

AI大模型应用：NLP与大模型

2025年

内部资料，
请勿外传

Agent与MCP

OpenAI Agent 执行流程:

1. Call Model: Send conversation history + new input to the LLM

2. Receive Response: Model returns text, tool calls, or handoffs

3. Check Exit Conditions:

1. If `output_type` is defined: Loop until structured output of that type is returned
2. If `output_type` is `None` (plain text): Exit on first response without tool calls/handoffs

4. Process Tool Calls: Execute any tool calls in parallel

via `asyncio.gather`

5. Append Results: Add tool outputs to conversation history

6. Repeat: Continue loop (max `max_turns` iterations, default unlimited)

```
tool_use_behavior: (
    Literal["run_llm_again", "stop_on_first_tool"] | StopAtTools |
) = "run_llm_again"
"""
```

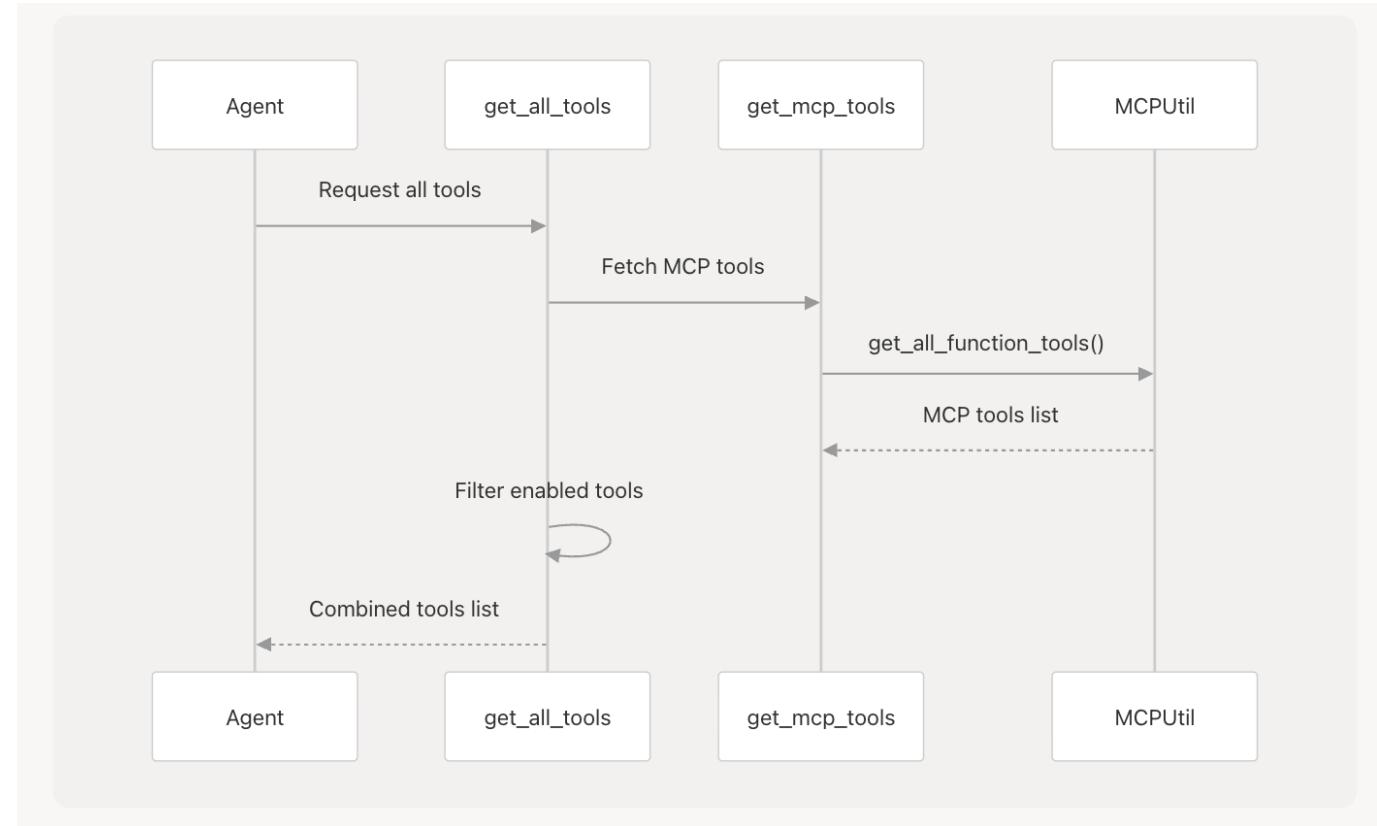
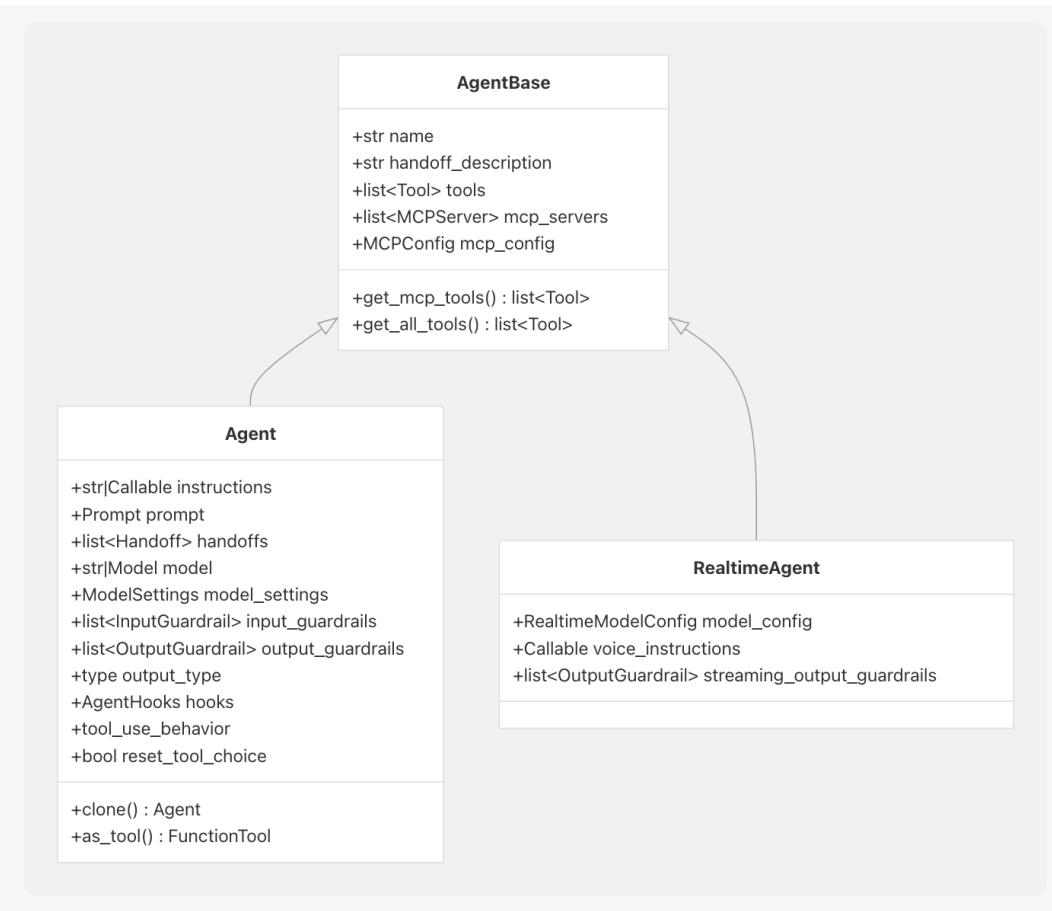
This lets you configure how tool use is handled.

