AI 大模型开发工程师之
ChatGLM大模型开发

讲师：李希沅

目录

1 ChatGLM代码调用

01、ChatGLM API使用

步骤1： 导入相关的库

```python
from transformers import AutoTokenizer, AutoModel
```

步骤2：加载tokenizer，首次加载需要的时间较长 使用 AutoTokenizer.from_pretrained 方法，加载预训练的tokenizer "THUDM/chatglm3-6b" 。trust_remote_code=True 表示信任远程代码。

```python
tokenizer = AutoTokenizer.from_pretrained("THUDM/chatglm3-6b",
trust_remote_code=True)
```

步骤3：加载预训练模型

使用 AutoModel.from_pretrained 方法，加载预训练的模型 "THUDM/chatglm3-6b" 到CUDA设备上。 trust_remote_code=True 表示信任远程代码（如果有）， device='cuda' 表示将模型加载到CUDA设备上以便使用GPU加速

要注意的是，根据显卡显存的不同，需要考虑加载不同精度的模型。13GB显存以上的显卡可以直接按照上述代码加载全精度的模型。

```python
model = AutoModel.from_pretrained("THUDM/chatglm3-6b", trust_remote_code=True,
device='cuda')
```

02、ChatGLM API使用

量化版本模型加载

```python
model = AutoModel.from_pretrained("THUDM/chatglm3-6b",trust_remote_code=True).quantize(8).cuda()
```

```python
model = AutoModel.from_pretrained("THUDM/chatglm3-6b",trust_remote_code=True).quantize(4).cuda()
```

多显卡方式

如果有多张 GPU，但是每张 GPU 的显存大小都不足以容纳完整的模型，那么可以将模型切分在多张GPU上。首先安装 accelerate: pip install accelerate，然后通过如下方法加载模型：

```python
from utils import load_model_on_gpus
model = load_model_on_gpus("THUDM/chatglm3-6b", num_gpus=2)
```

03、ChatGLM API使用

步骤4：实例化模型
接下来则需要对模型进行实例化操作，并且设置为评估模式：

markdown

```python
model = model.eval()
```

Python

步骤5：调用模型并获取结果

```python
response, history = model.chat(tokenizer, "你好", history=[])
print(response)
```

Python

```
你好👋！我是人工智能助手 ChatGLM3-6B，很高兴见到你，欢迎问我任何问题。
```

```python
print(history)
```

Python

```
[{'role': 'user', 'content': '你好'}, {'role': 'assistant', 'metadata': '', 'content': '你好👋！我是人工智能助手 ChatGLM3-6B，很高兴见到你，欢迎问我任
```

2 OpenAI风格API调用

01、OpenAI风格API调用

步骤一：启动OpenAI服务

```
root@autodl-container-794d4f961c-59cb97cd:~/glm3/ChatGLM3/openai_api_demo# pwd
/root/glm3/ChatGLM3/openai_api_demo
root@autodl-container-794d4f961c-59cb97cd:~/glm3/ChatGLM3/openai_api_demo# ll
total 32
drwxr-xr-x  3 root root  139 Nov 18 10:06 ./
drwxr-xr-x 11 root root 4096 Nov 18 09:50 ../
drwxr-xr-x  2 root root   42 Nov 18 10:06 __pycache__/
-rw-r--r--  1 root root 7959 Nov 18 09:49 openai_api.py
-rw-r--r--  1 root root 2446 Nov 18 09:49 openai_api_request.py
-rw-r--r--  1 root root   29 Nov 18 09:49 requirements.txt
-rw-r--r--  1 root root 8345 Nov 18 09:49 utils.py
root@autodl-container-794d4f961c-59cb97cd:~/glm3/ChatGLM3/openai_api_demo# python openai_api.py
Loading checkpoint shards: 100%|██████████████████████████████| 7/7 [00:10<00:00,  1.46s/it]
INFO:     Started server process [1607]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

02、OpenAI风格API调用

步骤二：调用模型代码编写

```python
## OpenAI风格代码调用，首先启动 python openai_api.py

# 使用curl命令测试返回
# curl -X POST "http://127.0.0.1:8000/v1/chat/completions" \
# -H "Content-Type: application/json" \
# -d "{\"model\": \"chatglm3-6b\", \"messages\": [{\"role\": \"system\", \"content\": \"You are ChatGLM3, a large language model trained by Zh

