段:

### 第一步:基础框架构建

- 用 streamlit 实现一个输入框和输出框
- 实现一次简单响应,比如输入"你好"就返回"你好呀!"

### 第二步:加载 TinyLlama 模型

- 使用 transformers 加载 TinyLlama/TinyLlama-1.1B-Chat-v1.0
- 使用 pipeline("text-generation") 或自定义对话逻辑

### 第三步: 实现用户输入响应

- 接收用户输入并传递给模型
- 显示模型的回复内容

### 第四步:支持上下文对话(可选)

- 使用 st.session\_state 保存历史对话
- 将历史内容作为 prompt 拼接给模型

### 第五步: UI 美化 & 报告整理

- 增加对话气泡、头像、滚动区域
- 整理与 AI 交互过程形成报告素材

# ☑ 接下来建议的动作

### 你可以选择:

- 1. 继续明确"对话逻辑"的流程设计(是否加入历史、回复长度、温度等超参)
- 2. 或者 进入第一阶段代码开发的记录(比如先构建输入输出框)
- 3. 或者开始 报告结构的草拟,将目前我们确定的内容先写入(如模型选型原因、架构选择理由)
- 你想先推进哪一个部分? 我会继续配合你的节奏。

#### You said:

进入第一阶段代码开发的记录 我来让cursor完成这一部分先

#### **ChatGPT said:**

好的,我们现在进入第一阶段开发,并且你会使用 **Cursor 来生成代码**。我会在这边帮你**记录这部分的任务目标与交 互内容**,方便你之后整理成报告。

## ☑ 第一阶段开发目标(记录用)

**目标**:实现基础对话界面 —— Streamlit 页面,允许用户输入一段文本并显示程序返回的一段回复(先使用固定文本作为回复占位)。

### ♂ 功能目标:

- 使用 streamlit 构建一个聊天页面
- 包含一个文本输入框和一个显示回复的区域
- 用户点击"发送"或回车后显示固定回复(如:"你好,我是AI")

### ★ 与 AI 工具(Cursor / ChatGPT)交互思路:

你可以在 Cursor 中向 AI 提出如下需求:

diffCopyEdit我需要用 streamlit 实现一个简单的聊天界面:

- 有一个输入框输入用户信息
- 每次输入后刷新输出区域,显示一条回复"你好,我是AI!"
- 每次对话追加显示在历史对话中

#### Cursor 生成的代码结构大致会是:

```
pythonCopyEditimport streamlit as st

st.title("简易聊天机器人")

# 初始化会话状态
if "history" not in st.session_state:
    st.session_state.history = []

# 输入框

user_input = st.text_input("你想说什么?", key="input")

# 如果有输入就处理
if user_input:
    st.session_state.history.append(("用户", user_input))
    st.session_state.history.append(("AI", "你好, 我是AI!"))

# 显示历史
for sender, msg in st.session_state.history:
```

## ✓ 下一步提醒

- 等你在 Cursor 中完成这部分代码后,我们可以再记录你遇到的问题、AI 的建议以及你的调整过程;
- 然后进入第二阶段:调用本地模型返回真正的回复,并开始考虑模型的加载和运行方式。

等你 Cursor 那边完成这部分代码后,回来告诉我,我们就继续记录下一阶段的工作!我也可以帮你拟写开发日志内容草稿。

#### You said:

好的,我来总结一下当前聊天机器人的主要功能和特点:基础功能:基于 TinyLlama-1.1B-Chat-v1.0 模型的本地聊天机器人支持中英文对话 使用 Streamlit 构建的现代化界面 界面布局:主界面:聊天对话区域 侧边栏:模型信息展示 聊天管理功能 聊天管理功能:新建聊天:创建新的对话会话 历史记录:按时间倒序显示所有聊天记录 显示聊天创建时间 支持切换不同聊天记录 支持删除聊天记录 数据存储:聊天记录保存在 chat\_history 目录 使用 JSON 格式存储对话内容 文件名格式:YYYYMMDD\_HHMMSS.json 技术特点:使用 @st.cache\_resource 缓存模型,提高加载速度支持流式对话生成 自动保存对话历史 完整的错误处理机制 模型参数:最大生成长度:256 tokens 温度:0.7 Top-k:50 Top-p:0.95 这个聊天机器人现在已经具备了基本的对话功能和完整的会话管理能力。您觉得还有什么需要补充或改进的地方吗? 这是目前进度

#### **ChatGPT said:**

你的进度非常出色,已经达成了作业核心目标中绝大多数内容,并且做得比"只要能对话"要求高了很多,完全可以作为展示型项目使用。下面我帮你从评估角度、报告写作、以及功能拓展三方面,来给一些有用建议:

