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：

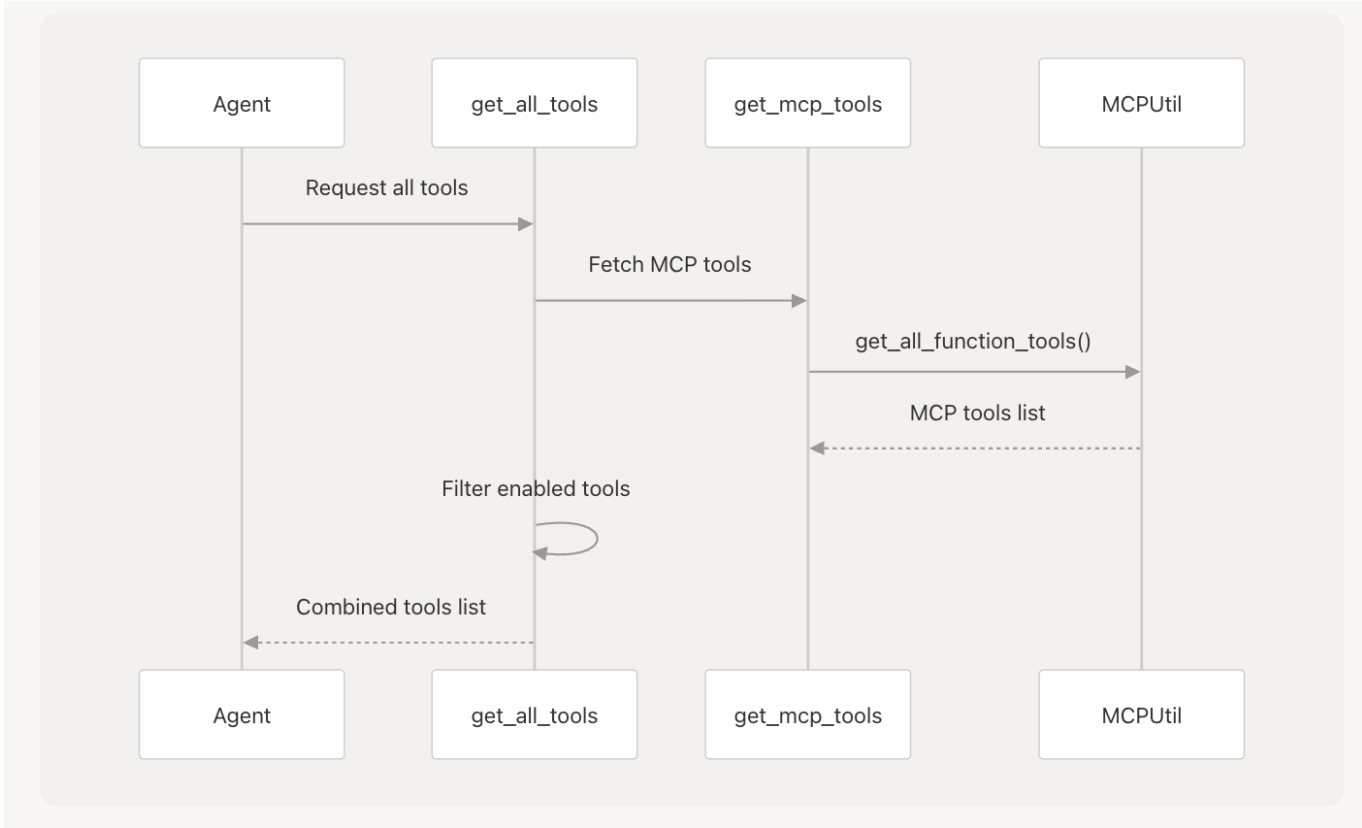
1. **Call Model**: Send conversation history + new input to the LLM
2. **Receive Response**: Model returns text, tool calls, or handoffs
3. **Check Exit Conditions**:
    1. If output_type is defined: Loop until structured output of that type is returned
    2. If output_type is None (plain text): Exit 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 th
    and gets to respond.
- "stop_on_first_tool": The output from the first tool call is trea
    In other words, it isn't sent back to the LLM for further proce
    as the final output.
- A StopAtTools object: The agent will stop running if any of the t
    `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 ru
  tool results. It must return a `ToolsToFinalOutputResult`, which
  calls result in a final output.