# 使用Python代码测返回
import requests
import json

base_url = "http://127.0.0.1:8000" # 本地部署的地址,或者使用你访问模型的API地址

def create_chat_completion(model, messages):
    data = {
        "model": model, # 模型名称
        "messages": messages, # 会话历史
        "stream": use_stream, # 是否流式响应
        "max_tokens": 100, # 最多生成字数
        "temperature": 0.8, # 温度
        "top_p": 0.8, # 采样概率
    }

    response = requests.post(f"{base_url}/v1/chat/completions", json=data)
    decoded_line = response.json()
    content = decoded_line.get("choices", [{}])[0].get("message", "").get("content", "")
    return content
```

Python

03、OpenAI风格API调用

步骤三：模型调用

```python
chat_messages = [
    {
        "role": "system",
        "content": "You are ChatGLM3, a large language model trained by Zhipu.AI. Follow the user's instructions carefully. Respond using
    },
    {
        "role": "user",
        "content": "你好，给我讲一个故事，大概100字"
    }
]
create_chat_completion("chatglm3-6b", chat_messages)
```

Python

3 ChatGLM3 Function Calling之天气查询

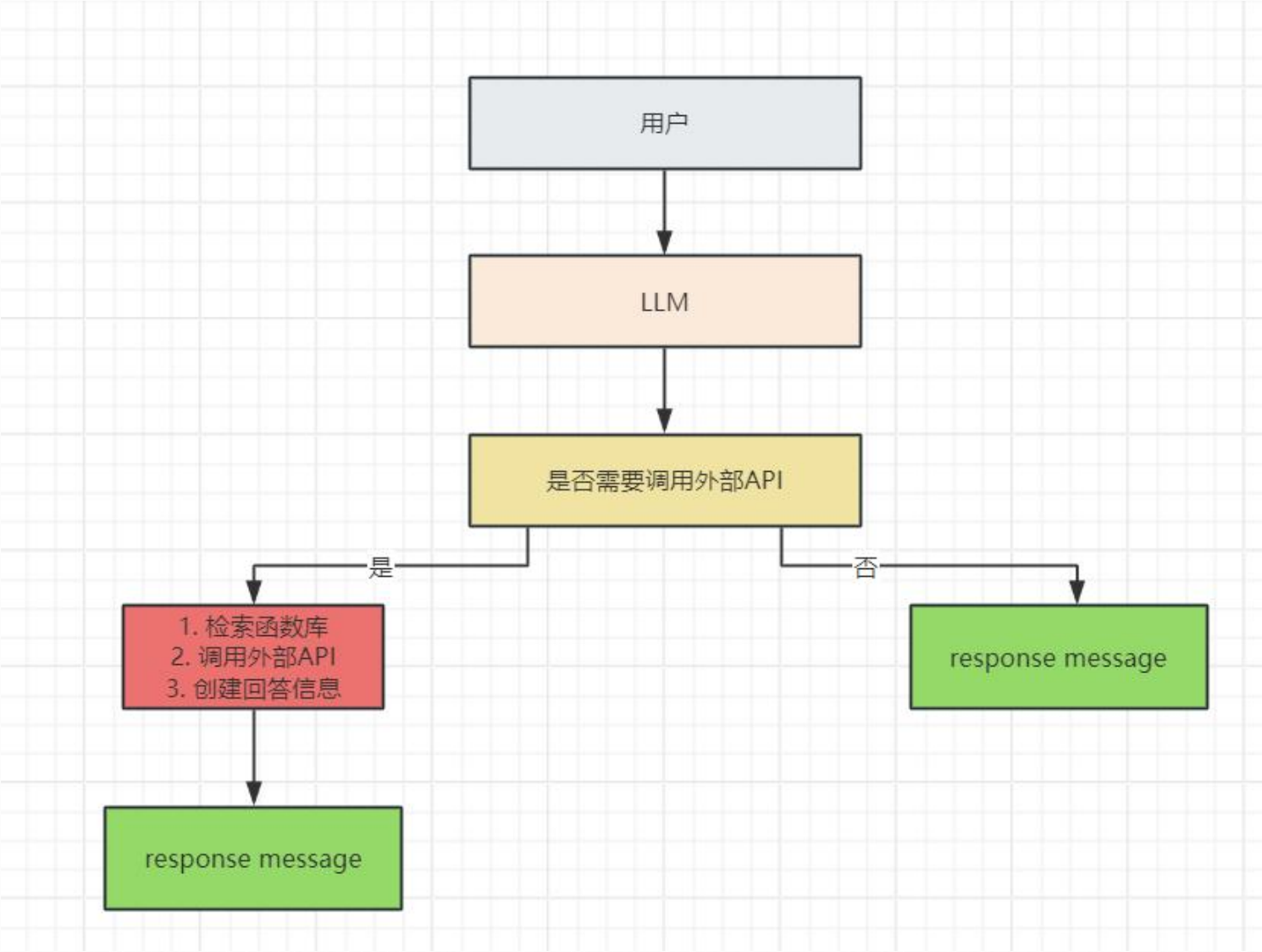
01、ChatGLM3史诗级功能

ChatGLM3 是智谱AI和清华大学 KEG 实验室联合发布的新一代对话预训练模型。ChatGLM3-6B 是 ChatGLM3 系列中的开源模型，在保留了前两代模型对话流畅、部署门槛低等众多优秀特性的基础上，ChatGLM3-6B 引入了如下特性：

1. **更强大的基础模型：** ChatGLM3-6B 的基础模型 ChatGLM3-6B-Base 采用了更多样的训练数据、更充分的训练步数和更合理的训练策略。在语义、数学、推理、代码、知识等不同角度的数据集上测评显示，**ChatGLM3-6B-Base 具有在 10B 以下的基础模型中最强的性能。**
2. **更完整的功能支持：** ChatGLM3-6B 采用了全新设计的 Prompt 格式，除正常的多轮对话外。同时原生支持工具调用（Function Call）、代码执行（Code Interpreter）和 Agent 任务等复杂场景。
3. **更全面的开源序列：** 除了对话模型 ChatGLM3-6B 外，还开源了基础模型 ChatGLM3-6B-Base、长文本对话模型 ChatGLM3-6B-32K。以上所有权重对学术研究**完全开放**，在填写问卷进行登记后**亦允许免费商业使用。**

正式接轨AI Agent开发

02、Function Calling流程



03、获取天气的API接口

https://openweathermap.org/

