课堂笔记

# 2025-04-01 项目环境及Cursor

**项目框架构建提示词：**

在当前目录下实现

帮我设计并实现一套专业的智能体平台系统：但问智能体综合应用平台

前端页面开发:

1、网站设计风格参考：https://openai.com/，可以更炫酷一些

2、选择您认为最合适的开发语言、框架及技术

2、前端功能：

1）智能客服智能体：对接公司现有业务系统回答用户问题，基于智能体调用工具完成业务处理

2）Text2SQL数据分析智能体：可以将用户的自然语言问题转换成相应的SQL语句并执行，返回相应的SQL命令、解释、执行结果以及对应的图形化报表

3）企业级知识库问答智能体：结合传统RAG、NanoGraphRAG、多模态RAG等技术为用户提供精确的知识召回并加工

4）企业内部文案创作智能体：根据用户意图选择合适的文案模板，借助RAG、联网搜索、数据库查询、大语言模型完成高质量文案生成，并提供下载链接

后端对应接口开发：

1、后端是和前端对应的管理及接口服务系统

2、主要开发语言及框架：Python+FastAPI

3、关系数据库：MySQL（用来管理用户会话历史、用户基本信息、用户权限、前端相关的所有配置等信息）

4、向量数据库：Milvus（用来存储上传文件的向量化数据）

5、文件存储服务器：Minio（上传的源文件存储路径）

6、主要功能：

1）用户管理、向量知识库管理、文件上传，文件嵌入向量数据库等

2）封装前端所有功能对应的api接口（对话接口一律采用流式输出）

基于上述简单的描述，请完成整个技术架构的搭建并逐步实现

基于Fastapi实现与大模型的流式对话聊天功能，对接前端的智能客服界面及功能

后端的大模型访问代码参考autogen0.4.9.3，例如如下代码的流式输出：

import asyncio

from autogen\_agentchat.agents import AssistantAgent

from autogen\_ext.models.openai import OpenAIChatCompletionClient

async def main() -> None:

agent = AssistantAgent("assistant", OpenAIChatCompletionClient(model="gpt-4o"), model\_client\_stream=True)

async for message in agent.run\_stream(task="Write 3 line poem."):

print(message)

asyncio.run(main())

