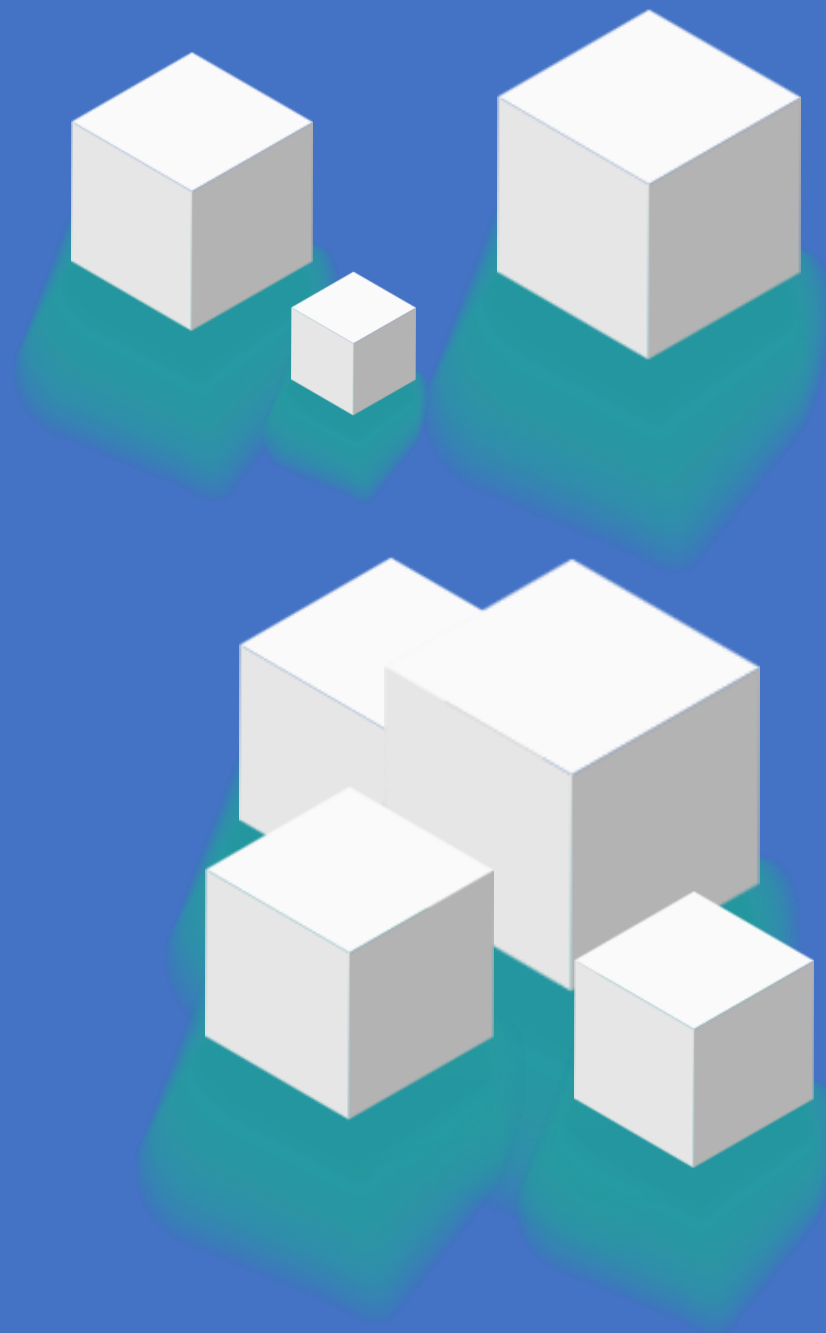
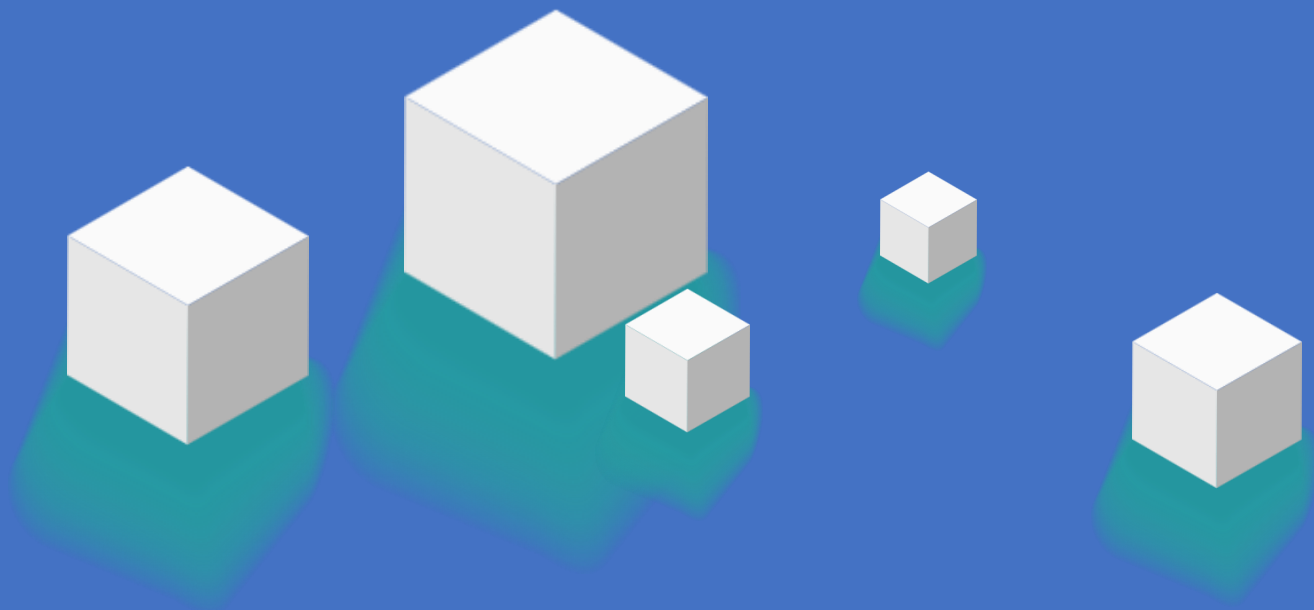