```python
import json
import requests
def get_weather(loc):
    """
    查询即时天气函数
    :param loc: 必要参数，字符串类型，用于表示查询天气的具体城市名称，\
    注意，中国的城市需要用对应城市的英文名称代替，例如如果需要查询北京市天气，则loc参数需要输入'Beijing'；
    :return: OpenWeather API查询即时天气的结果，具体URL请求地址为: https://api.openweathermap.org/data/2.5/weather\
    返回结果对象类型为解析之后的JSON格式对象，并用字符串形式进行表示，其中包含了全部重要的天气信息
    """
    # Step 1.构建请求
    url = "https://api.openweathermap.org/data/2.5/weather"

    # Step 2.设置查询参数
    params = {
        "q": loc,
        "appid": open_weather_key,    # 输入API key
        "units": "metric",            # 使用摄氏度而不是华氏度
        "lang":"zh_cn"                # 输出语言为简体中文
    }

    # Step 3.发送GET请求
    response = requests.get(url, params=params)

    # Step 4.解析响应
    data = response.json()
    return json.dumps(data)
```

Python

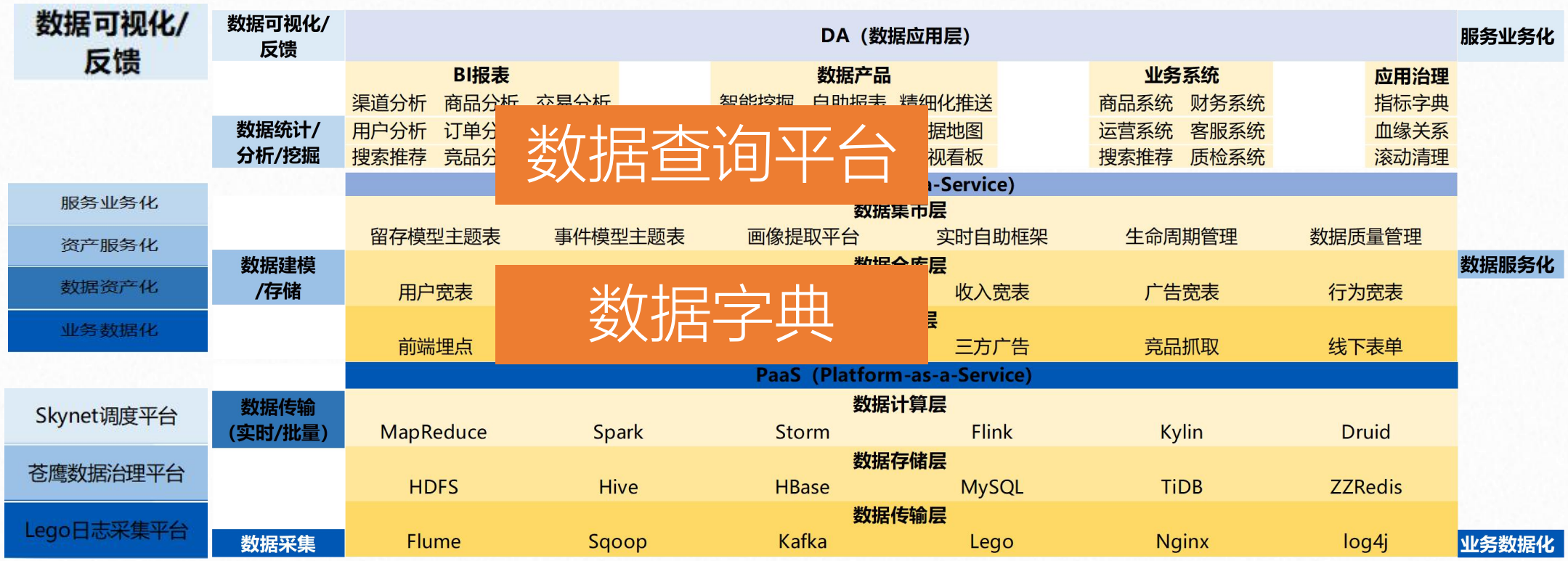
04、代码落地

代码落地

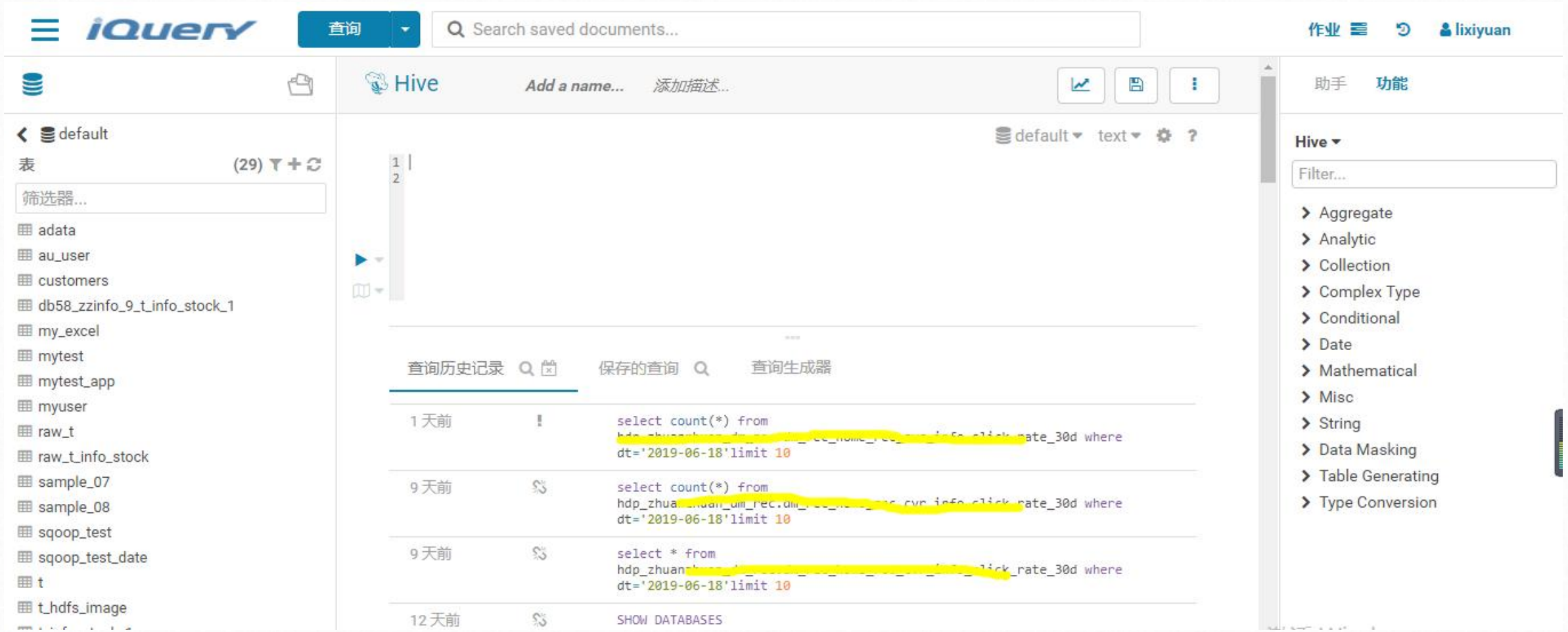
4 自助大数据查询平台架构设计与落地