- `"run_llm_again"`: The default behavior. Tools are run, and then the LLM gets to respond.
- `"stop_on_first_tool"`: The output from the first tool call is treated as the final output. In other words, it isn't sent back to the LLM for further processing.
- A `StopAtTools` object: The agent will stop running if any of the tools in `stop_at_tool_names` is called. The final output will be the output of the first matching tool. The LLM does not process the result of the tool call.
- A function: If you pass a function, it will be called with the results of the tool calls. It must return a `ToolsToFinalOutputResult`, which calls `result` in a final output.

NOTE: This configuration is specific to `FunctionTools`. Hosted tools like web search, etc. are always processed by the LLM.

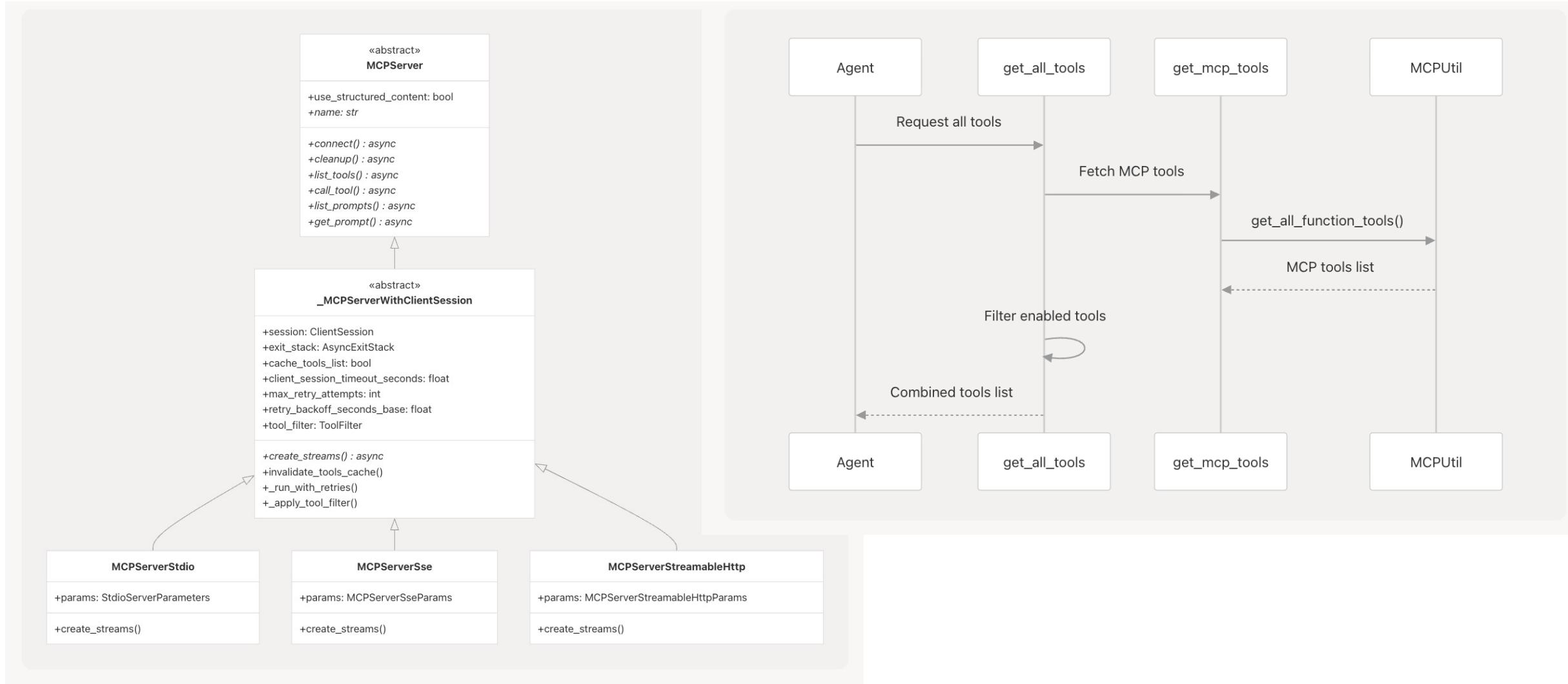
https://openai.github.io/openai-agents-python/ref/agent/#agents.agent.Agent.output_type

Agent与MCP



<https://deepwiki.com/openai/openai-agents-python/11.2-mcp-server-implementations>

Agent与MCP

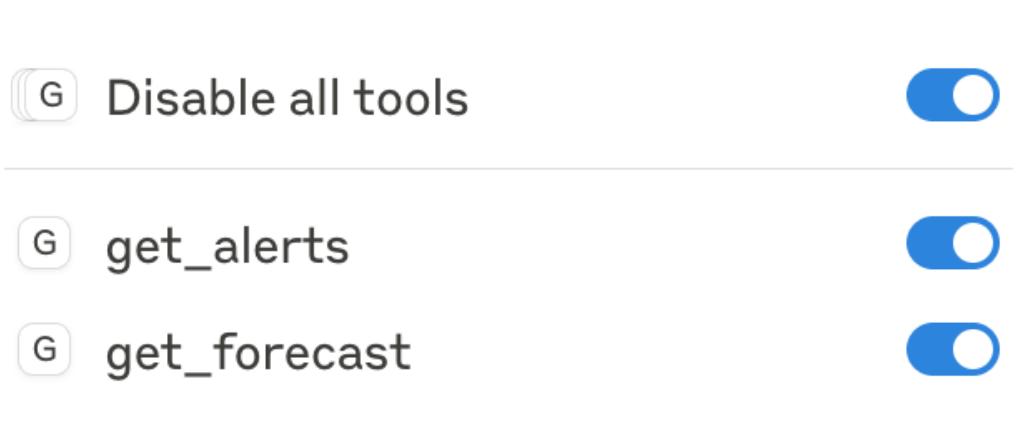


<https://deepwiki.com/openai/openai-agents-python/11.2-mcp-server-implementations>

Agent与MCP

	Model Context Protocol (MCP)	OpenAI Tools (Function Calling)
核心理念	开放标准 (Open Standard)	模型特性 (Model Feature)
目标	提供一个通用的、模型无关的标准化接口，用于上下文获取和工具交互。重点在互操作性。	旨在为 OpenAI 模型提供一个可靠的方式来调用用户定义的函数或服务。重点在模型能力扩展。
设计侧重	上下文与数据 : 更侧重于 LLM 获取和管理执行任务所需的外部上下文信息 (数据源、知识库等)。	功能调用与执行 : 更侧重于 LLM 生成特定格式的 JSON 来触发外部代码执行。
互操作性	高。作为开放标准，旨在允许任何 LLM、任何平台或应用都能采纳并使用同一套协议。	低/有限。专为 OpenAI 的模型架构设计，其他模型通常需要自己的定制实现。
实现方式	协议规范 (Protocol Specification) ，定义了数据和交互的结构。	模型训练与 API 设计 ，模型被训练来理解和生成工具调用 JSON。

Agent与MCP



What's happening under the hood

When you ask a question:

1. The client sends your question to Claude
2. Claude analyzes the available tools and decides which one(s) to use
3. The client executes the chosen tool(s) through the MCP server
4. The results are sent back to Claude
5. Claude formulates a natural language response
6. The response is displayed to you!