Function Calling与协作



>> 今天的学习目标

Function Calling与协作

- 什么是Function Calling
- CASE: 天气调用 (Qwen3 Function Calling)
- CASE: 门票助手

Function Calling

Thinking: Function Calling在大模型中的作用是什么？

- 扩展模型能力

大模型本身无法直接操作外部系统（如数据库、计算工具），但通过调用预设函数，可以完成：

实时数据获取（天气、股价、新闻）

复杂计算（数学运算、代码执行）

操作外部系统（发送邮件、控制智能设备）

- 结构化输出

模型可将用户自然语言请求转化为结构化参数，传递给函数。例如：

用户说“明天北京天气如何？” → 模型调用 `get_weather(location="北京", date="2025-05-06")`

- 动态决策流程

模型可根据上下文决定是否/何时调用函数，甚至链式调用多个函数（如先查天气，再推荐穿搭）。

Function Call是大模型与真实世界交互的“桥梁”，从语言理解 => 具体行动

Function Calling

Thinking: Function Calling与MCP的区别?

维度	Function Calling	MCP
定位	模型厂商私有接口（如OpenAI, Qwen）	开放协议（类似 HTTP/USB-C）
扩展性	需为每个模型单独适配	一次开发，多模型兼容
复杂性	适合简单、单次调用任务	支持多轮对话、复杂上下文管理
生态依赖	依赖特定模型（如 GPT-4）	跨模型、跨平台（如 Claude、Cursor）
安全性	依赖云端 API 密钥	支持本地化数据控制

Function Calling

Thinking: 已经有了MCP还需要Function Calling么?

简单、原子化任务使用Function Calling会更方便

- 查询天气 `get_weather(city="北京")`
- 计算数学公式 `calculate(expression="3+5")`
- 发送单条通知 `send_email(to="user@example.com")`

优势:

开发快捷: 无需配置 MCP Server, 直接通过模型 API 调用预定义函数。

低延迟: 单次请求-响应, 无需协议层开销。

MCP 可能成为主流, 但 Function Calling 作为底层能力仍将存在

CASE: 天气调用 (Function Calling)

Qwen3

Qwen3:

于 2025年4月29日 发布，包含 8 种不同规模的模型，涵盖 密集 (Dense) 和 混合专家 (MoE) 架构，全部基于 Apache 2.0 开源协议，支持免费商用

MoE 模型 (高效推理) :

- Qwen3-235B-A22B (总参数 2350 亿, 激活 220 亿) —— 旗舰级模型, 性能接近 Gemini 2.5 Pro。
- Qwen3-30B-A3B (总参数 300 亿, 激活 30 亿) —— 高效推理, 仅需 10% 激活参数即可超越前代 QwQ-32B。

密集模型 (全参数激活) :

- Qwen3-32B、14B、8B、4B、1.7B、0.6B, 其中 Qwen3-4B 性能媲美前代 Qwen2.5-72B

Qwen3

	Qwen3-235B-A22B <i>MoE</i>	Qwen3-32B <i>Dense</i>	OpenAI-o1 <i>2024-12-17</i>	Deepseek-R1	Grok 3 Beta <i>Think</i>	Gemini2.5-Pro	OpenAI-o3-mini <i>Medium</i>
ArenaHard	95.6	93.8	92.1	93.2	-	96.4	89.0
AIME'24	85.7	81.4	74.3	79.8	83.9	92.0	79.6
AIME'25	81.5	72.9	79.2	70.0	77.3	86.7	74.8
LiveCodeBench <i>v5, 2024.10-2025.02</i>	70.7	65.7	63.9	64.3	70.6	70.4	66.3
CodeForces <i>Elo Rating</i>	2056	1977	1891	2029	-	2001	2036
Aider <i>Pass@2</i>	61.8	50.2	61.7	56.9	53.3	72.9	53.8
LiveBench <i>2024-11-25</i>	77.1	74.9	75.7	71.6	-	82.4	70.0
BFCL <i>v3</i>	70.8	70.3	67.8	56.9	-	62.9	64.6
MultiIF <i>8 Languages</i>	71.9	73.0	48.8	67.7	-	77.8	48.4

开源了两个 MoE 模型的权重：Qwen3-235B-A22B，以及Qwen3-30B-A3B

Qwen3

	Qwen3-30B-A3B <i>MoE</i>	QwQ-32B	Qwen3-4B <i>Dense</i>	Qwen2.5-72B-Instruct	Gemma3-27B-IT	DeepSeek-V3	GPT-4o <i>2024-11-20</i>
ArenaHard	91.0	89.5	76.6	81.2	86.8	85.5	85.3
AIME'24	80.4	79.5	73.8	18.9	32.6	39.2	11.1
AIME'25	70.9	69.5	65.6	15.0	24.0	28.8	7.6
LiveCodeBench <i>v5, 2024.10-2025.02</i>	62.6	62.7	54.2	30.7	26.9	33.1	32.7
CodeForces <i>Elo Rating</i>	1974	1982	1671	859	1063	1134	864
GPQA	65.8	65.6	55.9	49.0	42.4	59.1	46.0
LiveBench <i>2024-11-25</i>	74.3	72.0	63.6	51.4	49.2	60.5	52.2
BFCL <i>v3</i>	69.1	66.4	65.9	63.4	59.1	57.6	72.5
MultilF <i>8 Languages</i>	72.2	68.3	66.3	65.3	69.8	55.6	65.6

Qwen3-4B 这样的小模型也能匹敌 Qwen2.5-72B-Instruct 的性能

Qwen3

Models	Layers	Heads (Q / KV)	Tie Embedding	Context Length
Qwen3-0.6B	28	16 / 8	Yes	32K
Qwen3-1.7B	28	16 / 8	Yes	32K
Qwen3-4B	36	32 / 8	Yes	32K
Qwen3-8B	36	32 / 8	No	128K
Qwen3-14B	40	40 / 8	No	128K
Qwen3-32B	64	64 / 8	No	128K