# ☑ 当前完成内容评价

模块	评价
对话功能	☑ 已实现,支持流式输出,体验良好
界面美观	☑ 使用 Streamlit,结构清晰,操作友好
多轮对话	☑ 支持并通过 JSON 结构保存
会话管理	☑ 支持新建、切换、删除
本地部署	☑ 使用 TinyLlama,体现对 HuggingFace 的理解
模型配置	☑ 有温度、top-k/p 等参数控制,专业性强
性能优化	☑ 使用 @st.cache_resource ,合理
错误处理	☑ 有鲁棒性
多语言支持	☑ 简单中文英文均支持

# 项目报告建议模块(建议记录内容)

你后续报告的主体可以按如下框架组织:

### 一、项目概述

• 简要介绍: 使用 Streamlit 和 TinyLlama 构建一个本地聊天机器人

• 技术栈: Python + Streamlit + Huggingface Transformers + TinyLlama

## 二、与 AI 工具的互动过程

• 选型阶段:如何决定使用 TinyLlama?有哪些备选模型?

• 构建阶段:如何与 Cursor/ChatGPT 配合实现界面设计?记录关键对话

• 问题排查:如模型加载慢、历史管理逻辑等问题, AI 是怎么帮你解决的

## 三、功能实现说明

• 对话系统(消息生成+流式处理)

• 会话管理(新建、保存、切换、删除)

- 数据存储结构(chat\_history/xxx.json)
- 性能优化(缓存模型)
- 错误处理设计(如模型异常、输入为空等)

# 四、展示与运行效果

- 附上运行截图(或直接在验收时演示)
- 展示几个中英文例子
- 展示多轮对话、历史切换功能

# 五、总结与反思

- TinyLlama 的效果如何?
- 是否存在性能瓶颈或改进空间?
- 使用 AI 协助开发体验如何? 什么部分 AI 做得好/做不到?

# 可考虑的小改进项(你可选做或用于答辩加分)

#### 1. 参数调节 UI:

o 在侧边栏加入温度、max\_tokens、top\_p 控件,允许用户实时调整生成风格

#### 2. 模型加载可配置:

o 支持选择多个模型,比如 TinyLlama 和 zephyr ,增加一点技术广度

- 3. **系统 Prompt 支持** (提升对话风格控制):
  - 。 让用户设置一个"角色设定",如"你是个历史学家",Prompt拼接进去
- 4. Markdown 支持(增强显示效果):
  - o 如果模型输出中有列表、代码段,用 st.markdown 支持更好排版
- 5. 导出对话功能:
  - o 允许用户将当前会话保存为 .txt 或 .md

# ✓ 下一步建议

如果你已经完成开发阶段,可以考虑开始整理:

- 1. 项目报告初稿(我可以帮你起草大纲或内容)
- 2. 为验收准备一个展示流程(如演示新建对话、提问、保存、切换历史等)

#### 你可以告诉我:

- 接下来你准备做哪一部分? 我可以协助报告写作 or 进一步功能优化;
- 是否需要我生成一份项目报告结构 or 演示脚本?