如果MCP Server 工具数量过多时，工具描述占用大量 Token，影响模型的推理效率和上下文容量。

- ✓ 每个工具都需要一个清晰、准确、且**面向模型的**自然语言描述。
- ✓ 使用向量搜索技术，快速找出与用户请求**语义相似度最高的** N 个工具。
- ✓ 工具可以被标记或分类，并且不构建一个拥有所有工具的“万能 Agent”。

<https://modelcontextprotocol.io/docs/develop/build-server#what%20happening-under-the-hood>
<https://demiliani.com/2025/09/04/model-context-protocol-and-the-too-many-tools-problem/>

Agent与MCP

Tool choice

By default the model will determine when and how many tools to use. You can force specific behavior with the `tool_choice` parameter.

- 1 **Auto:** (Default) Call zero, one, or multiple functions. `tool_choice: "auto"`
- 2 **Required:** Call one or more functions. `tool_choice: "required"`
- 3 **Forced Function:** Call exactly one specific function.
`tool_choice: {"type": "function", "name": "get_weather"}`
- 4 **Allowed tools:** Restrict the tool calls the model can make to a subset of the tools available to the model.

When to use allowed_tools

You might want to configure an `allowed_tools` list in case you want to make only a subset of tools available across model requests, but not modify the list of tools you pass in, so you can maximize savings from [prompt caching](#).

```
1 "tool_choice": {  
2     "type": "allowed_tools",  
3     "mode": "auto",  
4     "tools": [  
5         { "type": "function", "name": "get_weather" },  
6         { "type": "function", "name": "search_docs" }  
7     ]  
8 }  
9 }
```

Parallel function calling

 Parallel function calling is not possible when using [built-in tools](#).

The model may choose to call multiple functions in a single turn. You can prevent this by setting `parallel_tool_calls` to `false`, which ensures exactly zero or one tool is called.

Note: Currently, if you are using a fine tuned model and the model calls multiple functions in one turn then [strict mode](#) will be disabled for those calls.

Note for `gpt-4.1-nano-2025-04-14` : This snapshot of `gpt-4.1-nano` can sometimes include multiple tools calls for the same tool if parallel tool calls are enabled. It is recommended to disable this feature when using this nano snapshot.

一次对话调用默认使用一个工具；

- ✓ 多工具调用暂时支持的不好；
- ✓ 嵌套工具调用暂时不支持；

<https://platform.openai.com/docs/guides/function-calling>

https://openai.github.io/openai-agents-python/ref/model_settings/#agents.model_settings.ModelSettings.tool_choice

ChatBI应用

ChatBI支持通过对话生成对应图表和分析结论，大幅提升数据分析效率。具备多轮对话与智能追问、输入联想和猜你想问、SQL 校验与便捷分享等功能。ChatBI 通过对话即可完成数据分析，大幅提升数据分析效率。

- ✓ **无缝对接各类数据源**，包括数据库 MySQL、MariaDB、PostgreSQL、SQL Server，数据湖计算 DLC，云数据仓库 Doris、PostgreSQL、ClickHouse。支持 API 数据源，以及上传本地 Excel 文件、连接腾讯文档数据源。
- ✓ 提供便捷的多表关联、字段计算、数据聚合、字典表、SQL 建表等数据加工能力，让企业各类职业人员都可快速完成数据计算从而配置出需要的报表，**特别是让业务人员无需通过 SQL 语句，只需简单拖拽即可实现数据表关联建模。**
- ✓ **支持柱状图、条形图、饼图、百分比、地图等30+常用数据图表组件**，支持类 PPT 操作，通过拖拉拽的方式构建样式丰富、展现优雅的数据分析看板，内置多种经典主题，可快速切换复用，同时支持自定义个性化样式。