Models	Layers	Heads (Q / KV)	# Experts (Total / Activated)	Context Length
Qwen3-30B-A3B	48	32 / 4	128 / 8	128K
Qwen3-235B-A22B	94	64 / 4	128 / 8	128K

开源了两个 MoE 模型的权重

此外，6个 Dense 模型也已开源，包括 Qwen3-32B、Qwen3-14B、Qwen3-8B、Qwen3-4B、Qwen3-1.7B 和 Qwen3-0.6B，均在 Apache 2.0 许可下开源。

Summary

Qwen3具有广泛的应用场景：

- 企业级 AI（金融分析、智能客服）
- 端侧 AI（手机、IoT 设备本地部署）
- 代码生成 & AI 编程助手
- 多语言翻译 & 全球化应用

Apache 2.0 协议是一种宽松的自由软件许可协议：

自由使用：允许用户免费使用、修改、分发软件，适用于商业或非商业项目。

专利授权：明确授予用户与软件相关的专利权利（贡献者自动授权，避免专利诉讼风险）。

衍生作品：允许修改代码并闭源发布（需遵守协议条款）。

Qwen3具有广泛的应用场景，且具有商业友好性，允许将授权代码融入专有软件，无需公开衍生作品代码，适合企业使用。

CASE：天气调用（Function Calling）

TO DO：使用Qwen3调用高德Function Calling，查询天气

实现通过 Qwen3 与 天气工具集成，自动调用高德地图天气 API 查询指定城市的实时天气信息，并输出结果。

Thinking：关键步骤都有哪些？

Step1，设置 DashScope API Key

```
dashscope.api_key = "sk-xxxxxx"
```

设置通义千问的 API Key，用于身份认证。

可以调用的模型API列表 <https://help.aliyun.com/zh/model-studio/models>

qwen-turbo-latest 始终等同最新快照版 Batch 调用半价	最新版	思考模式 131,072 非思考模式 1,000,000	思考模式 129,024 非思考模式 1,000,000	8,192 思维链 最长 38,912	0.0003 元	思考模式 0.006 元 非思考模式 0.0006 元	各 100 万 Token 有效期：百炼 开通后 180 天内
qwen-turbo-2025-04-28 又称 qwen-turbo-0428、 Qwen3							

CASE: 天气调用 (Function Calling)

Step2, 定义 Function Tool 工具 (高德天气 API)

```
weather_tool = {  
    "type": "function",  
    "function": {  
        "name": "get_current_weather",  
        "description": "Get the current weather in a given  
location",  
        "parameters": {  
            "type": "object",  
            "properties": {  
                "location": {  
                    "type": "string",  
                    "description": "The city name, e.g. 北京",
```

```
                },  
                "adcode": {  
                    "type": "string",  
                    "description": "The city code, e.g. 110000 (北京)",  
                }  
            },  
            "required": ["location"],  
        },  
    },  
}
```

定义了`get_current_weather`工具, 描述了参数 (城市名和城市代码), 用于让 Qwen3 知道如何调用外部天气查询工具。

CASE：天气调用（Function Calling）

Step3, 实现 Function Tool 工具（调用高德天气 API）

```
def get_weather_from_gaode(location: str, adcode: str = None):  
    """调用高德地图API查询天气"""  
    gaode_api_key = "xxxxxx" # 替换成你的高德API Key  
    base_url =  
    "https://restapi.amap.com/v3/weather/weatherInfo"  
    params = {  
        "key": gaode_api_key,  
        "city": adcode if adcode else location,  
        "extensions": "base",  
    }  
    response = requests.get(base_url, params=params)
```

```
if response.status_code == 200:  
    return response.json()  
else:  
    return {"error": f"Failed to fetch weather:  
{response.status_code}"}
```

通过 HTTP GET 请求调用高德天气 API，获取指定城市的实时天气。

支持通过城市名或 adcode 查询。

CASE：天气调用（Function Calling）

Step4, 主流程函数: run_weather_query

```
def run_weather_query():  
    """使用 Qwen3 + Function 查询天气"""  
    messages = [  
        {"role": "system", "content": "你是一个智能助手,  
可以查询天气信息。"},  
        {"role": "user", "content": "北京现在天气怎么样? "} ]  
    response = dashscope.Generation.call(  
        model="qwen-plus-2025-04-28",  
        messages=messages,  
        tools=[weather_tool],  
        tool_choice="auto", )
```

```
    if response.status_code == HTTPStatus.OK:  
        # 检查是否需要调用工具  
        if "tool_calls" in response.output.choices[0].message:  
            tool_call =  
                response.output.choices[0].message.tool_calls[0]  
            if tool_call["function"]["name"] ==  
                "get_current_weather":  
                # 解析参数并调用高德API  
                import json  
                args = json.loads(tool_call["function"]["arguments"])  
                location = args.get("location", "北京")  
                adcode = args.get("adcode", None)
```

CASE：天气调用（Function Calling）

```
weather_data =  
get_weather_from_gaode(location, adcode)  
    print(f"查询结果: {weather_data}")  
else:  
    print(response.output.choices[0].message.content)  
else:  
    print(f"请求失败: {response.code} -  
{response.message}")
```

整体流程：

1. 构造对话消息

- 系统消息：设定助手身份。
- 用户消息：提出天气查询请求。

2. 调用 Qwen3 大模型

- 传入模型名称、对话消息、工具定义。
- `tool_choice="auto"` 让模型自动决定是否调用工具。

3. 处理模型返回结果

- 如果模型决定调用工具（即 `tool_calls` 存在），解析工具调用参数。
 - 调用本地 `get_weather_from_gaode` 函数，获取天气数据。打印查询结果。
- 如果模型未调用工具，直接输出模型回复内容。