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

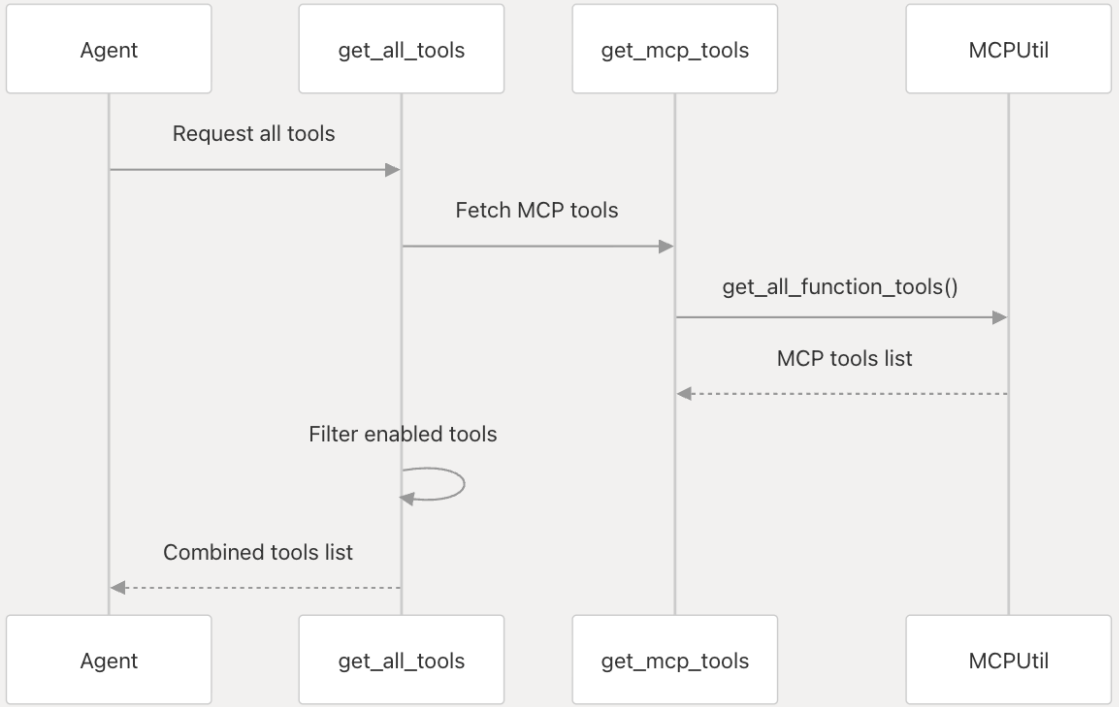
https://openai.github.io/openai-agents-python/ref/agent/#agents.agent.Agent.output_type

Agent与MCP



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

Agent与MCP

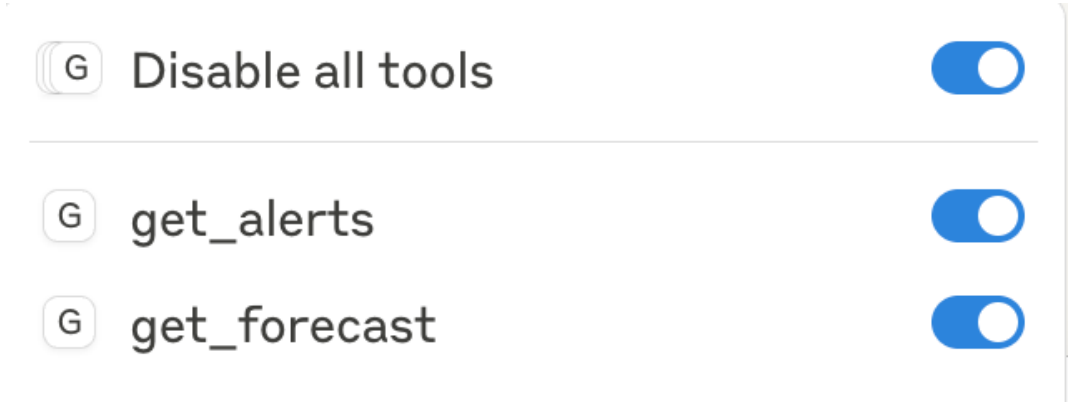


https://deepwiki.com/openai/openai-agents-python/11.2-mcp-server-implementations

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%E2%80%99s-happening-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.

```
"tool_choice": {
    "type": "allowed_tools",
    "mode": "auto",
    "tools": [
        { "type": "function", "name": "get_weather" },
        { "type": "function", "name": "search_docs" }
    ]
  }
}
```