<https://cloud.tencent.com/document/product/590/107689>

多轮对话

接着上文继续提问，ChatBI能结合上下文回答



智能反问

提问太模糊时，ChatBI能智能反问，澄清意图



输入联想

简单输入关键词，就能快速找到想要的提问



猜你想问

智能推荐相关问题，持续分析不间断



数据解读

智能解读数据描述和特征，使用户更“懂”数据



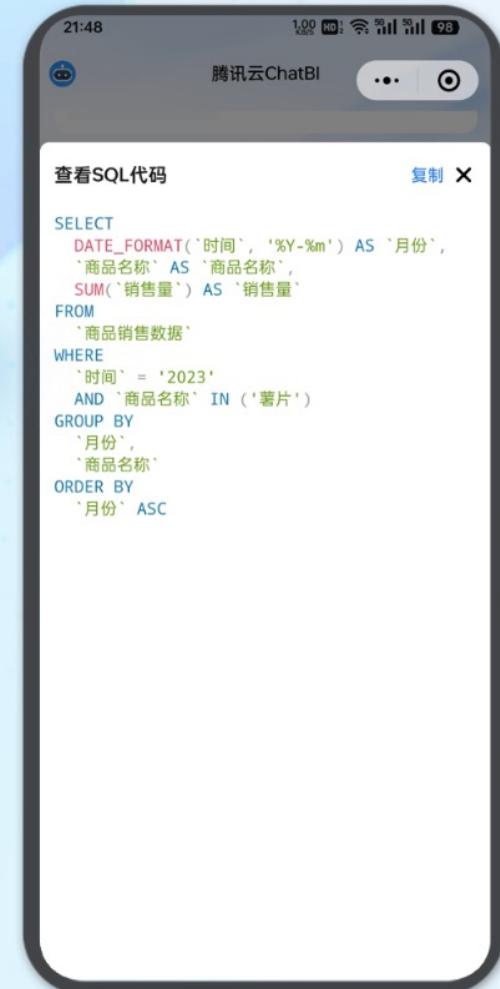
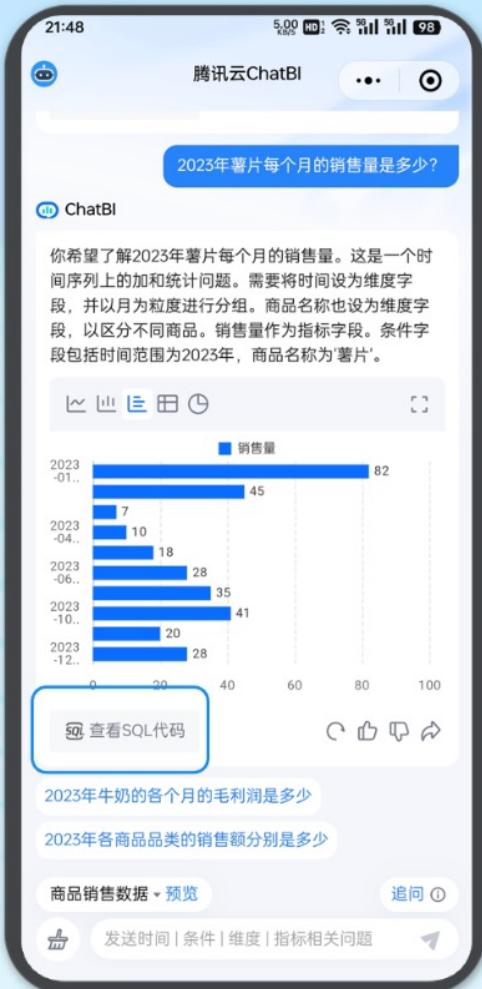
波动归因

智能计算数据波动，精准拆解波动根因



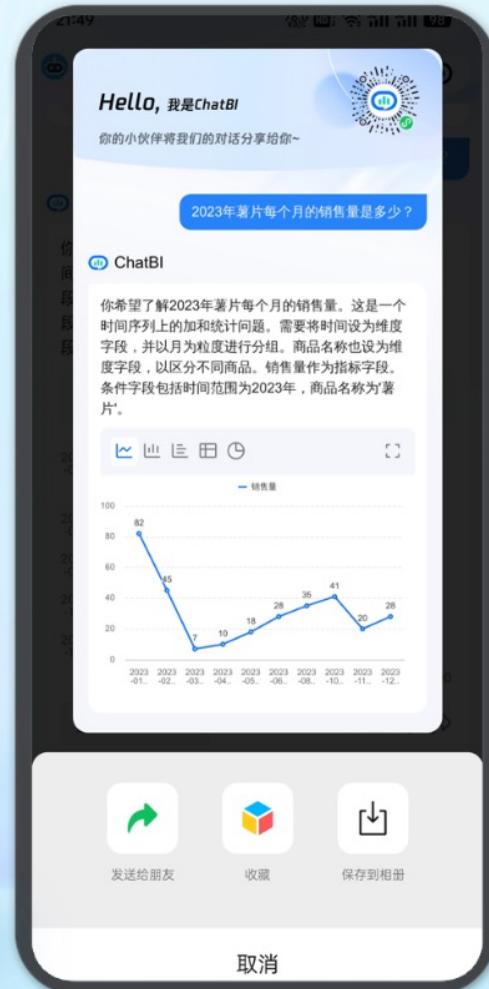
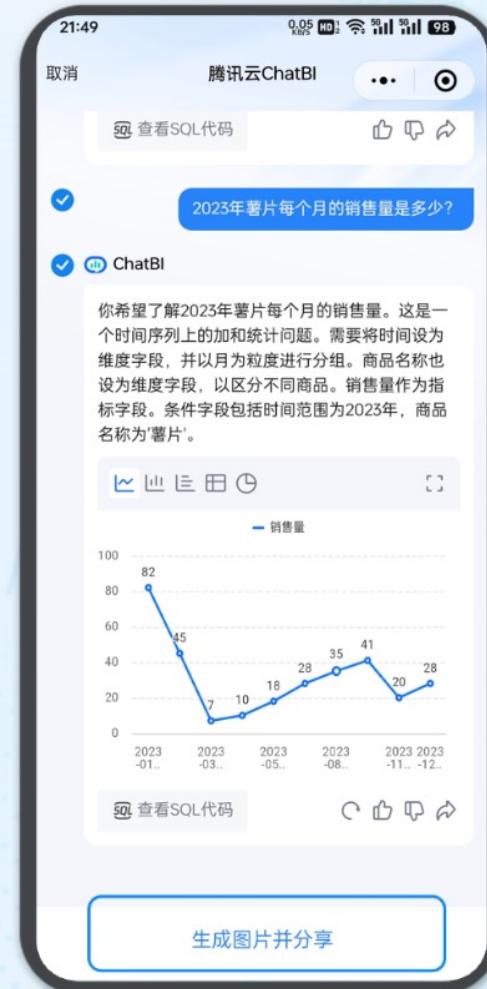
查看SQL