CASE: 门票助手

CASE： 门票助手

TO DO： 搭建门票助手，可以对门票业务进行查询

2023年4、5、6月一日门票，二日门票的销量多少？ 帮我按照周进行统计

2023年7月的不同省份的入园人数统计

帮我查看2023年10月1-7日销售渠道订单金额排名

字段名	含义说明
order_time	订单日期
account_id	预定用户ID
gov_id	商品使用人身份证号
gender	使用人性别
age	年龄
province	使用人省份
SKU	商品SKU名
product_serial_no	商品ID
eco_main_order_id	订单ID
sales_channel	销售渠道
status	商品状态
order_value	订单金额
quantity	商品数量

tkr_orders数据表

CASE： 门票助手

Thinking： 如何使用Function Call， 整体的搭建流程是怎样的？

Step1. 系统初始化

- 设置系统 prompt， 描述门票表结构和常见查询需求。
- 注册 SQL 查询工具（exc_sql）， 用于执行数据查询。

Step2. 助手实例化

使用 Qwen-Agent 的 `Assistant` 类， 加载 LLM 配置、系统 prompt 和 function_list（只包含 exc_sql）。

Step3. 设置交互模式

选择WebUI 模式， 用户通过网页输入问题， 助手自动完成 SQL 查询并返回结果， 右侧可列出常见问题。

Step4. Function Call 机制

- 用户输入自然语言问题。
- LLM 解析意图并自动生成 SQL 查询语句。
- exc_sql 工具被自动调用， 执行 SQL 并返回查询结果。
- 结果通过终端或 WebUI 展示给用户。

CASE: 门票助手 (system_prompt)

```
system_prompt = ""我是门票助手，以下是关于门票订单表相关的字段，我可能会编写对应的SQL，对数据进行查询
```

```
-- 门票订单表
```

```
CREATE TABLE tkt_orders (
```

```
    order_time DATETIME,      -- 订单日期
```

```
    account_id INT,           -- 预定用户ID
```

```
    gov_id VARCHAR(18),       -- 商品使用人ID (身份证号)
```

```
    gender VARCHAR(10),       -- 使用人性别
```

```
    age INT,                  -- 年龄
```

```
    province VARCHAR(30),     -- 使用人省份
```

```
    SKU VARCHAR(100),         -- 商品SKU名
```

```
    product_serial_no VARCHAR(30), -- 商品ID
```

```
    eco_main_order_id VARCHAR(20), -- 订单ID
```

```
    sales_channel VARCHAR(20),  -- 销售渠道
```

```
    status VARCHAR(30),        -- 商品状态
```

```
.....
```

Thinking: 是否能找到原始的数据表metadata, 是否有常用的术语需要提供, 方便后续撰写SQL

CASE：门票助手（exc_sql工具注册）

```
from qwen_agent.tools.base import BaseTool, register_tool
```

```
@register_tool('exc_sql')
```

```
class ExcSQLTool(BaseTool):
```

```
    """
```

```
    SQL查询工具，执行传入的SQL语句并返回结果。
```

```
    """
```

```
    description = '对于生成的SQL，进行SQL查询'
```

```
    parameters = [{
```

```
        'name': 'sql_input',
```

```
        'type': 'string',
```

```
        'description': '生成的SQL语句',
```

```
        'required': True
```

```
    ]
```

```
def call(self, params: str, **kwargs) -> str:
```

```
    import json
```

```
    args = json.loads(params)
```

```
    sql_input = args['sql_input']
```

```
    database = args.get('database', 'ubr')
```

```
    # 创建数据库连接
```

```
    engine = .....
```

```
    try:
```

```
        df = pd.read_sql(sql_input, engine)
```

```
        # 返回前10行，防止数据过多
```

```
        return df.head(10).to_markdown(index=False)
```

```
    except Exception as e:
```

```
        return f"SQL执行出错: {str(e)}"
```

Thinking：这个地方是否可以通过 用户的需求，自适应进行调整？

CASE： 门票助手 （Assistant 初始化与 function_list）

```
def init_agent_service():
    """初始化门票助手服务"""
    llm_cfg = {
        'model': 'qwen-turbo-2025-04-28',
        'timeout': 30,
        'retry_count': 3,
    }
    try:
        bot = Assistant(
            llm=llm_cfg,
            name='门票助手',
            description='门票查询与订单分析',
            system_message=system_prompt,
            function_list=['exc_sql'], # 只传工具名字字符串
```

```
        print("助手初始化成功！ ")
        return bot
    except Exception as e:
        print(f"助手初始化失败: {str(e)}")
        raise
```

CASE：门票助手（Web 图形界面）

```
def app_gui():
    """图形界面模式，提供 Web 图形界面"""
    ...
    bot = init_agent_service()
    chatbot_config = {
        'prompt.suggestions': [
            '2023年4、5、6月一日门票，二日门票的销量多少？帮我按照周进行统计',
            '2023年7月的不同省份的入园人数统计',
            '帮我查看2023年10月1-7日销售渠道订单金额排名',
        ]
    }
```

```
WebUI(
    bot,
    chatbot_config=chatbot_config
).run()
```

function call 相关逻辑全部通过 Assistant 的 function_list 机制与工具注册实现。
exc_sql 工具负责 SQL 查询，注册后可被 LLM调用。
支持终端(app_tui)和 WebUI 两种交互方式。

CASE：门票助手

2023年4、5、6月一日门票，二日门票的销量多少？帮我按照周进行统计



user

- ▶ Start calling tool "exc_sql" ...
- ▶ Finished tool calling.



以下是2023年4、5、6月一日门票和二日门票的周销量统计：

周数	一日门票销量	二日门票销量
2023-13	3367	2732
2023-14	19444	14267
2023-15	20123	14888
2023-16	19399	15007
2023-17	19489	14603
2023-18	19933	15249
2023-19	19422	14783
2023-20	19594	14428
2023-21	19163	14339
2023-22	19573	14297

▼ Start calling tool "exc_sql" ...

```
{
  "sql_input": "SELECT DATE_FORMAT(order_time, '%Y-%U') AS week, \n SUM(CASE WHEN SKU LIKE 'Universal Studios Beijing One-Day%' THEN quantity ELSE 0 END) AS one_day_ticket_sales, \n SUM(CASE WHEN SKU LIKE 'USB%' THEN quantity ELSE 0 END) AS two_day_ticket_sales \nFROM tkt_orders \nWHERE order_time BETWEEN '2023-04-01' AND '2023-06-30' \nGROUP BY week \nORDER BY week;"
}
```

▼ Finished tool calling.

week	one_day_ticket_sales	two_day_ticket_sales
2023-13	3367	2732
2023-14	19444	14267
2023-15	20123	14888
2023-16	19399	15007
2023-17	19489	14603
2023-18	19933	15249
2023-19	19422	14783
2023-20	19594	14428
2023-21	19163	14339
2023-22	19573	14297

如果需要更详细的分析或进一步的数据，请告诉我！



门票助手



CASE: 门票助手

2023年7月的不同省份的入园人数统计



user

- ▶ Start calling tool "exc_sql" ...
- ▶ Finished tool calling.