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

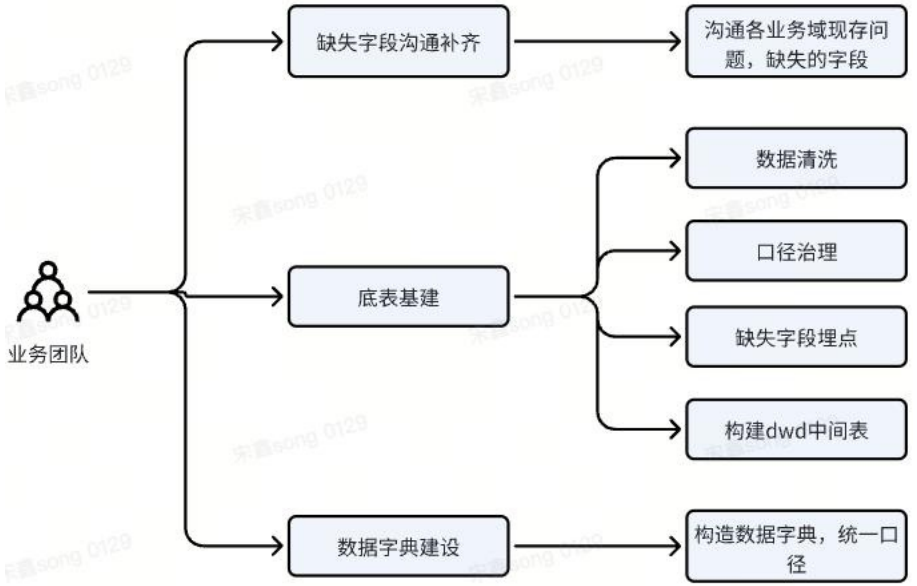
步骤3：拆分Agent 为 多Agent框架；

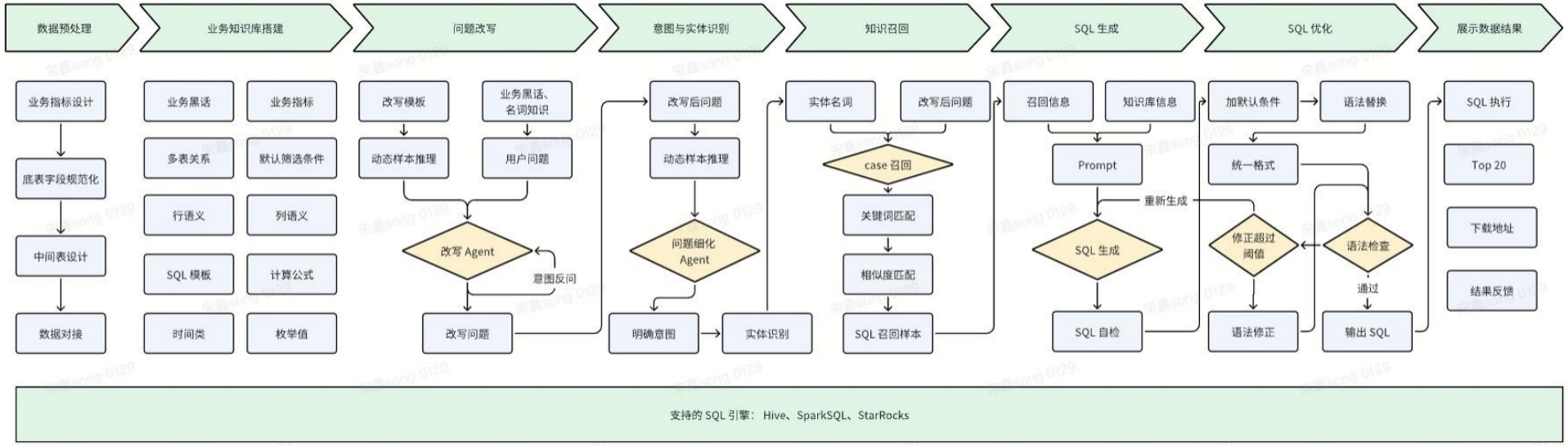
生成SQL -> 优化SQL

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