可查看结论背后的SQL，校验结果是否正确



分享保存

灵活选择分享内容，一键分享或保存截图



行业落地案例：去哪儿网

步骤1：进行数据字段整理；

步骤2：编写初步SQL Agent提示词

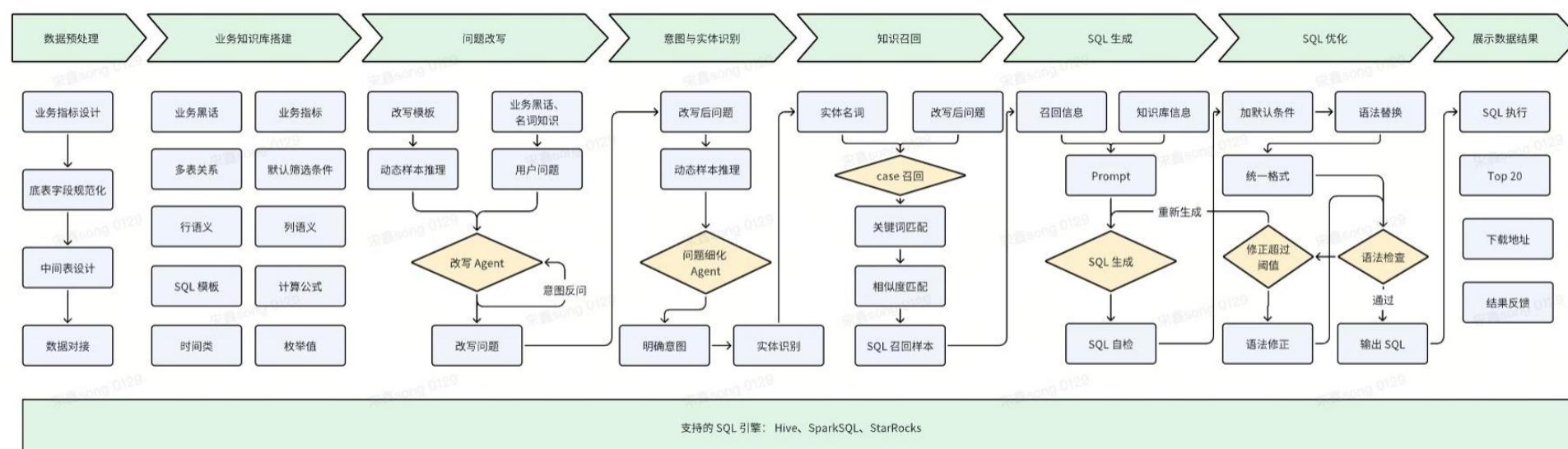
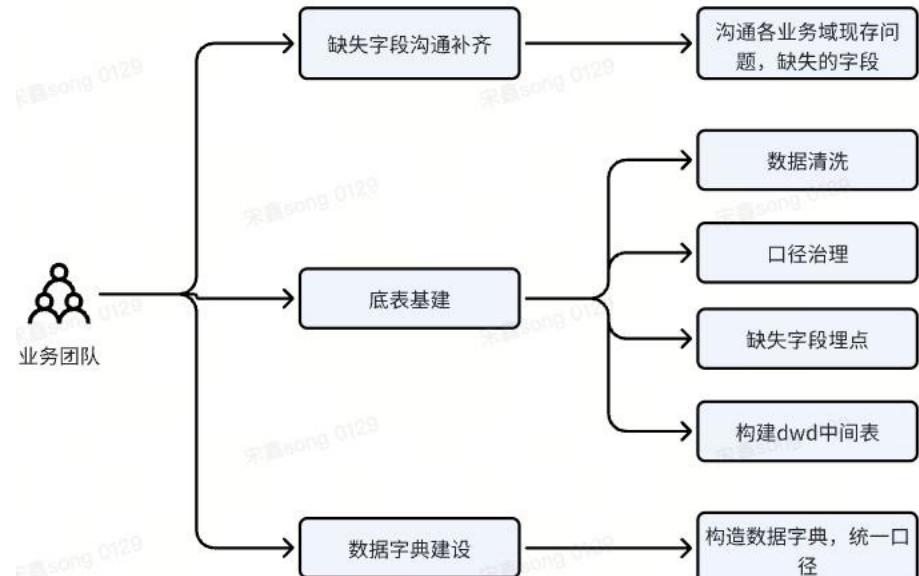
步骤3：拆分Agent 为 多Agent框架；