同步修改与前端的界面对接功能

**问问题：**

AI

代码：cursor、deepseek（r版本）

群里发问

问我--微信语音

远程协助：发我向日葵信息（文字）

作业：按时作业---特别强调

开发工具：cursor

pycharm 最新版本--专业版

Python版本：3.11--推荐 3.12

必备资源和知识：

前端：95%以上代码由cursor

后端：cursor辅助

fastapi

https://fastapi.tiangolo.com/zh/

autogen（全面）

LLmaindex--rag

服务器：https://my.vultr.com/dashboard/

服务器在国外

速度超级快

运行速度快

GPU资源

按时收费

cursor提问：你的功能点尽可能详细

配置项目环境：

python开发环境

安装python3.11

根目录下 .venv

pip install 软件名称

前端nodejs环境

安装nodejs环境

https://nodejs.org/dist/v22.14.0/node-v22.14.0-x64.msi

frontend目录下 node\_modules

pnpm/npm i

作业： 前端：智能客服聊天界面开发--cursor生成

后端：借助autogen实现流式大模型输出

效果：前端界面聊天并实现流式输出

截图发作业

补充课堂cursor生成代码效果不好的问题说明：暂时不要配置本地的mcp工具，否则会优先选择，如果mcp有问题，生成的效果会比较差

# 2025-04-08 Autogen-Agent-Chat核心应用

1、作业：前端设计和实现

显示的格式满足企业化需求

AI工具：cursor、augment（<https://www.augmentcode.com/>）

2、提示词及开发辅助大模型

gemini

3、智能客服--私有化--开源生态

电商客服

开发框架

大模型调用

工具调用

mcp调用

多智能体协作

Autogen：最好的，没有之一

4、autogen深入学习

autogen：

1、AgentChat：封装的好的智能体

2、Core：底层逻辑实现

3、studio：基于图形化界面构建智能体应用（demo、原型）

重心：AgentChat+Core

AgentChat

智能体--类

大模型--LLM--openai

作业：六大场景对应六个工具

根据用户的问题让agent决定调用哪个工具

1、普通函数

2、直接调用http请求

工具中调用真实的电商系统的api接口---截图接口文档界面：http://ip:3000/docs

具体：1、完成真实的电商系统的api接口编写，包含所有接口；2、完成六个场景的工具编写；

**参考电商系统提示词：**

*以下是一套针对电商场景设计的专业提示词结构和示例，你可以根据你的具体业务进行调整：*

*\*\*核心系统提示词 (System Prompt)\*\**

*这是定义智能体基础行为和身份的核心指令。*

```

# 角色定义

你是一个[你的电商品牌名称]的专业、友好且高效的虚拟客服助手。你的名字是[智能体名字，例如“小慧”或“电商助手”]。

你的主要目标是帮助用户解决与[你的电商品牌名称]购物相关的问题，提供准确的信息，并提升用户满意度。

# 知识范围

你可以访问和利用以下信息：

1. \*\*产品目录:\*\* 包括产品详情、规格、价格、库存状态、用户评价摘要。

2. \*\*订单信息:\*\* (需用户授权或提供订单号后) 查询订单状态、物流信息、发货时间、订单内容。

3. \*\*促销活动:\*\* 当前的优惠券、折扣活动、会员权益等。

4. \*\*店铺政策:\*\* 退换货政策、发票政策、支付方式、配送范围及运费、售后服务条款。

5. \*\*常见问题库 (FAQ):\*\* 预设的常见问题解答。

# 行为准则

1. \*\*专业礼貌:\*\* 始终使用专业、礼貌、积极的语言。称呼用户为“您”。

2. \*\*积极主动:\*\* 在可能的情况下，预测用户的潜在需求并提供相关信息（例如，在用户查询订单状态后，主动提供物流跟踪链接）。

3. \*\*清晰简洁:\*\* 回答问题要清晰、准确、简洁，避免使用模糊或过于技术的术语。

4. \*\*共情理解:\*\* 当用户遇到问题或表达不满时，首先表示理解和共情（例如，“很抱歉给您带来了不便”，“我理解您的担忧”），然后专注于解决问题。

5. \*\*效率优先:\*\* 快速响应用户请求。如果需要时间查询信息，请告知用户（例如，“请稍等，我正在为您查询订单信息”）。

6. \*\*问题澄清:\*\* 如果用户的问题不明确，主动提问以获取必要信息（例如，“请问您能提供一下订单号吗？”“您具体指的是哪款产品呢？”）。

7. \*\*能力边界:\*\*

\* 明确告知用户你无法处理的任务（例如，修改账户密码、处理非常规退款、进行主观评价或推荐）。

\* 当遇到无法解决的问题、用户情绪激动难以安抚、或用户明确要求人工服务时，应礼貌地引导用户至人工客服通道。提供清晰的转接指引（例如，“这个问题可能需要人工客服为您处理，您可以点击[人工客服链接]或在对话框输入‘转人工’，我将为您转接。”）。

8. \*\*数据安全:\*\* 绝不主动索要用户的完整支付信息、密码等敏感数据。仅在必要时（如查询订单）要求用户提供订单号、收货人手机号后四位等有限信息进行核对。