01、万亿级电商大数据平台

- 总体中台架构设计与实践
  - 分层架构设计体系



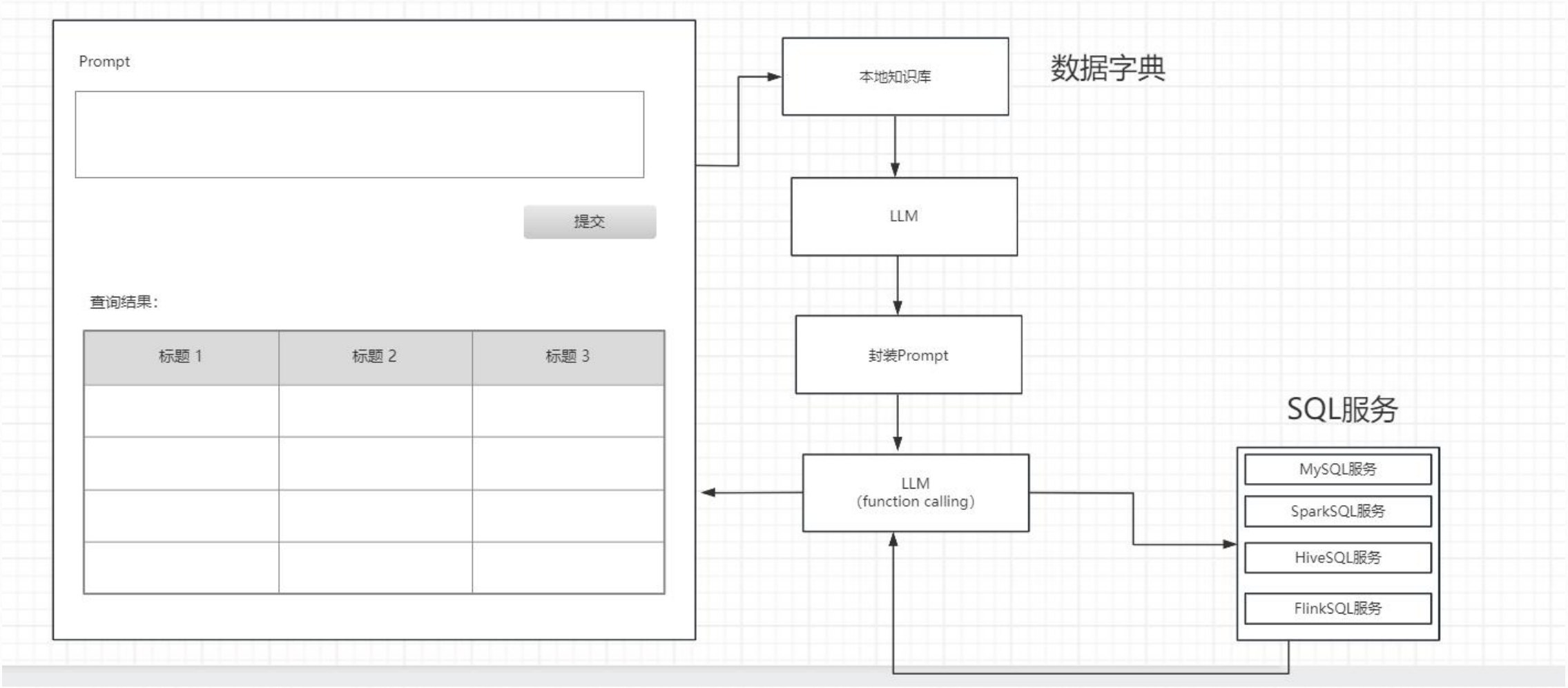
02、iQuery自助查询平台



03、数据字典

序号	列名	数据类型	长度	小数位	标识	主键	允许空	默认值	说明
1	UserID	int	4	0	√	√			用户标识
2	GameID	int	4	0				0	游戏标识
3	ProtectID	int	4	0				0	密保标识
4	SpreaderID	Int	4	0				0	推广员标识
5	Accounts	nvarchar	62	0					用户帐号
6	RegAccounts	nvarchar	62	0					注册帐号
7	UnderWrite	nvarchar	126	0				('')	个性签名
8	LogonPass	nchar	64	0					登录密码
9	InsurePass	nchar	64	0					安全密码
10	EMail	nvarchar	62	0				(N'')	电子邮件
11	FaceID	smallint	2	0				0	头像标识
12	Experience	int	4	0				0	经验数值
13	UserRight	int	4	0				0	用户权限
14	MasterRight	int	4	0				0	管理权限
15	ServiceRight	int	4	0				0	服务权限
16	MasterOrder	tinyint	1	0				0	管理等级
17	MemberOrder	tinyint	1	0				0	会员等级
18	MemberOverDate	datetime	8	3				1980-1-1	过期日期
19	Gender	tinyint	1	0				0	用户性别
20	Nullity	bit	1	0				0	禁止服务
21	StunDown	bit	1	0				0	关闭标志
22	MoorMachine	tinyint	1	0				0	固定机器