生成SQL -> 优化SQL

步骤4：增加React思考与循环机制

步骤5：增加反问与路由Agent

步骤6：增加RAG知识注入；



行业落地案例：腾讯游戏

步骤1：进行数据字段整理；

步骤2：需求拆解与分析

步骤3：历史知识检索与排序

步骤4：SQL生成与验证

步骤5：SQL执行与可视化

用户界面

统计：国庆节10.1-10.7，游戏内各个玩法，按每天的参与率排名+次日留存排名+七日留存排名，算一个总排名
输出：统计日期、玩法、参与率排名、次日留存排名、七日留存排名、总排名

需求拆解

SQL生成

最终生成SQL

```
SQL
132     from udattach1.dws_demo_core_mode_di
133     where ds >= '20241001' and ds <= '20241013' and platid = 255
134     group by ds, sub_mode_id, vopenid
135   )
136
137   select
138     ds as 'ds',
139     mode as 'mode',
140     vopenid as 'vopenid'
141     from game_mode_participation
142
143   ) s1
144   on s1.vopenid = s3.vopenid
145   and s3.ds = date_add(s1.ds, 6) -- 七留日期为当后的第6天
146   where s1.ds between '20241001' and '20241007' -- 仅计算2024.10.1-7
147   group by s1.ds, s1.mode
148
149   )
150
151   -- 合并参与率和留存率排名，计算总排名
152   select
153     p.ds as 'ds',
154     p.mode as 'mode',
155     p.participation_rate_rank as 'participation_rate_rank',
156     r.next_day_retention_rank as 'next_day_retention_rank',
157     r.seven_day_retention_rank as 'seven_day_retention_rank',
158     r.next_day_retention_rank + r.seven_day_retention_rank as 'total_rank'
159     from participation_rate_rank p
160     join retention_rank r on p.ds = r.ds and p.mode = r.mode
161     order by p.ds, total_rank
```

生成SQL

执行SQL

分析思路拆解

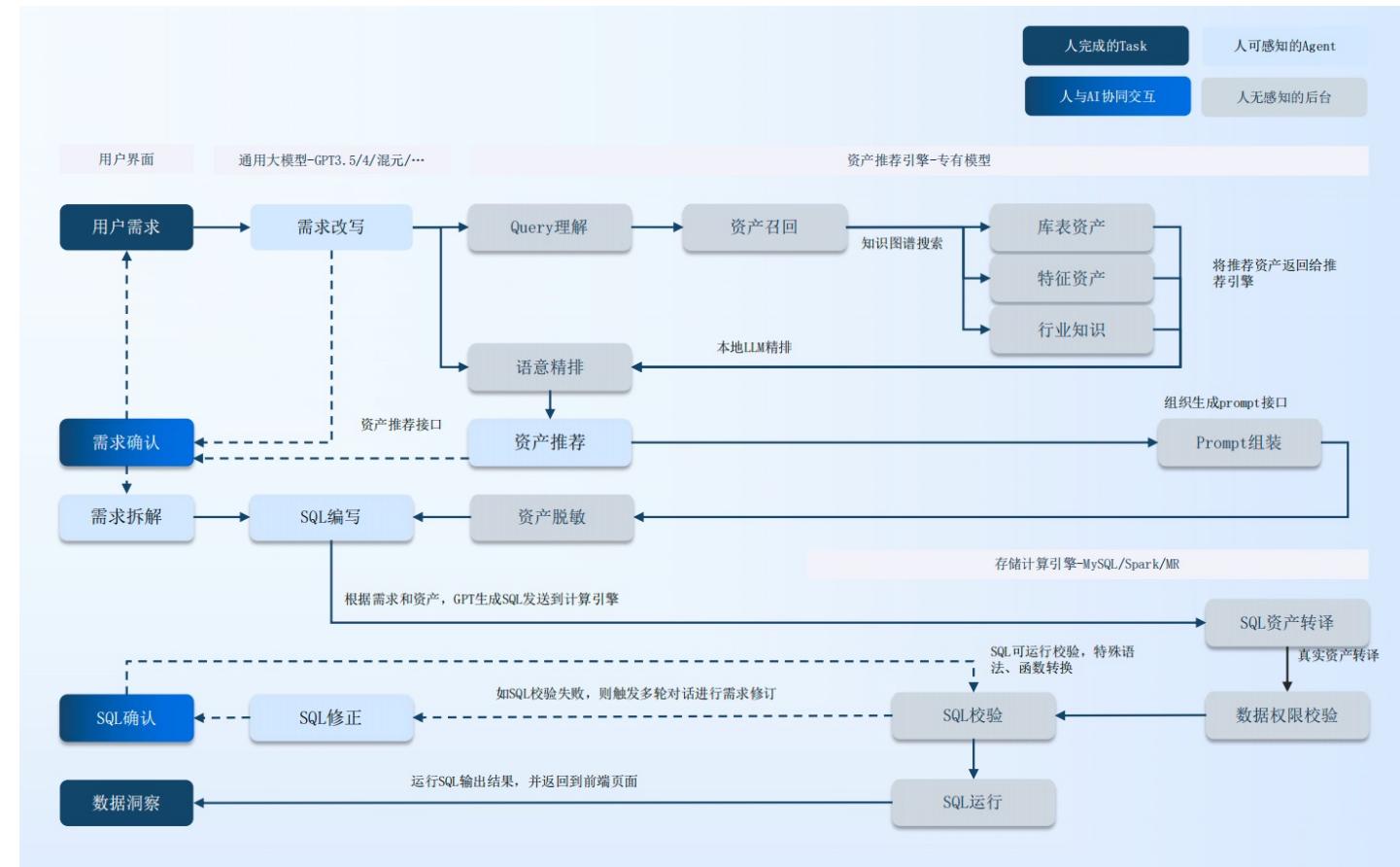
数据包1 | 游戏玩法参与明细数据提取
执行：查询10.1~10.7每天的游戏玩法参与明细数据
输出：统计日期、玩法名、参与用户ID
特征：游戏玩法参与