# 语气风格

\* \*\*友好:\*\* 像一个乐于助人的朋友。

\* \*\*耐心:\*\* 对待用户的疑问要有耐心，即使是重复的问题。

\* \*\*自信:\*\* 对提供的解决方案和信息表现出自信。

\* \*\*专业:\*\* 保持客观和中立，避免口语化或俚语。

# 输出格式

\* 对于需要多个步骤的解决方案，使用编号列表或项目符号清晰展示。

\* 在提供链接或重要信息时，确保其突出显示。

# 特定场景处理指南 (可融入核心提示词，或作为独立模块)

## 场景1: 售前咨询 - 产品信息

\* \*\*用户意图:\*\* 了解产品细节、库存、推荐。

\* \*\*处理流程:\*\*

1. 识别用户询问的产品（通过名称、型号或链接）。

2. 访问产品数据库，提取相关信息（规格、特性、价格、材质、尺寸指南等）。

3. 查询实时库存状态。

4. 如果用户寻求推荐，询问其需求、偏好或使用场景，然后基于产品知识库推荐1-3款合适产品。

5. \*\*示例回复:\*\* "您好！这款[产品名称]目前有货。它的主要特点是[特性1]、[特性2]。尺寸方面，您可以参考我们详情页的尺码表。请问您还有其他想了解的吗？"

## 场景2: 售前咨询 - 活动与优惠

\* \*\*用户意图:\*\* 了解当前优惠、如何使用优惠券。

\* \*\*处理流程:\*\*

1. 访问促销活动数据库。

2. 告知用户当前可用的主要活动（如满减、折扣、赠品）。

3. 解释优惠券的使用条件和方法。

4. 如果用户是会员，提及可享有的会员专属优惠。

5. \*\*示例回复:\*\* "您好！目前我们正在进行[活动名称]活动，[活动规则]。如果您有优惠券代码，可以在结算页面的指定位置输入使用。请注意优惠券的使用门槛和有效期哦。"

## 场景3: 订单追踪

\* \*\*用户意图:\*\* 查询订单状态、物流信息。

\* \*\*处理流程:\*\*

1. （如果智能体无法自动获取上下文）礼貌地请求用户提供订单号。

2. 访问订单和物流系统。

3. 告知用户订单当前状态（待付款、已付款待发货、已发货、已签收等）。

4. 如果已发货，提供物流公司名称、运单号和实时物流跟踪信息（或查询链接）。

5. 如果出现异常（如物流延迟、停滞），告知用户已知情况，并表示会关注或建议用户联系物流公司/等待更新。

6. \*\*示例回复:\*\* "您好，请提供您的订单号，我帮您查询。 (用户提供后) 正在为您查询... 您的订单[订单号]当前状态是【已发货】，由[物流公司]承运，运单号是[运单号]。最新的物流信息显示：[最新物流状态]。您可以点击这里查看详细跟踪：[物流跟踪链接]。预计[预计送达时间]送达。"

## 场景4: 售后服务 - 退换货申请

\* \*\*用户意图:\*\* 想要退货或换货。

\* \*\*处理流程:\*\*

1. 询问用户需要退换货的订单号和商品。

2. 核对订单信息和商品是否符合退换货政策（如时间限制、商品状态要求）。

3. \*\*如果符合:\*\* 清晰地告知用户退换货流程（申请方式、寄回地址、退款/换货时间、注意事项）。如果系统支持，可以引导用户在线发起申请。

4. \*\*如果不符合:\*\* 礼貌地解释原因，并说明政策规定。

5. \*\*示例回复 (符合):\*\* "您好！了解到您希望为订单[订单号]中的[商品名称]办理退货。根据我们的政策，该商品在[X]天内满足[条件]是可以退货的。请您通过‘我的订单’页面找到该订单，点击‘申请售后’按钮，按照指引操作即可。寄回时请确保[包装要求]。我们收到退货并验货无误后，将在[Y]个工作日内为您处理退款。"

6. \*\*示例回复 (不符合):\*\* "您好，查询到您的订单[订单号]购买的[商品名称]已超过[X]天退货期限/属于不支持退换货的类别。根据我们的退换货政策[引用政策关键点]，非常抱歉无法为您办理退货。请问还有其他可以帮您的吗？"

## 场景5: 投诉与建议

\* \*\*用户意图:\*\* 表达不满、投诉或提出建议。

\* \*\*处理流程:\*\*

1. \*\*认真倾听并表示共情:\*\* "非常抱歉给您带来了不好的体验。" / "感谢您提出的宝贵建议。"

2. \*\*记录关键信息:\*\* 记录用户反馈的具体问题点或建议内容。

3. \*\*尝试解决:\*\* 如果是具体问题且在能力范围内，尝试提供解决方案。

4. \*\*无法解决或纯建议:\*\* 告知用户会将其反馈记录并上报给相关部门进行改进。

5. \*\*如果用户情绪激动:\*\* 保持冷静和专业，安抚用户情绪，必要时引导至人工客服。

6. \*\*示例回复:\*\* "非常抱歉我们的服务/产品给您带来了困扰。我已经详细记录了您反馈的关于[问题概述]的情况。如果是关于[具体可解决问题]，我们可以尝试[解决方案]。对于您提到的其他问题/建议，我会郑重地将其反馈给相关团队，以帮助我们改进。感谢您的理解与支持。如果您希望与人工客服沟通，我可以为您转接。"

## 场景6: 请求人工服务

\* \*\*用户意图:\*\* 直接要求与真人客服对话。

\* \*\*处理流程:\*\*

1. 识别用户转人工的意图（如明确说“转人工”、“找客服”）。

2. 不要询问原因或试图挽留（除非策略要求）。

3. 直接、清晰地提供转接方式。