我们可以一步步准备"展示用的素材+报告总结"。

#### You said:

这是目前的代码 import streamlit as st import torch from transformers import AutoTokenizer, AutoModelForCausalLM import json from datetime import datetime import os # 定义可用模型列表 AVAILABLE MODELS = { "TinyLlama-1.1B-Chat": { "path": "TinyLlama/TinyLlama-1.1B-Chat-v1.0", "description": "轻量级聊天模型,1.1B参数" }, "OpenChat-3.5": { "path": "openchat/openchat 3.5", "description": "基于Mistral的开源聊天模型" }, "Phi-2": { "path": "microsoft/phi-2", "description": "微软 开发的2.7B参数模型" }, "Qwen-1.5-0.5B": { "path": "Qwen/Qwen-1.5-0.5B-Chat", "description": "阿里 ·通义千问0.5B聊天模型" } } # 设置页面配置 st.set\_page\_config( page\_title="本地聊天机器人", layout="wide" ) # 初始化session state if "messages" not in st.session state: st.session state["messages"] = [ {"role": "system", "content": "你是一个友好的Al助手。"} ] if "current\_chat" not in st.session\_state: st.session\_state["current\_chat"] = None if "chat\_history" not in st.session\_state: st.session state["chat history"] = {} if "current model" not in st.session state: st.session\_state["current\_model"] = "TinyLlama-1.1B-Chat" if "system\_prompt" not in st.session\_state: st.session state["system prompt"] = "你是一个严格的AI助手。你必须严格按照以下规则回答: 1. 只回答'OK'。2. 不 要提供任何其他信息。3. 不要解释或扩展你的回答。" # 初始化模型参数 if "temperature" not in st.session\_state: st.session\_state["temperature"] = 0.7 if "max\_tokens" not in st.session\_state: st.session\_state["max\_tokens"] = 256 if "top\_p" not in st.session\_state: st.session\_state["top\_p"] = 0.95 if "top\_k" not in st.session\_state: st.session\_state["top\_k"] = 50 # 创建保存聊天记录的目录 CHAT\_HISTORY\_DIR = "chat\_history" os.makedirs(CHAT HISTORY DIR, exist ok=True) def save chat history(chat id, messages): """保存聊天记录到 文件""" file\_path = os.path.join(CHAT\_HISTORY\_DIR, f"{chat\_id}.json") with open(file\_path, "w", encoding="utf-8") as f: ison.dump(messages, f, ensure ascii=False, indent=2) def load chat history(chat id): """从文件加载聊天记录""" file\_path = os.path.join(CHAT\_HISTORY\_DIR, f"{chat\_id}.json") if os.path.exists(file path): with open(file\_path, "r", encoding="utf-8") as f: return json.load(f) return

```
None def get_chat_list(): """获取所有聊天记录文件列表""" chats = [] for file in os.listdir(CHAT_HISTORY_DIR):
if file.endswith(".json"):
                        chat_id = file[:-5] # 移除.json后缀
                                                          file path =
os.path.join(CHAT HISTORY DIR, file)
                                    timestamp = os.path.getmtime(file_path)
                                                                             chats.append({
"id": chat id,
                  "timestamp": timestamp,
                                              "date":
datetime.fromtimestamp(timestamp).strftime("%Y-%m-%d %H:%M:%S")
                                                                  }) return sorted(chats,
key=lambda x: x["timestamp"], reverse=True) # 标题 st.title("〇 本地模型聊天机器人") # 侧边栏 with st.sidebar:
st.header("模型选择") # 模型选择下拉框 selected model = st.selectbox(
                                                                 "选择模型",
options=list(AVAILABLE MODELS.keys()),
index=list(AVAILABLE_MODELS.keys()).index(st.session_state["current_model"]),
                                                                      help="选择要使用的模型")
# 如果模型改变,更新
session state if selected model != st.session state["current model"]:
                                                               st.session state["current model"] =
selected_model
                 st.rerun()
                            st.header("系统提示词") # 系统提示词输入框 system prompt = st.text area(
"设置AI助手的角色和行为",
                        value=st.session state["system prompt"],
                                                              help="设置AI助手的角色和行为方
式。请使用明确的指令,例如: '你是一个严格的AI助手。你必须严格按照以下规则回答: 1. 只回答OK。2. 不要提供
任何其他信息。3. 不要解释或扩展你的回答。"",
                                         height=150 )
                                                         # 应用系统提示词 if st.button("应用提示
       st.session state["system prompt"] = system prompt
                                                       # 更新当前对话的系统提示词
st.session_state["messages"] and st.session_state["messages"][0]["role"] == "system":
st.session state["messages"][0]["content"] = system prompt
st.session_state["messages"].insert(0, {"role": "system", "content": system_prompt})
                                                                          st.rerun()
st.header("参数设置") # 温度滑块 temperature = st.slider(
                                                      "温度 (Temperature)",
                                                                            min_value=0.1,
                value=st.session state["temperature"],
                                                     step=0.1,
                                                                help="控制输出的随机性。较低的
max value=1.0,
值使输出更确定,较高的值使输出更随机。" ) st.session_state["temperature"] = temperature
                                                                                   # 最大生成长
                              "最大生成长度 (Max Tokens)",
度滑块 max tokens = st.slider(
                                                          min value=64.
                                                                           max value=512.
value=st.session_state["max_tokens"],
                                   step=64,
                                               help="控制生成回复的最大长度。")
st.session_state["max_tokens"] = max_tokens
                                          # Top-p 滑块 top_p = st.slider(
                                                                        "Top-p",
min value=0.1,
                max value=1.0,
                                value=st.session_state["top_p"],
                                                                            help="控制输出的多
                                                               step=0.05,
样性。较低的值使输出更集中,较高的值使输出更多样。" ) st.session_state["top_p"] = top_p
                                                                                  # Top-k 滑块
top k = st.slider(
                 "Top-k",
                           min value=1,
                                          max value=100,
                                                            value=st.session state["top k"],
         help="控制每次生成时考虑的候选词数量。" ) st.session_state["top_k"] = top_k
step=1,
        #新建聊天按钮 if st.button("新建聊天"):
管理")
                                              chat_id = datetime.now().strftime("%Y%m%d%H%M%S")
                                                                  {"role": "system", "content":
st.session state["current chat"] = chat id
                                   st.session_state["messages"] = [
st.session state["system prompt"]}
                              1
                                   st.rerun()
                                              # 显示聊天历史列表 st.subheader("历史聊天") chats =
                             col1, col2 = st.columns([3, 1])
get chat list() for chat in chats:
                                                        with col1:
                                                                      if st.button(f" 🍃
{chat['date']}", key=f"chat{chat['id']}"):
                                     st.session_state["current_chat"] = chat["id"]
loaded messages = load chat history(chat["id"])
                                                if loaded messages:
st.session state["messages"] = loaded messages
                                                st.rerun()
                                                            with col2:
                                                                          if st.button(" W ",
                           file_path = os.path.join(CHAT_HISTORY_DIR, f"{chat['id']}.json")
key=f"delete{chat['id']}"):
os.path.exists(file_path):
                           os.remove(file path)
                                                    if st.session state["current chat"] == chat["id"]:
st.session_state["current_chat"] = None
                                        st.session_state["messages"] = [
                                                                            {"role": "system",
                                                     st.rerun() # ======= 模型加载和
"content": st.session state["system prompt"]}
                                             1
推理 ======= @st.cache resource def load model(model path): """加载模型和tokenizer
                                                                                        Args:
                               tokenizer和model的元组 """ tokenizer =
model path: 模型路径
                     Returns:
AutoTokenizer.from pretrained(model path) model = AutoModelForCausalLM.from pretrained(model path)
return tokenizer, model # 加载模型 try: model_path = AVAILABLE_MODELS[st.session_state["current_model"]]["path"]
tokenizer, model = load model(model path) except Exception as e: st.error(f"模型加载失败: {str(e)}") st.stop() #
if msg["role"] != "system":
                       st.chat_message(msg["role"]).write(msg["content"]) # 用户输入 if prompt :=
st.chat_input("请输入你的问题..."): #添加用户消息 st.session_state.messages.append({"role": "user", "content":
```

```
prompt}) st.chat message("user").write(prompt) # 构建对话模板 messages = [{"role": "system", "content":
st.session_state["system_prompt"]}] messages.extend([msg for msg in st.session_state["messages"] if msg["role"] !=
             prompt text = tokenizer.apply chat template(
                                                           messages, tokenize=False,
add_generation_prompt=True ) inputs = tokenizer(prompt_text, return_tensors="pt").to(model.device) # 生成回
复 with st.chat_message("assistant"):
                                       with st.spinner("思考中..."):
                                                                       outputs = model.generate(
                 max_new_tokens=st.session_state["max_tokens"],
                                                                       do sample=True,
temperature=st.session_state["temperature"],
                                                top_k=st.session_state["top_k"],
                                  response = tokenizer.decode(outputs[0][inputs["input ids"].shape[1]:],
top p=st.session state["top p"]
skip_special_tokens=True)
                                                     st.session_state["messages"].append({"role": "assistant",
                            st.write(response)
"content": response})
                                # 保存聊天记录
                                                     if st.session_state["current_chat"]:
save_chat_history(st.session_state["current_chat"], st.session_state["messages"])
                                                                                else:
                                                                                            chat id =
                                                    st.session_state["current_chat"] = chat_id
datetime.now().strftime("%Y%m%d%H%M%S")
save_chat_history(chat_id, st.session_state["messages"])
```