以下是2023年7月不同省份的入园人数统计：

省份	入园人数
北京市	99893
江苏省	47290
河北省	46602
山东省	43762
广东省	42601
上海市	40782
浙江省	38042
辽宁省	24910
湖北省	24231
四川省	22672

- ▼ Start calling tool "exc_sql" ...

```

{"sql_input": "SELECT province, SUM(quantity) AS visitor_count \nFROM tkt_orders \nWHERE order_time BETWEEN '2023-07-01' AND '2023-07-31' \nGROUP BY province \nORDER BY visitor_count DESC;"}

```

- ▼ Finished tool calling.

province	visitor_count
北京市	99893
江苏省	47290
河北省	46602
山东省	43762
广东省	42601
上海市	40782
浙江省	38042
辽宁省	24910
湖北省	24231
四川省	22672

如果需要更详细的分析或进一步的数据，请告诉我！



门票助手

CASE： 门票助手

帮我查看2023年10月1-7日销售渠道订单金额排名



user

▼ Start calling tool "exc_sql" ...

{
 "sql_input": "SELECT sales_channel, SUM(order_value) AS total_order_value \nFROM tkt_orders \nWHERE order_time BETWEEN '2023-10-01' AND '2023-10-07' \nGROUP BY sales_channel \nORDER BY total_order_value DESC;"}
}

▼ Finished tool calling.

sales_channel	total_order_value
B2B_OTA	2.0915e+07
B2C_UBRAPP	1.03013e+07
B2B_TTAGIT	3.51182e+06

以下是2023年10月1日至7日销售渠道的订单金额排名：

销售渠道	订单金额
B2B_OTA	20,915,000
B2C_UBRAPP	10,301,300
B2B_TTAGIT	3,511,820

如果需要更详细的分析或进一步的数据，请告诉我！

TO DO： 快来搭建你的业务助手吧！



门票助手

打卡：业务助手



练习使用LLM/Agent + Function Calling使用，可以使用Qwen3或Qwen-Agent

如果你在本地有MySQL，可以基于自己的数据，进行业务查询

- 1) 某个月份的销量是多少？
- 2) 相比于上个月，这个月的销量环比增长多少？
- 3) 不同省份的销售额是多少？
- 4) 某个时间段，销售金额Top3的渠道是哪些？

CASE： 门票助手（可视化图表）

Thinking：如何在exc_sql查询数据之后，将数据进行可视化图表呈现？

方法1：编写新的函数 plot_data，传入 Markdown的图表，进行可视化

方法2：在原有exc_sql函数基础上，增加 plot_data的功能，返回结果包括：数据表markdown 以及 可视化的图表png

Thinking：方法1 的问题是什么？

Markdown传参可能更大，另外要绘制的x, y参数也不一定能传递准确

编写Markdown绘图有一定的难度，也需要先将Markdown转化为df (dataframe格式) 然后再进行绘制

在qwen-agent中，很难保存中间的 df，不同用户实例之间的维护成本可能较高

=> 采用方法2

可视化图表：实现步骤

在传统实现中，数据查询和可视化通常是分开的两个步骤（工具）：

- 先执行 SQL 查询获取数据
- 再调用可视化工具进行图表绘制

我们的优化是将两者集成到一个工具中，实现：

- 一次调用，完成查询和可视化
- 自动推断，图表类型和字段映射（x轴、y轴）
- 结果双输出，同时返回表格和图表

可视化图表：实现步骤

Step1, SQL查询获取数据

执行SQL查询

```
df = pd.read_sql(sql_input, engine)
```

生成markdown表格

```
md = df.head(10).to_markdown(index=False)
```

Step2, 自动推断图表字段

自动推断x/y字段

```
x_candidates = df.select_dtypes(include=['object']).columns.tolist()
```

```
if not x_candidates:
```

```
    x_candidates = df.columns.tolist()
```

```
x = x_candidates[0]
```

```
y_candidates = df.select_dtypes(include=['number']).columns.tolist()
```

```
y_fields = y_candidates
```

字段推断逻辑：

x轴字段： 优先选择第一个字符串类型（object）的列，如日期、分类名称等

y轴字段： 选择所有数值类型的列，支持多系列数据展示

可视化图表：实现步骤

Step3, 柱状图绘制

```
plt.figure(figsize=(8, 5))  
  
bar_width = 0.35 if len(y_fields) > 1 else 0.6  
  
x_labels = df[x].astype(str)  
x_pos = range(len(df))  
  
for idx, y_col in enumerate(y_fields):  
    plt.bar([p + idx*bar_width for p in x_pos], df[y_col], width=bar_width, label=y_col)
```

绘图逻辑：

- 创建适当大小的图表
- 根据y轴字段数量调整柱宽
- 支持多系列数据的并列柱状图
- 每个y轴字段绘制一组柱子
- 自动错开位置，避免柱子重叠

可视化图表：实现步骤

Step4, 图表样式设置

```
plt.xlabel(x)
plt.ylabel(','.join(y_fields))
plt.title(f"' & '.join(y_fields)} by {x}")
plt.xticks([p + bar_width*(len(y_fields)-1)/2 for p in x_pos], x_labels, rotation=45, ha='right')
plt.legend()
plt.tight_layout()
```

样式设置：

- 设置x轴、y轴标签
- 自动生成图表标题
- x轴标签45度倾斜，避免重叠
- 添加图例，区分多系列数据
- 调整图表布局，确保所有元素可见

可视化图表：实现步骤

Step5, 图表保存与返回

自动创建目录

```
save_dir = os.path.join(os.path.dirname(__file__), 'image_show')
```

```
os.makedirs(save_dir, exist_ok=True)
```