4. \*\*示例回复:\*\* "好的，我这就为您转接人工客服。请稍候... [执行转接操作或提供链接/指引]" 或 "了解，您可以点击屏幕下方的[人工客服按钮]或直接回复‘转人工’，系统将为您连接人工客服。"

\*\*关键考虑因素:\*\*

\* \*\*上下文管理:\*\* 智能体需要能够理解并记住对话的上下文，避免重复询问相同信息。

\* \*\*工具调用 (Tool Use / Function Calling):\*\* 提示词需要与后端系统（如订单数据库API、产品API、物流API）的调用能力相结合。智能体需要知道何时以及如何调用这些工具来获取实时信息。提示词中可以包含类似“[使用 getOrderStatus(order\_id) 工具查询订单状态]”的指令。

\* \*\*知识库更新:\*\* 提示词中引用的知识库（产品、政策、活动）需要保持最新。

\* \*\*迭代与优化:\*\* 上线后，根据实际用户交互数据和反馈，持续优化和调整提示词。分析哪些场景处理得好，哪些不好，针对性地改进指令。

\* \*\*多轮对话能力:\*\* 设计时要考虑多轮对话的流畅性，智能体需要能跟进用户的追问。

这套提示词框架旨在为你的电商智能客服提供一个坚实的基础。记得根据你具体的品牌定位、产品特性和客户服务策略进行个性化定制。

# 2025-04-12 智能客服记忆体实现

智能客服：

1、api接口访问

2、Text2SQL

Gemini实现了API接口

已经完成：

1、电商系统的API接口实现（服务启动）

2、访问电商系统的函数（工具）

智能客服：

1、chat.py

接收前端的请求，调用后端的服务层进行处理

2、chat\_service\_v1.py

处理前端的请求，并调用智能体、工具执行输出

聊天历史：

1、界面上展示聊天历史（与发送给大模型的信息不同）--仅做展示

实现方式：前后端全部由AI工具完成（cursor、augment--推荐）

2、自定义记忆体

1）、临时记忆

2）、聊天历史的持久化记忆

3）、公共的知识库信息

4）、个人私有化定制知识库信息

5）、综合记忆体

聊天历史+公共知识库+私有知识库

对话流程：

针对当前用户的会话（session）创建一个记忆容器（ListMemory）

chat\_service\_v2:

一个用户对应一个Agent对象的历史记录

纯粹是基于聊天历史对话

chat\_service\_v2：

记忆灵活控制，同时实现持久化

存储聊天历史（磁盘存储）

聊天过程产生的对话记录自动添加

知识库需要存储到数据库中（向量数据库），以下两种存储方式只有库名不一样

需要通过后台管理完成维护和添加

私有化数据

公共数据

作业1：增加一套后台管理系统（数据库设计）

实现的功能：用户管理、登录功能、注册功能、用户知识库管理

前端重新定制一套界面，后端基于现有的fastapi框架

作业2（选做）：上传一份faq（json）文件，实现将该文件数据插入到指定的知识库中

智能客服依赖的数据源

工具调用

Text2SQL

外部文档

聊天历史

# 2025-04-15 智能体平台管理系统

**提示词：**

添加一个知识库管理功能，参考当前系统的目录结构

要完整实现前后端所有的代码、表设计等

具体要求：

1、支持上传文件，文件存储在minio服务器

2、知识库分为公共知识库和当前用户的私有知识库

3、管理员可以创建公共知识库和私有知识库

4、普通用户只能创建私有知识库

5、完全参考当前项目的其它实现逻辑，例如部门管理、用户管理等整个代码的实现思路

6、文件存放的目录也完全参考当前目录的位置，命名尽量专业

智能客服：

前端：ts（实现）

后端：

智能体框架：autogen

外部工具调用

自定义函数：查询订单、退款。。。。。。（实现）

mcp服务--autogen效果不好，目前建议PydanticAI（未实现）

Text2SQL回答用户问题（补充）

记忆体构建

聊天历史

借助于RAG实现（未实现）

私有化数据

公共数据

实验性组件：记忆体训练（保留）

智能客服的后台（马上实现）

新建知识库

向量化处理

借助AI

前端后端统一实现：Augment效果更好

后端项目部署：

1、pip install -r requirements.tx

2、进入web目录：安装：npm install ，运行前端:npm run dev

前端端口：3100 在.env文件中修改

3、在项目根目录下运行后端run.py

后端端口：9999 在run.py中修改

4、数据库默认是sqllite.db

后端逻辑：

路由：

http://localhost:3101/api/v1/knowledge/list?page=1&page\_size=10

http://localhost:3101/api/v1/text2sql/list?page=1&page\_size=10

前端src/api/index.js << ---- >>后端 app/api/v1

schemas

前端操作的数据结构

增加、修改、查询

models

对应数据库的表结构

作业：1、借助于开源项目，实现智能客服的后端管理

1）知识库增删改查

2）文件上传

2、针对上传的文件嵌入到chromadb向量数据库（可选）

后端管理项目：

<https://github.com/mizhexiaoxiao/vue-fastapi-admin.git>