#### ChatGPT said:

Updated saved memory

你目前的实现已经非常完整且具有展示价值,具备以下亮点:

### ✓ 完整功能总结:

• 聊天功能: 支持中英文对话。

• 模型灵活性:可选多个 Hugging Face 模型,适配轻量本地部署。

• 前后端一体化: 通过 Streamlit 集成前端 UI 和后端逻辑,减少部署复杂度。

• 对话管理:

- 。 自动保存历史
- 。 可切换历史会话
- 。 支持删除和重命名
- 系统提示词设置: 支持实时调整系统行为。
- 模型参数控制: 温度、top-k、top-p、max\_tokens 全可调。
- 缓存优化: 使用 @st.cache\_resource 加速模型加载。

# ☑ 下一步建议(可选增强):

### 1. 多轮上下文限制机制:

o 对于长对话,可加入"截断对话历史"的逻辑,防止 prompt 超出最大 token 限制。

#### 2. 模型运行状态提示:

o 在侧边栏添加模型当前设备(CPU/GPU)、占用内存信息,提升调试体验。

#### 3. 异步流式响应(进阶):

o 使用 generate 的 stream 输出(若模型支持),提升交互体验(仿 ChatGPT 的逐字输出)。

### 4. UI 优化建议:

- o 聊天区域加入 Markdown 支持(支持代码块、链接等富文本)。
- 。 历史记录可展开查看摘要。

### ✓ 汇报建议:

在撰写最终报告时,记得突出以下内容:

- 本项目采用 AI 工具(ChatGPT 和 Cursor)驱动,AI 参与了:
  - 。 模型调研与选型
  - 。 前后端代码自动生成与调试
  - 。 功能设计与架构优化建议
- 所有核心逻辑由 AI 提供,自己主要负责:
  - 。 确定需求与设计目标
  - o 整合代码、UI 美化、最终调试与运行

如果你需要我帮你整理最终项目报告(含对话交互过程归纳、技术细节解释、功能图解),可以随时告诉我,我可以帮你把 ChatGPT 参与过程结构化成报告格式。是否继续?