23	MachineSerial	nchar	64	0				('')	机器序列
24	WebLogonTimes	int	4	0				0	登录次数
25	GameLogonTimes	int	4	0				0	登录次数
26	RegisterIP	nvarchar	30	0					注册地址
27	LastLogonIP	nvarchar	30	0					登录地址
28	RegisterDate	datetime	8	3				getdate()	注册时间

04、方案设计


Prompt
提交
查询结果：
标题 1
标题 2
标题 3
本地知识库
数据字典
LLM
封装Prompt
LLM
(function calling)
SQL服务
MySQL服务
SparkSQL服务
HiveSQL服务
FlinkSQL服务

05、数据字典格式

擅长讲输出转化为不同格式，比如从一种语言翻译成另一种语言，帮助拼写、语法纠正以及编写正则表达式

1. user_info数据表

基本解释

user_info数据表记录了电信用户的个人基本情况，主要涵盖客户基本生物属性，包括性别、年龄状况、是否结婚以及是否经济独立等。

数据来源

`user_info`数据集由一线业务人员人工采集记录，并且通过回访确认相关信息，数据集的准确性和可信度都非常高。

各字段说明

列名	描述	取值范围	值解释	类型
customerID	客户ID，user_info数据表主键		由数字和字母组成的	VARCHAR(255)
gender	用户的性别	Female, Male	Female (女性), Male (男性)	VARCHAR(255)
SeniorCitizen	是否为老人	0, 1	0 (不是), 1 (是)	INT
Partner	用户是否有伴侣	Yes, No	Yes (有), No (没有)	VARCHAR(255)
Dependents	用户经济是否独立，往往用于判断用户是否已经成年	No, Yes	Yes (有), No (没有)	VARCHAR(255)

06、SQLServer服务

MySQL服务访问