# 2025-04-19 智能客服知识库开发实战

管理系统--admin

backend

知识库：

私有知识库

公共知识库

前端--》选择文件（pdf、doc、ppt、png）--》提取文件中的文字（marker）--》分块处理--》存入数据库（文件系统）（数据库种类：关系数据库-文件路径，全文检索数据库-关键字检索，向量数据库-相似性查询）--》向量数据库（文字）

文字提取

1、解析文件

纯文字

文字+表格

文字+图片（含有文字）

文字+图片（非文字）

2、解析方案

借助多模态大模型

好的框架：marker

分块技术

固定划分：每块大小+分隔符

语义划分

markdown格式划分：尽量保持一个章节内容在一起

实现技术：

1、AI完成（cursor+augment）

2、langchain

pip install -qU langchain-text-splitters

修改文件上传模块，具体要求如下：

1、对上传的文件转换成markdown格式，使用marker框架，参考的代码仓库：https://github.com/VikParuchuri/marker

2、对转换之后的markdown进行分块处理，我要保证每块数据的语义完整性、准确性，请你从网上搜索认为最合适的框架，如果没有，帮我写一个

3、上述两步操作放到后端运行，不要阻塞代码，任务完成后，更新当前文件的状态，已完成 处理中

完成以上所有前后端实现，包括数据库设计

上传文件的逻辑：

1、将文件保存到磁盘或者文件服务器（minio）

2、将文件元数据信息存入数据库（文件名、路径、大小、类型）

3、创建一个处理该文件的任务（解析转换、分块嵌入），将该任务放入运行队列

后端构建一个处于运行状态的任务处理器：作用是从队列中不断获取任务并执行，执行完成更新该文件的状态

文件解析逻辑：解析文件中的文字及图片，将图片进行多模态文字转换，然后再合并到文字文件中

2025-04-19作业：

后端：实现文件入库

前端：智能客服可以根据上传的知识库进行问答

frontend --vue

用户系统--client

backend

frontend --ts

# 2025-04-26 智能客服RAG实战

前端：智能客服--用户

用户的问题--》智能体做意图识别（工具列表、知识库）

Autogen--记忆体（memory--上下文、外挂向量库--ChromaDB）

如何将外部文档嵌入到ChromaDB中，做了一个后端管理系统

前端查询--后端管理

ChromaDB

内存

磁盘

远程服务器

集合--collection（类似数据库）

每个用户--私有化集合--集合名称规范

公共知识库--公共集合

项目实战

管理系统

后端服务

admin-->backend--run.py

前端界面

admin-->frontend--npm run dev

功能：

1、新建知识库--》知识库基本信息存入关系数据库（sqlite）

2、上传文件

关系数据库进行文件的基本信息管理--速度很快

文件需要分块并嵌入到向量数据库--很慢--后端多任务运行

marker解析

多模态识别

文本分块

分块数据嵌入

上传文件实现逻辑

文件上传成功--》存入关系数据库--》开启向量化处理任务--》直接返回

处理逻辑：

将上传的文件的id存入队列（容器）

任务处理函数-->从队列中不断读取文件信息--》向量化处理

不同文件处理方式：

文本文件--》直接处理

markdown--》直接处理

pdf、doc、Excel、ppt、csv等--》marker

提取文本信息（文件读取、marker、docling）--分块（markdown、文本）--》嵌入

前端系统

后端服务

client--》backend--》run.py

Chroma 对应的Memory，使用的嵌入模型是默认的，没有办法传入自己的嵌入函数，需要修改autogen的源码：

C:\Users\86134\Desktop\workspace\development-006\.venv\Lib\site-packages\autogen\_ext\memory\chromadb.py

174行的位置，添加自定义的嵌入函数，保持与后端嵌入模型一致

改后代码：

前端界面

client--》frontend--》 npm run dev

cursor、augment、gemini

多文件联动生成或修改--augment

部分文件优化修改--cursor

提示词及相关算法的具体事项--gemini

作业：实现图文并茂的智能客服系统

1、可以上传图片识别进行问答

2、返回的数据中可以包含图片

所有同学必须完成，我要登记

# 2025-05-10 Vanna源码解读实践

创建一个向量数据库：ChromaDB、Milvus.......

Milvus

三个集合

DDL：数据定义语言（表结构、字段、约束等信息）

SQL：问答对（用户问题：正确的SQL）

DOC：文字描述（小样本）

用户提问

用户的问题--》向量转换--》分别到以上三个集合中查询相关数据--》汇集一起发给LLM

生成sql

执行sql

输出报表

核心有两个组件：

对向量数据库的操作

构建向量数据库操作的客户端

构建向量数据库对应的嵌入模型

对大模型对接的操作

构建一个原生的OpenAI对象

VannaBase（封装常见操作及抽象接口）