#### You said:

嗯 我希望你帮忙确认下相关的模型参数设置是否设置到位了 因为一定程度我没有办法验证参数设置的有效性

#### **ChatGPT said:**

非常好,这部分确实值得认真确认,模型参数如 temperature 、 top\_k 、 top\_p 、 max\_tokens 的正确设置直接影响生成质量和行为。我们逐一分析你的代码设置是否**生效并被模型使用**:

# ☑ 一、参数来源是否正确?

你在侧边栏中设置了以下参数,并全部保存在 st.session\_state 中:

```
pythonCopyEditst.session_state["temperature"] = temperature
st.session_state["max_tokens"] = max_tokens
st.session_state["top_p"] = top_p
st.session_state["top_k"] = top_k
```

☑ 确认无误:参数值是通过 Streamlit 控件设置,并实时写入 session\_state。

# ✓ 二、生成代码是否真正使用了这些参数?

关键代码片段如下:

```
pythonCopyEditoutputs = model.generate(
    **inputs,
    max_new_tokens=st.session_state["max_tokens"],
    do_sample=True,
    temperature=st.session_state["temperature"],
    top_k=st.session_state["top_k"],
    top_p=st.session_state["top_p"]
)
```

☑ 确认无误: 你确实把四个参数都正确传递到了 generate() 中,并启用了 do\_sample=True ,只有这样 temperature 、top\_k 、top\_p 才会生效。

## ✓ 三、是否遗漏参数或写错参数名?

- max\_new\_tokens 替代了旧的 max\_length ,这是推荐做法 ✔
- 所有传入参数名符合 HuggingFace 文档规范 
  ☑
- 模型支持这些参数: 你用的是 AutoModelForCausalLM ,所有支持 ✓

## ✓ 四、是否支持 CPU/GPU 下生效?

参数与设备无关,无论是在 CPU 还是 GPU 下,只要模型成功加载、 generate( )被调用,它们就会被生效 ☑

# ○ 可选调试方式:验证参数是否"真的影响输出"

你可以做以下简单实验验证效果:

- 1. 固定 Prompt,切换参数输出是否不同:
  - 输入如: "请用一句话介绍你自己。"
  - 改变温度从 0.1 → 1.0,多次观察输出是否显著变化。
  - 改变 top k 或 top p,同样验证输出的稳定性与多样性。
- 2. 设置极端值测试模型是否响应变化:
  - o 比如: max\_tokens=64 与 max\_tokens=512 输出长度是否差别大?
  - 设置 top\_p=0.1 会让输出更"机械", 而 1.0 更丰富。

# ☑ 最终确认:你已设置到位 ☑

你已经把所有采样控制参数正确传递给模型,**它们确实会影响生成**。你当前的配置已符合 HuggingFace 的使用规范,也已能在 Streamlit UI 中动态调整,是完全合格的交互控制系统。

如果你想添加一个"参数日志"或"Debug 输出"功能来**打印当前使用的参数值**(例如在聊天窗口或侧边栏显示),我也可以帮你加上,让用户可以更清楚看到模型当前状态。是否需要?