生成唯一文件名

```
filename = f'bar_{int(time.time()*1000)}.png'
```

```
save_path = os.path.join(save_dir, filename)
```

```
plt.savefig(save_path)
```

```
plt.close()
```

```
img_path = os.path.join('image_show', filename)
```

```
img_md = f'![柱状图]({img_path})'
```

```
return f'{md}\n\n{img_md}'
```

保存与返回逻辑：

- 自动创建图片保存目录
- 生成基于时间戳的唯一文件名
- 保存图片到本地
- 生成markdown格式的图片引用
- 返回"表格+图片"组合结果

CASE： 门票助手（可视化图表）

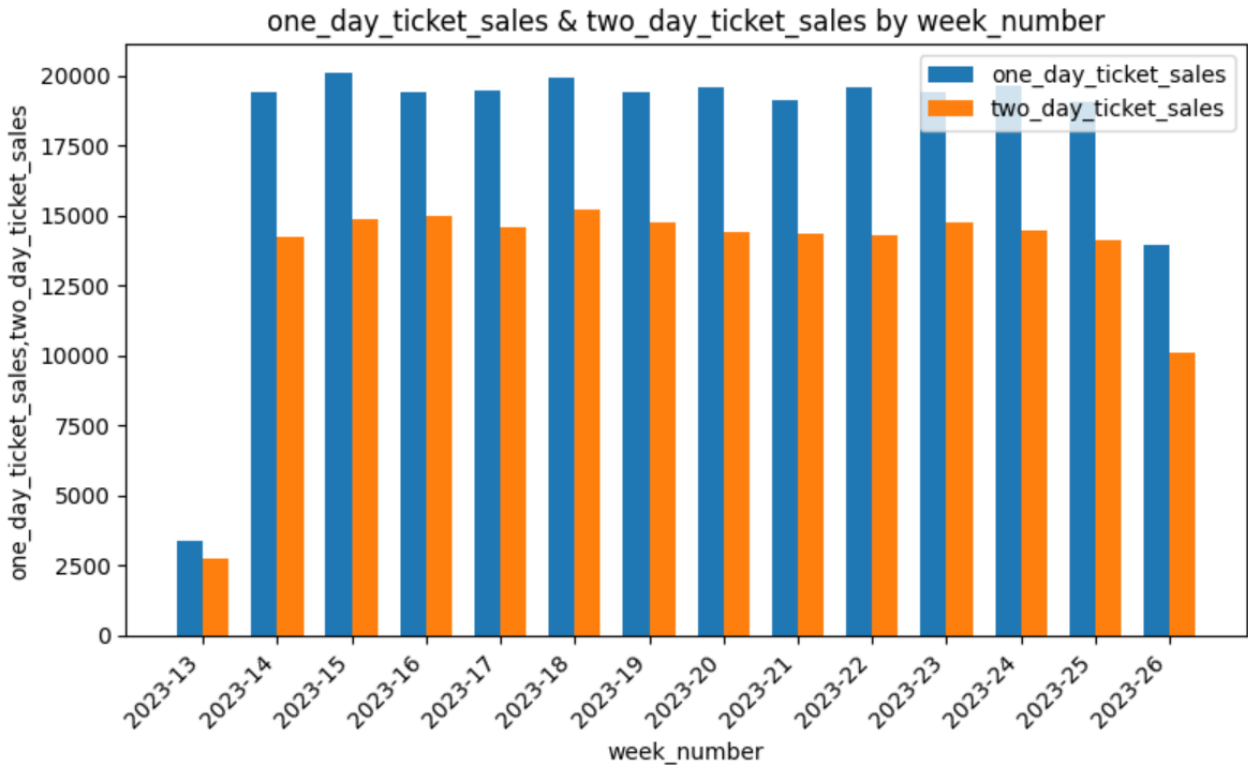
2023年4、5、6月一日门票，二日门票的销量多少？ 帮我按照周进行统计

► Start calling tool "exc_sql" ...

► Finished tool calling.

以下是2023年4、5、6月一日门票和二日门票的周销量统计：

week_number	一日门票销量	二日门票销量
2023-13	3367	2732
2023-14	19444	14267
2023-15	20123	14888
2023-16	19399	15007
2023-17	19489	14603
2023-18	19933	15249
2023-19	19422	14783
2023-20	19594	14428
2023-21	19163	14339
2023-22	19573	14297

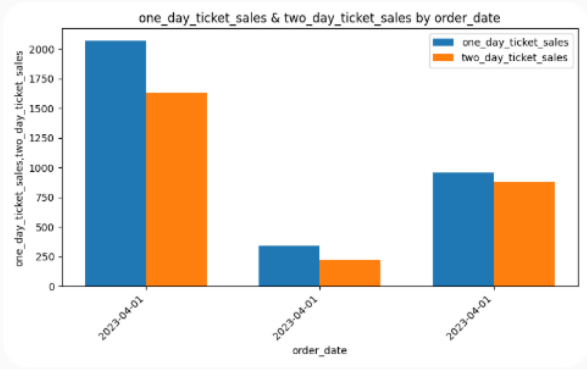


CASE： 门票助手（可视化图表）

我看到第13周数据有异常，帮我按照天、渠道的维度进行统计，给我他们的表现

- ▶ Start calling tool "exc_sql" ...
 - ▶ Finished tool calling.
- 以下是2023年第13周（即2023-04-01）一日门票和二日门票按天和销售渠道的销量统计：

order_date	sales_channel	一日门票销量	二日门票销量
2023-04-01	B2B_OTA	2069	1633
2023-04-01	B2B_TTAGIT	339	222
2023-04-01	B2C_UBRAPP	959	877



Thinking： 这里为什么提取的数据不对？

WHERE
order_time BETWEEN '2023-04-01' AND '2023-06-30'
AND DATE_FORMAT(order_time, '%Y-%U') = '2023-13'