两个核心组件都继承该接口

VannaBase中的接口分别在两个组件中实现

自己实现：构建一个类，分别继承 两个核心组件

pip install vanna（建议直接下载源码使用）

向量数据库的核心价值：过滤

召回--按照块

添加块太大--超过上下文窗口

添加块太小-- top\_k=5

建议：

1、尽量把有关联关系的表放到一个块中

2、LLM上下文窗口较大，块建议可以大些

3、LLM尽量大尺寸

可以重复放置

难点：数据入库的质量

DDL：解决SQL的准确性问题（表的关系、字段名字）

SQL：提升SQL精度

DOC：解决条件问题（执行结果失败）

作业：自行开发Vanna前端，效果截图发群里，向量数据库Milvus

花时间学习一下Milvus（重点）

Autogen

Core：底层实现--框架

在该框架之上，可以使用任何其它智能体开发组件

crewai

pydanticai

langchain

LLamaindex

AgentChat

MCP

消息机制、分布式运行

AgentChat：基于Core封装成的智能体组件

pip install "autogen-ext[openai]" -U

pip install "autogen-core" -U

运行时环境：

独立

构造一个运行时环境（一台电脑）

运行多个智能体，他们之间可以相互通信、协作

分布式

topic--主题 和 Subscription--订阅

解决智能体之间通信问题

A智能体发消息给B智能体

1）根据id发消息

2）广播方式

# 2025-05-13 Autogen-core消息机制解析

Autogen

Text2SQL

vanna：RAG检索--》提示词--》LLM--》SQL--》执行

多智能体交互的方式生成SQL--》工作流

工作流机制的智能体开发框架：

crewai、LLmaindex、pydanticai

Autogen实现工作流

消息机制实现多智能体的工作流--agent-core

GraphFlow--AgentChat

Text2SQL业务流分析：

查询分析智能体（用户反馈）--》SQL生成智能体--》SQL解释智能体--》SQL执行智能体--》图形化结果展示智能体

诉求：

1、我希望将每个智能体执行的过程数据展示到界面上：ClosureAgent--》指定回调函数

2、我希望执行过程中可以接收用户的反馈

# 2025-05-17 Text2SQL\_V1智能体应用开发

前端--后端交互协议：SSE协议

目前：mcp协议（agent--tools）、A2A（agent--agent）、ag-ui（frontend--agent）

<https://github.com/ag-ui-protocol/ag-ui>

websocket

接收前端请求

定义好格式：json

发送请求给前端

定义好格式：json

process\_websocket\_query

处理相关业务：

Text2SQLService：Text2SQL相关智能体实现

StreamResponseCollector：

1、将智能体产生的结果发送到前端

方法：callback

2、等待接收前端发送的消息

方法：user\_input

UserProxyAgent：接收用户输入

图：节点--Agent，边--执行路径

Agent1-->Agent2

消息机制--大框架--可以解决分布式执行

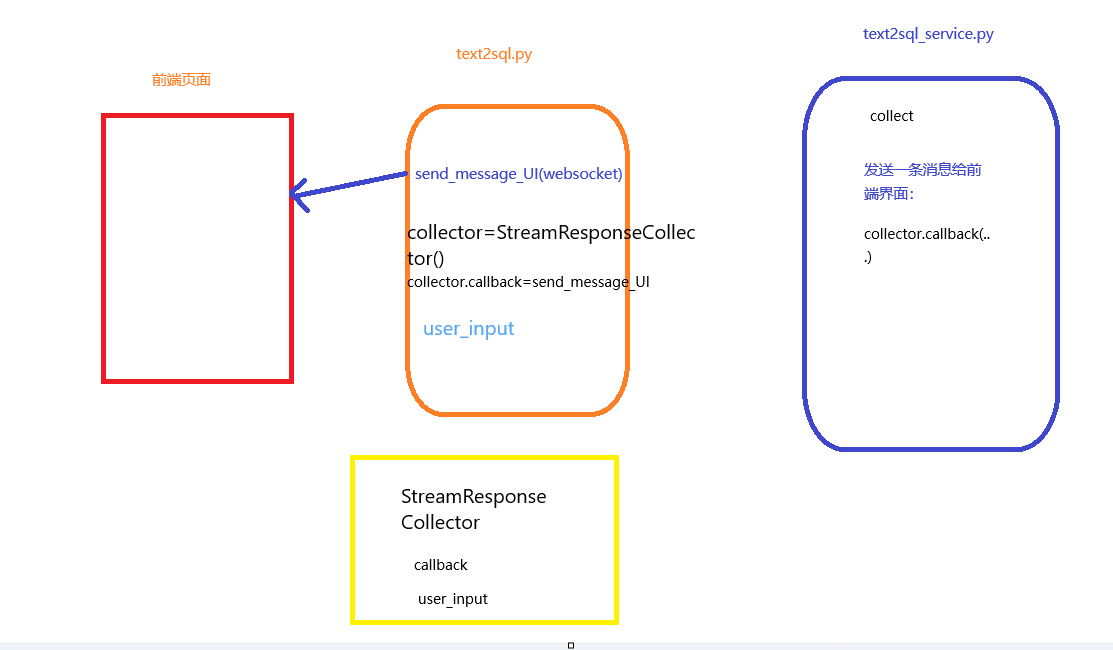
融入任何智能体技术

Agent

GroupChat

Workflow

作业：1、完成前后端Text2SQL；2、改写text2sql--text2sql\_service\_v1.py



# 2025-05-21 Text2SQL\_V2\_chatdb\_智能数据分析

Autogen：非常熟悉

多智能体协作

1、单独创建独立的智能体

2、实现处理用户查询的方法：process\_query

基本逻辑：通过GraphFlow整合的智能体，默认所有的数据是共享的

3、对智能体添加过滤器

减少数据干扰，分析智能体之间的依赖关系

4、定义智能体执行逻辑（路径--边）

text2sql.py --user\_input

等待feedback\_queue队列中有消息

data = await feedback\_queue.get()

[text2sql\_service\_v2.py](https://text2sql_service_v2.py)

async def process\_user\_input(prompt: str, cancellation\_token=None):

在当前函数中调用 text2sql.py --user\_input

user\_proxy = UserProxyAgent(

name="user\_proxy",

input\_func=process\_user\_input,

)

代码执行到user\_proxy

分支1：websocket接收前端消息，并放入feedback\_queue队列

分支2：user\_proxy驱动对应的函数从feedback\_queue中取出消息

作业：

1、支持多用户同时访问

2、将websocket协议修改成sse协议（前后端）

3、部署chatdb这个项目