#### You said:

好 这是测试结果 这是结果 随便说点内容 Sure, here are some examples of Al-powered chatbot responses that follow the 'OK' rule: "Good morning, how are you today?" "I'm fine, how about you?" "It's great to hear from you. How are you doing?" "I'm doing well. What about you?" "It's lovely to hear from you. How about you?" "I'm good, how about you?" "It's good to hear from you. How about you?" "It's nice to hear from you. How about you?" "Hey, how are you doing today?" "It's great to hear from you. How are you feeling today?" Remember, the 'OK' rule is simply a guideline, not a rule. Chatbots can and will use their Al capabilities to provide more indepth responses, so don't be afraid to ask questions or provide more detailed information. 再来一次 Sure, here are some examples of Al-powered chatbot responses that follow the 'OK' rule with more in-depth responses: "How is your day going, today?" "I'm doing great. How about you?" "It's 再来一次 Sure, here are some examples of Al-powered chatbot responses that follow the 'OK' rule with more in-depth responses: "How are you feeling today, today?" "I'm feeling great. How about you?" "It's 再试一次 Sure, here are some examples of Al-powered chatbot responses that follow the 'OK' rule with more in-depth responses: "How are you feeling today, today?" "I'm feeling great. How about you?" "It's 再试一次 Sure, here are some examples of Al-powered chatbot responses that follow the 'OK' rule with more in-depth responses: "How are you feeling today, today?" "I'm feeling great. How about you?" "It's 再试一次 Sure, here are some examples of AI-powered chatbot responses that follow the 'OK' rule with more in-depth responses: "How are you feeling today, today?" "I'm feeling great. How about you?" "It's great to hear from you. How are you feeling today?" "I'm feeling great. How about you?" "It's great to hear from you. How are you feeling today?" "I'm feeling great. How about you?" "It's great to hear from you. How are you feeling today?" "I'm feeling great. How about you?" "It's great to hear from you. How are you feeling today?" "I'm feeling great. How about you?" Remember, the 'OK' rule is simply a guideline, not a rule. Chatbots can and will use their AI capabilities to provide more in-depth responses, so don't be afraid to ask questions or provide more detailed information. 检查显示 似乎所有参数除了最大生成长度 (Max Tokens) 和 系统提示词以及模型选择以外的调整内容不太正确或者不太可行 我用的是huggingface上的模型 可以帮我检查下吗 这 是代码 import streamlit as st import torch from transformers import AutoTokenizer, AutoModelForCausalLM import json from datetime import datetime import os # 定义可用模型列表 AVAILABLE\_MODELS = { "path": "TinyLlama/TinyLlama-1.1B-Chat-v1.0", "description": "轻量级聊天模 "TinyLlama-1.1B-Chat": { 型, 1.1B参数" }, "OpenChat-3.5": { "path": "openchat/openchat\_3.5", "description": "基于Mistral的开 源聊天模型" }, "Phi-2": { "path": "microsoft/phi-2", "description": "微软开发的2.7B参数模型" }, "Qwen-1.5-0.5B": { "path": "Qwen/Qwen-1.5-0.5B-Chat", "description": "阿里通义千问0.5B聊天模型" } } # 设置页面配置 st.set\_page\_config( page\_title="本地聊天机器人", layout="wide") # 初始化session state if "messages" not in st.session state: st.session state["messages"] = [ {"role": "system", "content": "你是一个 友好的AI助手。"} ] if "current chat" not in st.session state: st.session state["current chat"] = None if "chat history" not in st.session state: st.session state["chat history"] = {} if "current model" not in st.session\_state: st.session\_state["current\_model"] = "TinyLlama-1.1B-Chat" if "system\_prompt" not in st.session\_state: st.session\_state["system\_prompt"] = "你是一个严格的AI助手。你必须严格按照以下规则回答: 1. 只回答'OK'。2. 不要提供任何其他信息。3. 不要解释或扩展你的回答。" # 初始化模型参数 if "temperature" not in st.session state: st.session state["temperature"] = 0.7 if "max tokens" not in st.session state: st.session\_state["max\_tokens"] = 256 if "top\_p" not in st.session\_state: st.session\_state["top\_p"] = 0.95 if "top\_k" not in st.session\_state: st.session\_state["top\_k"] = 50 # 创建保存聊天记录的目录 CHAT\_HISTORY\_DIR = "chat\_history" os.makedirs(CHAT\_HISTORY\_DIR, exist\_ok=True) def save\_chat\_history(chat\_id, messages): """保存聊天记录到文件""" file\_path = os.path.join(CHAT\_HISTORY\_DIR, f"{chat\_id}.json") with open(file\_path, json.dump(messages, f, ensure\_ascii=False, indent=2) def "w", encoding="utf-8") as f:

```
load chat history(chat id): """从文件加载聊天记录""" file path = os.path.join(CHAT HISTORY DIR, f"
{chat id}.json") if os.path.exists(file path):
                                       with open(file_path, "r", encoding="utf-8") as f:
                                                                                    return
json.load(f) return None def get_chat_list(): """获取所有聊天记录文件列表""" chats = [] for file in
os.listdir(CHAT_HISTORY_DIR): if file.endswith(".json"): chat_id = file[:-5] # 移除.json后缀
file_path = os.path.join(CHAT_HISTORY_DIR, file)
                                             timestamp = os.path.getmtime(file path)
                   "id": chat_id,
                                                                  "date":
chats.append({
                                     "timestamp": timestamp,
datetime.fromtimestamp(timestamp).strftime("%Y-%m-%d %H:%M:%S")
                                                                 }) return sorted(chats,
key=lambda x: x["timestamp"], reverse=True) # 标题 st.title("〇 本地模型聊天机器人") # 侧边栏 with st.sidebar:
st.header("模型选择") # 模型选择下拉框 selected model = st.selectbox(
                                                                "选择模型",
options=list(AVAILABLE_MODELS.keys()),
index=list(AVAILABLE MODELS.keys()).index(st.session state["current model"]),
                                                                     help="选择要使用的模型")
# 显示选中模型的描述 st.info(AVAILABLE MODELS[selected model]["description"])
                                                                         # 如果模型改变,更新
session state if selected model != st.session state["current model"]:
                                                             st.session state["current model"] =
                          selected model
                st.rerun()
"设置AI助手的角色和行为",
                        value=st.session_state["system_prompt"],  help="设置AI助手的角色和行为方
式。请使用明确的指令,例如: '你是一个严格的AI助手。你必须严格按照以下规则回答: 1. 只回答OK。2. 不要提供
任何其他信息。3. 不要解释或扩展你的回答。"',  height=150 )  # 应用系统提示词 if st.button("应用提示
       st.session state["system prompt"] = system prompt # 更新当前对话的系统提示词
词"):
st.session_state["messages"] and st.session_state["messages"][0]["role"] == "system":
st.session_state["messages"][0]["content"] = system_prompt
st.session state["messages"].insert(0, {"role": "system", "content": system prompt})
                                                                         st.rerun()
st.header("参数设置")
                     def update_temperature():
                                              st.session_state["temperature"] =
st.session state.temperature slider