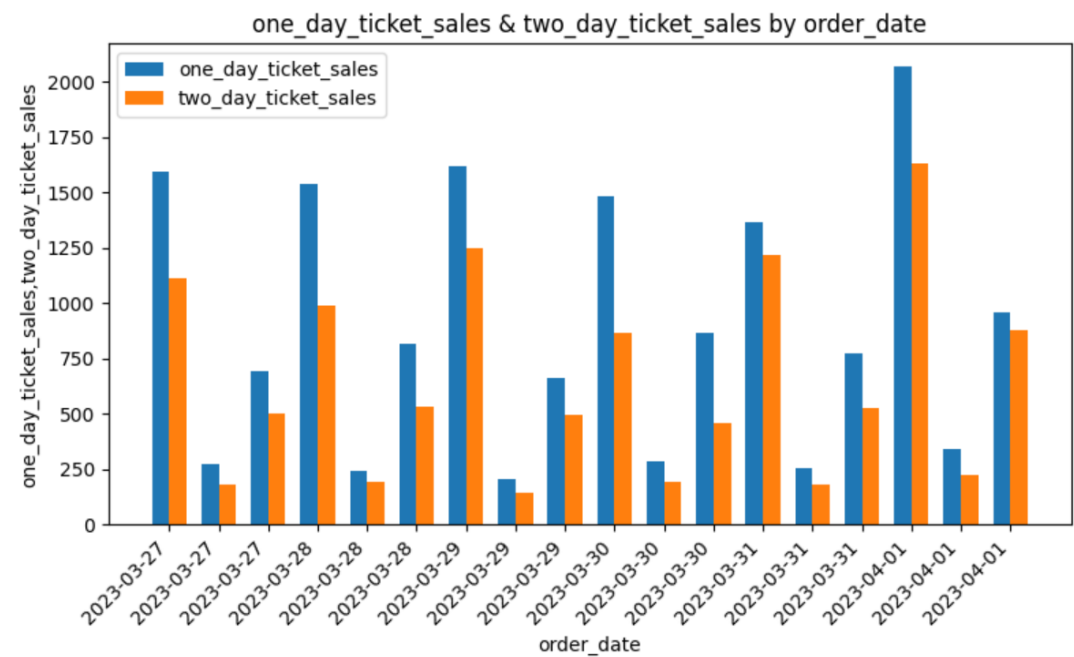
CASE： 门票助手（可视化图表）

我看到第13周（2023年3月27日-4月2日）数据有异常，帮我按照天、渠道的维度进行统计，给我他们的表现

- ▶ Start calling tool "exc_sql" ...
- ▶ Finished tool calling.

以下是2023年3月27日至4月2日（第13周）一日门票和二日门票按天和销售渠道的销量统计：

order_date	sales_channel	一日门票销量	二日门票销量
2023-03-27	B2B_OTA	1596	1111
2023-03-27	B2B_TTAGIT	274	183
2023-03-27	B2C_UBRAPP	694	499
2023-03-28	B2B_OTA	1537	991
2023-03-28	B2B_TTAGIT	245	190
2023-03-28	B2C_UBRAPP	816	535
2023-03-29	B2B_OTA	1618	1249
2023-03-29	B2B_TTAGIT	205	144
2023-03-29	B2C_UBRAPP	663	493
2023-03-30	B2B_OTA	1486	868



Thinking：如何将渠道显示出来？

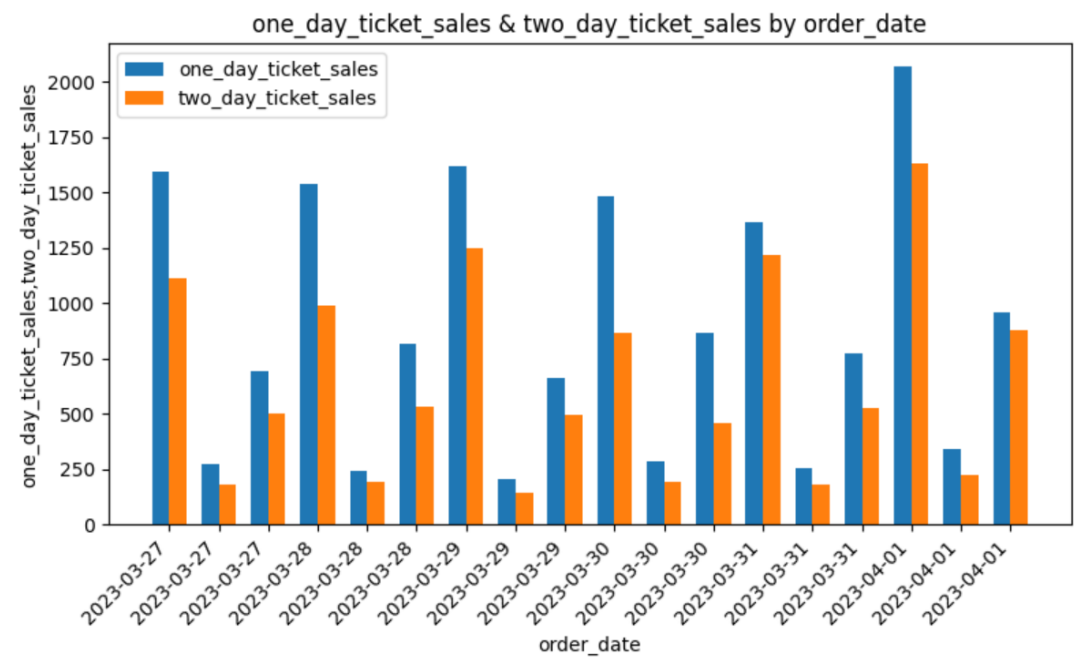
CASE： 门票助手（可视化图表）

我看到第13周（2023年3月27日-4月2日）数据有异常，帮我按照天、渠道的维度进行统计，给我他们的表现

- ▶ Start calling tool "exc_sql" ...
- ▶ Finished tool calling.