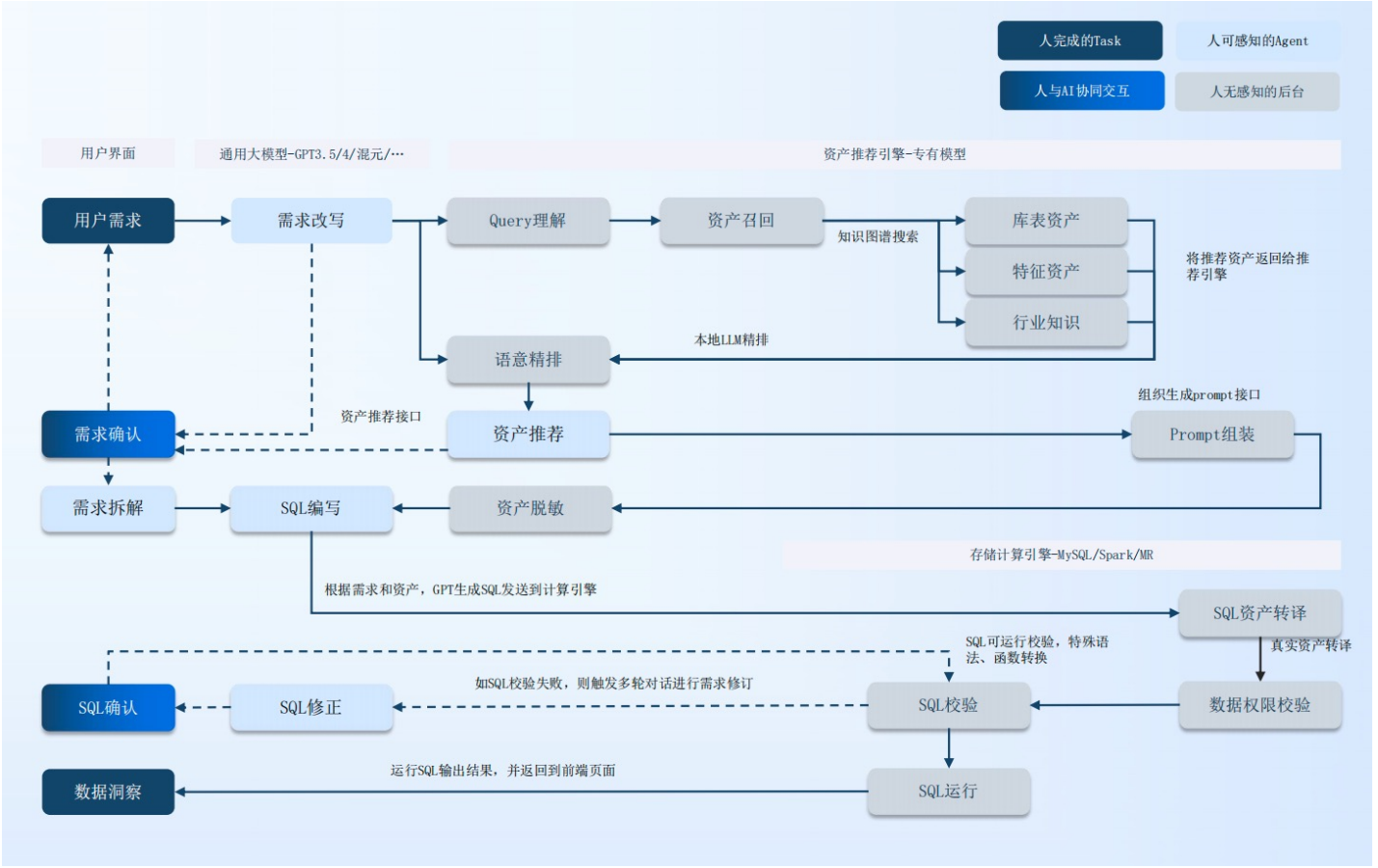
步骤1：进行数据字段整理；

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



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

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