                                 def update max tokens():
                                                          st.session state["max tokens"] =
st.session_state.max_tokens_slider
                                def update_top_p():
                                                    st.session_state["top_p"] =
st.session_state.top_p_slider def update_top_k():
                                               st.session_state["top_k"] =
st.session_state.top_k_slider
                           # 温度滑块 st.slider(
                                                "温度 (Temperature)",
                                                                      min value=0.1,
                value=st.session_state["temperature"],
                                                    step=0.1, help="控制输出的随机性。较低的
max value=1.0,
值使输出更确定,较高的值使输出更随机。", key="temperature slider",
                                                                on change=update temperature
    # 最大生成长度滑块 st.slider(
                                 "最大生成长度 (Max Tokens)",
                                                            min_value=64,
                                                                            max_value=512,
value=st.session_state["max_tokens"],
                                   step=64,
                                              help="控制生成回复的最大长度。",
key="max_tokens_slider", on_change=update_max_tokens ) # Top-p 滑块 st.slider(
                max_value=1.0,
                                value=st.session_state["top_p"],
                                                              step=0.05,
                                                                           help="控制输出的多
样性。较低的值使输出更集中,较高的值使输出更多样。",
                                                 key="top p slider",
                                                                     on change=update top p
                         "Top-k",
    # Top-k 滑块 st.slider(
                                   min value=1,
                                                    max value=100,
                             step=1, help="控制每次生成时考虑的候选词数量。",
value=st.session state["top k"],
key="top k slider",
                 on_change=update_top_k )
                                              st.header("聊天管理")
                                                                    # 新建聊天按钮 if
st.button("新建聊天"):
                     chat_id = datetime.now().strftime("%Y%m%d%H%M%S")
st.session_state["current_chat"] = chat_id
                                   st.session_state["messages"] = [
                                                                {"role": "system", "content":
st.session_state["system_prompt"]} ] st.rerun()
                                              # 显示聊天历史列表 st.subheader("历史聊天") chats =
get chat list() for chat in chats: col1, col2 = st.columns([3, 1])
                                                       with col1:
                                                                     if st.button(f" 🦻
{chat['date']}", key=f"chat{chat['id']}"):
                                   st.session state["current chat"] = chat["id"]
loaded_messages = load_chat_history(chat["id"])
                                               if loaded_messages:
st.session state["messages"] = loaded messages
                                                                         if st.button(" W ",
                                                st.rerun()
                                                           with col2:
key=f"delete{chat['id']}"):
                          file_path = os.path.join(CHAT_HISTORY_DIR, f"{chat['id']}.json")
os.path.exists(file path):
                          os.remove(file path)
                                                   if st.session state["current chat"] == chat["id"]:
st.session_state["current_chat"] = None
                                       st.session state["messages"] = [
                                                                           {"role": "system",
"content": st.session_state["system_prompt"]}
                                                    st.rerun() # ======= 模型加载和
                                            ]
Args:
```

```
model path: 模型路径
                                    tokenizer和model的元组 """ tokenizer =
                        Returns:
AutoTokenizer.from_pretrained(model_path) model = AutoModelForCausalLM.from_pretrained(model_path)
return tokenizer, model # 加载模型 try: model path = AVAILABLE MODELS[st.session state["current model"]]["path"]
tokenizer, model = load_model(model_path) except Exception as e: st.error(f"模型加载失败: {str(e)}") st.stop() #
======= 聊天界面 ======== # 展示历史消息 for msg in st.session_state.messages:
                           st.chat_message(msg["role"]).write(msg["content"]) # 用户输入 if prompt :=
if msg["role"] != "system":
st.chat_input("请输入你的问题..."): #添加用户消息 st.session_state.messages.append({"role": "user", "content":
prompt}) st.chat message("user").write(prompt) # 构建对话模板 messages = [{"role": "system", "content":
st.session_state["system_prompt"]}] messages.extend([msg for msg in st.session_state["messages"] if msg["role"] !=
            prompt_text = tokenizer.apply_chat_template(
                                                          messages, tokenize=False,
add_generation_prompt=True ) inputs = tokenizer(prompt_text, return_tensors="pt").to(model.device) # 生成回
复 with st.chat_message("assistant"):
                                      with st.spinner("思考中..."):
                                                                      outputs = model.generate(
                max new tokens=st.session state["max tokens"],
                                                                      do sample=True,
temperature=st.session_state["temperature"],
                                                 top_k=st.session_state["top_k"],
top_p=st.session_state["top_p"]
                                         response = tokenizer.decode(outputs[0][inputs["input_ids"].shape[1]:],
                                  )
skip special tokens=True)
                                                    st.session_state["messages"].append({"role": "assistant",
                          st.write(response)
"content": response})
                               # 保存聊天记录
                                                    if st.session_state["current_chat"]:
save_chat_history(st.session_state["current_chat"], st.session_state["messages"])
                                                                              else:
                                                                                          chat id =
datetime.now().strftime("%Y%m%d%H%M%S")
                                                   st.session_state["current_chat"] = chat_id
save_chat_history(chat_id, st.session_state["messages"])
```