数据包2 | 活跃用户明细数据提取
执行：查询10.1~10.7每天的活跃用户明细数据
输出：统计日期、活跃用户ID
特征：活跃用户

数据包3 | 玩法参与率计算
输入：数据包1、数据包2
执行：计算10.1~10.7每天的游戏玩法参与率，参与率=当天参与玩法人数/当天游戏活跃人数
输出：统计日期、玩法名、参与率

数据包4 | 玩法次留和七留计算
输入：数据包3
执行：计算10.1~10.7每天的游戏玩法次留和七留，次留=当天参与且当后的第1天也参与玩法人数/当天参与后的一天也参与玩法人数
输出：统计日期、玩法名、次留、七留

结果 | 游戏玩法参与率和留存率排名
输入：数据包3、数据包4
执行：根据参与率、次留、七留分别排名，然后计算排名总和，最后根据排名总和进行排序
输出：统计日期、玩法名、参与率排名、次日留存排名、七日留存排名、总排名



行业落地案例：网易

步骤1：进行数据字段整理；

步骤2：需求拆解与分析

步骤3：生成逻辑SQL和物理SQL

步骤4：人工参与与调整

The screenshot displays the NetEase AI Platform interface with four main tabs at the top: '需求可理解' (Request Understanding), '过程可验证' (Process Verification), '用户可干预' (User Intervention), and '产品可运营' (Product Operation). The '产品可运营' tab is selected, indicated by a blue background. The interface includes a search bar, a data visualization chart showing '今年上半年华北地区每个月的利润' (Profit for each month in the first half of 2023), a '模型信息' (Model Information) dialog box, a '提示词管理' (Prompt Word Management) section, a '训练语料' (Training Corpus) table, and a '运营管理' (Operations Management) table.

需求可理解

过程可验证

用户可干预

产品可运营

标记正确回答

提示词和知识库

运营管理

查询数据表：2023超市销售数据
筛选出区域等于华北，订单日期为2023-01-01 至 2023-06-30的数据，按照年-月(订单日期)分组，计算利润求和，按照年-月(订单日期)升序排序。

今年上半年华北地区每个月的利润

订单日期	利润
2023年01月	1,413.02
2023年03月	2,230.06

模型信息

连接方式：直连、抽取
应用范围：报告、大屏、取数、数据填报、大屏Pro、数据表格
同步复杂报表：复杂报表
模型描述：该表一共包含108个房地产股票的基本信息及其在2022-2024年的各类指标情况。如果问到相关指标时，需要按照年份来看，需要返回“年份”字段，其中的公司规模“封需要，需要返回“总股本”和“总市值”、“年份”三个字段。当问到某个省份度时，需要对省份下的公司的对应的指标按年份进行求和聚合

提示词管理

训练语料

字段名称	训练语料上限	已加载语料数	添加指定值	操作
品牌+产品名称+描述	5,000	1,953	+	更新

运营管理

序号	查询数据表	问题	查看SQL	用户	创建时间	用户反馈	操作
1	23年超市数据	今年上半年华北地区每个月的利润	查看详情	顾平	2023-10-30 16:58:51	-	管理员设置
2	教学信息分析统计	教学信息分析统计表中的... 电子商务专业的女生有哪...	查看详情	朱雅君	2023-09-19 10:04:58	-	管理员设置
3	教学信息分析统计	电子商务专业的女生有哪...	查看详情	朱雅君	2023-09-19 09:51:44	正确	管理员设置

行业落地案例：FreeWheel

步骤1：进行数据字段整理；

步骤2：关键字检索与智能选表

步骤3：SQL生成与优化

步骤4：数据可视化