以下是2023年3月27日至4月2日（第13周）一日门票和二日门票按天和销售渠道的销量统计：

order_date	sales_channel	一日门票销量	二日门票销量
2023-03-27	B2B_OTA	1596	1111
2023-03-27	B2B_TTAGIT	274	183
2023-03-27	B2C_UBRAPP	694	499
2023-03-28	B2B_OTA	1537	991
2023-03-28	B2B_TTAGIT	245	190
2023-03-28	B2C_UBRAPP	816	535
2023-03-29	B2B_OTA	1618	1249
2023-03-29	B2B_TTAGIT	205	144
2023-03-29	B2C_UBRAPP	663	493
2023-03-30	B2B_OTA	1486	868



Thinking：如何将渠道显示出来？

CASE：门票助手（可视化图表）

Thinking：如何将渠道显示出来？

采用分组与透视

如果存在 object_columns，则用 pd.pivot_table 以 x 轴为 index，object_columns 为 columns，num_columns 为 values，自动聚合

支持多 object、多数值列，自动生成多级分组

```
def generate_chart_png(df_sql, save_path):
```

```
.....
```

Version更新：

assistant_ticket_bot-1：实现了function call调用exc_sql

assistant_ticket_bot-2：添加了基本的绘制图表功能

assistant_ticket_bot-3：进一步完善绘制图表功能，支持多类别变量的透视图可视化

Function Call的微调

Thinking: Function call是否需要微调?

通常大模型具备Function Call的能力，但有时候针对特定的Function Call理解和提参能力不强 => 可以微调

Function Call 的微调目标，是教会模型下面两件事：

- 判断是否需要调用函数（比如“查天气”需要，而“写首诗”不需要）。
- 正确提取参数并生成格式化的调用指令（比如转成JSON）。

Function Call的微调

Thinking: 微调的步骤是怎样的?

Step1, 准备数据

输入: 用户问题 (如“北京今天天气? ”) 。

函数描述: 告诉模型有哪些函数可用 (比如get_weather(city, date)) 。

期望输出:

- 需要调用时: 生成JSON格式的函数调用 (如{"name":"get_weather", "arguments":{"city":"北京", "date":"今天"}}) 。
- 不需要时: 直接生成回答 (如“你好, 今天星期一...”) 。

Step2, 模型训练

选一个擅长理解指令的预训练模型, 如Qwen。

用上述数据训练模型, 让它学会“根据问题和函数描述, 决定是否调用函数, 并生成正确格式”。

Function Call的微调

Function Call微调的关键点：

- **数据质量**：需覆盖各种场景（需要/不需要调用、参数变化等）。
- **函数描述要清晰**：模型靠描述理解函数用途。
- **避免误触发**：加入足够多“无需调用”的样本，防止模型滥用函数。

Function Call的微调

微调数据集参考：

<https://huggingface.co/datasets/NousResearch/hermes-function-calling-v1>

训练 LLM 模型根据自然语言指令执行函数调用并返回结构化输出。数据集涵盖了各种对话场景，要求 AI 代理能够解释查询并执行适当的单个或多个函数调用。

例如，当被要求“查找下周五从纽约飞往洛杉矶的航班”时，函数调用代理可以解释请求，生成必要的函数调用（例如 `search_flights`），并返回结果。

Dataset Viewer

Auto-converted to Parquet

</> API

Embed

Data Studio

Subset (5)

func_calling_singleturn · 1.89k rows

Split (1)

train · 1.89k rows

Search this dataset

id	conversations	category
string · lengths	list · lengths	string · classe
36	3	63 values
85f6c398-69c7-4df2-aed1-...	[{ "from": "system", "value": "You are a function calling AI model. You are provided with function...	IoT and Home Automation
89ef3c87-66bd-46ee-9297-...	[{ "from": "system", "value": "You are a function calling AI model. You are provided with function...	IoT and Home Automation
14657d01-d6d1-46df-8eb1-...	[{ "from": "system", "value": "You are a function calling AI model. You are provided with function...	IoT and Home Automation
c483f963-8a29-4ff0-a684-...	[{ "from": "system", "value": "You are a function calling AI model. You are provided with function...	IoT and Home Automation
81ad724a-bb74-420f-8221-...	[{ "from": "system", "value": "You are a function calling AI model. You are provided with function...	IoT and Home Automation
7d99abac-f27f-4ed2-c0cc-...	[{ "from": "system", "value": "You are a function calling AI model. You are provided with function...	IoT and Home Automation

< Previous

1

2

3

...

19

Next >

Function Call的微调

该数据集由 5 个数据集组成，分别是：

- func-calling-singleturn.json - 单轮函数调用
- func-calling.json - 多轮对话函数调用
- glaive-function-calling-5k.json - Glaive AI 更新和清理后的 Glaive 函数调用 5k 数据集
- json-mode-agentic.json - 高级 JSON 结构化输出样本
- json-mode-singleturn.json - 单轮 JSON 结构化输出样本

id	conversations	category	subc
string · lengths	list · lengths	string · classes	str
<div><div></div></div> 36100%	<div><div></div></div> 3100%	<div><div></div></div> IoT and Ho...0.8%24	
85f6c398-69c7-4df2-aed1-29d614a93a26	[{ "from": "system", "value": "You are a function calling AI model. You are provided with function signatures within <tools> </tools> XML tags. You may call one or more functions to assist with the user query. Don't make assumptions about what values to plug into functions.\n<tools>\n[{ 'type': 'function', 'function': { 'name': 'get_camera_live_feed', 'description': 'Retrieves the live feed from a specified security camera.', 'parameters': { 'type': 'object', 'properties': { 'camera_id': { 'type': 'string', 'description': 'The unique identifier for the camera.' }, 'stream_quality': { 'type': 'string', 'description': 'The desired quality of the live stream.', 'enum': ['720p', '1080p', '4k'] } }, 'required': ['camera_id'] } }, { 'type': 'function', 'function': { 'name': 'list_all_cameras', 'description': 'Lists all the security cameras connected to the home network.', 'parameters': { 'type': 'object', 'properties': { 'include_offline': { 'type': 'boolean', 'description': 'Whether to include	IoT and Home Automation	Security Management

func-calling-singleturn.json 中的某条训练样本



Thank You
Using data to solve problems