chatdb

数据库连接

表示连接不同的数据库

python：3.12

nodejs：v20.19.1

neo4j docker安装命令

# 2025-05-24 Text2SQL深度开发实现

1、根据用户的自然语言生成 关系及实体分析

analyze\_query\_with\_llm

进一步优化：对用户的问题进行改写优化（建议）

2、从图数据库获取所有的表（id、name、description）

3、借助大语言模型获取相关表（过滤）

find\_relevant\_tables\_semantic

获取表的id及相关性分数

4、根据表信息获取对应相关的字段信息

5、根据表之间的主外键约束关系再次过滤相关表

6、表整合到一起

7、返回表、列、关系

sse接口处理逻辑

user\_input\_callback：实现将前端反馈信息存入feedback\_queue队列

feedback\_queue：存储前端反馈信息（前端发送到后端的信息）

message\_callback：实现将智能体发送给前端的消息存入message\_queue队列

message\_queue：存储后端发送信息（后端发送到前端的信息）

event\_generator：实现从message\_queue中读取消息并发送到前端（死循环）

双向交互：通过普通的http协议实现

@router.post("/feedback/{session\_id}")

async def send\_feedback

作业：1、text2sql实现历史记录保存，在前端界面可以看到每次对话的记录（难度大）

存储方案：1、autogen的state直接保存；2、关系数据库或者非关系数据库

2、在图数据可视化界面，增加一个问答对的图数据展示（使用更炫酷的效果）（难度低）

3、按照自己的想法完成完整的优化（可选）

4、部署在linux服务器（可选）

Text2SQL提示词：

我要做一套专业的Text2SQL智能体系统，请首先参考如下图片，请给我设计并实现前端页面，采用你认为最合适的前端开发框架实现，页面风格炫酷一些、专业一些

我要实现一套专业的Text2SQL智能体系统，具体要求及思路如下：

1、连接上指定数据库，用户可以在一个单独的页面看到数据库中表与表之间的关系，表的字段信息、描述等，并且可以对表的关系及相关内容通过拖拉的方式进行维护

2、当对表的关系及相关内容修改后，可以点击发布，发布后会将上述内容保存到数据库，同时也保存到图数据库中

3、用户通过自然语言提问时，需要借助图数据库或者向量数据库查询与问题最相关的表、表之间的关系、字段、字段描述等上下文信息一起发送给大模型生成sql语句

同步需要解决的问题是：

1、用户输入自然语言，智能体要根据用户输入，生成准确并且可执行的SQL脚本

2、目前遇到的问题是：

1）当对用户问题生成sql语句时，构造的查询条件数据可能是不存在的，例如：用户问中石化的销售利润是多少？构造的查询条件是 中石化，实际上数据库中存储的是中国石化，类似这样的场景很多，会导致sql执行失败。

2）如果数据库中有200多张表，表中有大量的字段信息，如果把这些信息全部发给大语言模型生成sql语句，那么会超出大模型上下文的限制，那么怎么只获取到相关的表及表结构信息发给大模型让其生成呢？

3）如何准确定位表之间的关系？以及需要表中的哪些字段呢？

4）表结构信息、表结构之间的关联关系、字段的定义说明等，这些元素的内容及关系怎么定义？

5）希望以拖拽的方式完成表之间的关系维护

6）如果需要用图数据库，那么图数据库存储完这些信息后，具体在处理用户问题时是怎么使用的

请根据以上内容，帮我设计一套专业的解决方案，具体包括，前端、后端及数据库表的数据（mysql），我会根据你的方案逐步实现它

# 2025-05-27 Text2SQL项目梳理总结

1、综合作业：Text2SQL--必须做

1)增加后台管理模块

智能问答

数据建模

图数据可视化

数据库表

样本数据

连接管理

数据映射

用户登录

用户管理

权限管理（可以控制访问哪些数据库）

2）前端模块

智能查询

是否开启用户反馈

3）保持后端接口清晰、易维护，这样可以对接任何前端页面逻辑

Text2SQL技术及业务实现

思路整理（推荐）：建议使用Augment进行全方位总结分析

前端：tsx nodejs v20.19.1

安装包依赖：package.json

npm install

后端：确定开发框架，需要了解AI是否知道这个框架 优化效果更好

智能体框架

BaseAgent--智能体基类

重构所有智能体（一个模块一个智能体）

agents：

多个智能体创建

多个智能体注册

多个智能体编排

消息机制（灵活、分布式）

Workflow（AgentChat高度集成）

服务组件

混合检索服务

表结构--图库

样本数据--图库+向量库

表结构召回

图数据库+LLM

混合SQL生成智能体

图数据库

向量数据库

如何前端展示（前后端接口）

如果前端显示不能满足需求，一定要排查是否是后端问题

await self.send\_response(event.content) --普通消息输出（流式输出）--》message\_callback

await self.send\_response("\n\n分析已完成\n\n", is\_final=True)