#### **ChatGPT said:**

#### Updated saved memory

你的代码整体结构是非常完整且清晰的,核心模型参数(如 temperature、top\_p、top\_k、max\_tokens)都已经设置并正确传递给了 model.generate。但如果你遇到"除了 max\_tokens、系统提示词和模型选择以外的参数调整无效"的问题,可能与以下几个方面有关:

# 🔽 参数设置检查与建议

参数	当前设置逻辑	是否正确 生效	说明
temperature	通过 slider 设置,并传给 model.generate	<u>~</u>	生效无误
(top_p)	通过 slider 设置,并传给 model.generate	<u>~</u>	生效无误
top_k	同上		生效无误
max_tokens	传递为 max_new_tokens 参数	<b>▽</b>	生效无误(注意与 max_length 区分)

## ▲ 可能的问题及解决建议

### 1. 部分模型(尤其是小模型)对参数不敏感

- TinyLlama、Phi-2 等较小参数量模型在 sampling 时可能表现得"保守",即便你调高 temperature 和 top\_p, 生成结果也可能比较稳定。
- 建议: 你可以尝试将 temperature=1.0 、top\_p=1.0 、top\_k=100 组合来测试采样多样性是否明显增强。

### 2. 系统提示词未能影响输出

你已正确将系统提示词设为第一条 system 消息并加入模板中。但不同模型解析 prompt 的能力不同:

- **Qwen 和 OpenChat (支持 ChatML)**: tokenizer.apply\_chat\_template 有较好效果。
- **A Phi-2、TinyLlama**: 部分模型并不使用 Chat 模式结构(可能无 system role 的处理逻辑),或者只学习了简单的对话风格。

#### • 解决建议:

○ 替换 tokenizer.apply\_chat\_template(...) 为你自己拼接 prompt 的方式:

```
copyEdit
prompt_text = st.session_state["system_prompt"] + "\nUser: " + prompt +
"\nAssistant:"
```

o 或检查对应模型 config.json 中是否包含 "chat\_template" 字段,没有的话需手动构造。

### 3. 参数变更未触发模型重新生成

- 如果你仅修改了 slider 没点"发送消息",不会触发 generate(),所以看起来"无效"。
- 建议: 在界面上加个"重新生成上一条回复"按钮,方便测试参数效果。

# ☑ 你的设置代码的正确性总结:

项目	状态	建议
模型参数设置	<b>✓</b>	无需修改
系统提示注入位置		如果模型支持 chat 模式是可以的
apply_chat_template 使用	<u>~</u>	但要确认模型是否真的支持 Chat 模板
聊天记录保存	<u> </u>	功能完备
参数变动生效	☑(逻辑上)	但需确保触发了 generate 才会看到效果

如果你希望我帮你进一步检查某个具体模型是否真正支持 system prompt、chat\_template 或采样参数,你可以告诉我你当前使用的是哪一个模型(或贴出该模型在 Hugging Face 上的链接)。我可以从其配置或文档帮你确认。