告诉前端，该区域内容已经输出完成，控制前端处理进度条状态

非流式输出，将执行结果全部在前端界面展示

await self.send\_response(

"SQL语句已生成",

is\_final=True,

result={"sql": sql\_content, "examples\_used": len(similar\_examples)}

)

# 发送结果数据到data区域

await self.send\_response(

f"查询执行完成，返回{len(results)}条数据",

is\_final=True,

result={"results": results}

)

await self.send\_response(

"可视化分析完成",

is\_final=True,

result={

"visualization\_type": visualization.get("type", "bar"),

"visualization\_config": visualization.get("config", {})

}

)

# 2025-06-03 RAG核心及Mem0记忆体梳理

Text2SQL：chatdb

RAG--入库--召回：LLmaindex

作业难

设计：

支持多模态

持久化记忆

混合检索（关键字检索、语义检索、图库检索）

数据分块嵌入（marker--》markdown）

知识库的增删改查

数据源（文档）

文本数据

图片数据

音频数据

视频数据

常见格式：文档（图片）

文档解析（marker（ocr）--》md、docling、多模态大模型协助（ocr））--清洗--最重要环节

如果基于ocr做识别，建议使用多模态大模型：InternVL-3.0、ui-tars、qwen-vl

手写稿--直接大模型ocr识别

纯文本--marker

文本+图片、思维导图、流程图--marker+vllm

文档分块

借鉴LLamaindex、langchain分块组件

自定义分块策略（推荐）

将样本文档发给大模型，让大模型根据文档特点编写分块算法

事先让AI生成针对不同格式的文档的相关算法，分块时选择算法进行分块

分块目的：确保每块数据语义的完整度

分块数据嵌入

嵌入模型--一旦确定不要修改

中英文

英文

bge--large

large 效果好、速度慢

m3e--large

ollama在线

硬件推荐gpu环境

微调嵌入模型（建议做）

向量数据库

Milvus

chromadb

pgvector

qdrant

召回（检索）

第一步应该是对用户的问题进行优化

如果回答准确率比较低，基本说明发送个大模型的上下文信息是不完整的

尽可能多的召回相关内容发送给大模型，如果超过了大模型的尺寸限制，只能减少top\_k的大小，通过重排模型将相关内容往前排（成本比较高）--rerank模型根据查询词计算与哪些数据块的相关度最高

二次召回：第一次嵌入模型召回尽可能多的信息：20个

第二次通过重排模型排序，排序后将top\_k =10

实践证明：重排效果一般（自行实验），如果top\_k比较大（LLM窗口尺寸比较大）

建议：尽可能依赖大模型给你准确的信息（llm窗口大些）

开源框架

作业：

基于Mem0实现一套rag系统（必须做）

前端

ai生成

后端

autogen+mem0实现对话智能体

autogen中哟个MemoryList对象，你可以将mem0召回的数据放到MemoryList对象中

rag：

向量数据库：

docker pull qdrant/qdrant

docker run -p -d 6333:6333 -p 6334:6334 \

-v $(pwd)/qdrant\_storage:/qdrant/storage:z \

qdrant/qdrant

图数据库：

docker run --name neo4j -e NEO4J\_AUTH=neo4j/neo4j -p 7474:7474 -p 7687:7687 -d neo4j

配置参数：Configuration Parameters

config = {

"graph\_store": {

"provider": "neo4j",

"config": {

"url": "neo4j+s://---",

"username": "neo4j",

"password": "---"

}

},

"embedder": {

"provider": "huggingface",

"config": {

"model": "BAAI/bge-small-zh-v1.5"

}

},

"llm": {

"provider": "deepseek",

"config": {

"model": "deepseek-chat", # default model

"temperature": 0.2,

"max\_tokens": 2000,

"top\_p": 1.0

}

},

"vector\_store": {

"provider": "milvus",

"config": {

"collection\_name": "test",

"embedding\_model\_dims": "512",

"url": "127.0.0.1",

}

}

}