```python
import mysql.connector

# 创建数据库连接
cnx = mysql.connector.connect(user='your_username', password='your_password',
                              host='localhost', database='your_database')

# 创建游标对象
cursor = cnx.cursor()

# 执行查询语句
query = "SELECT * FROM your_table"
cursor.execute(query)

# 获取查询结果
results = cursor.fetchall()

# 处理结果
for row in results:
    # 处理每一行数据

# 关闭游标和连接
cursor.close()
cnx.close()
```

HiveServer服务访问

```python
from pyhive import hive

# 创建Hive连接
conn = hive.Connection(host="your_hostname", port=10000, username="your_username")

# 创建游标对象
cursor = conn.cursor()

# 执行查询语句
query = "SELECT * FROM your_table"
cursor.execute(query)

# 获取查询结果
results = cursor.fetchall()

# 处理结果
for row in results:
    # 处理每一行数据

# 关闭游标和连接
cursor.close()
conn.close()
```

07、SQLServer服务

SparkSQL服务访问

```python
from pyspark.sql import SparkSession

# 创建SparkSession对象
spark = SparkSession.builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

# 执行SQL查询
query = "SELECT * FROM your_table"
results = spark.sql(query)

# 处理结果
for row in results.collect():
    # 处理每一行数据

# 关闭SparkSession
spark.stop()
```

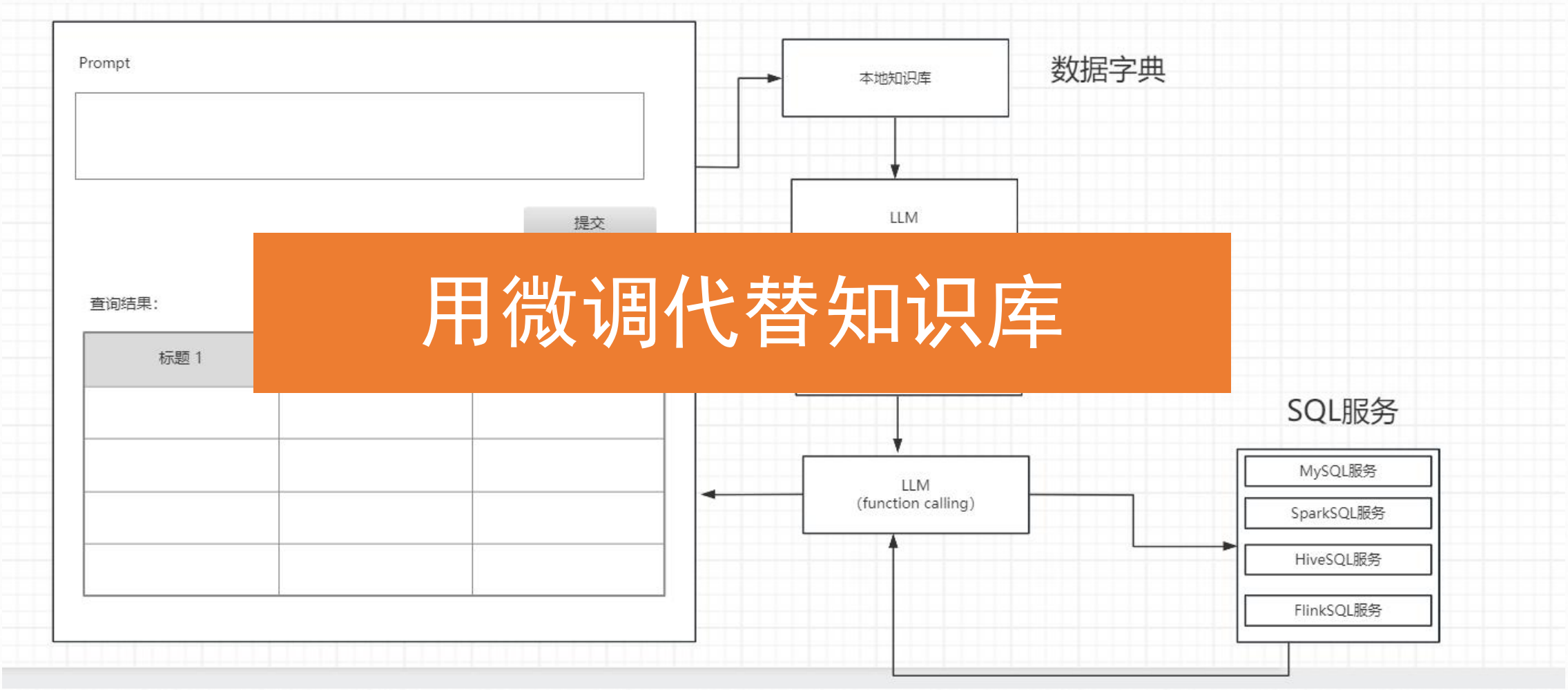
FlinkSQL服务访问

```python
from pyflink.table import EnvironmentSettings, TableEnvironment

# 设置FlinkSQL的执行环境
env_settings = EnvironmentSettings.new_instance().in_streaming_mode().use_blink_planner().build()
table_env = TableEnvironment.create(environment_settings=env_settings)

# 执行SQL查询
query = "SELECT * FROM your_table"
results = table_env.execute_sql(query)

# 将结果转换为Pandas DataFrame（可选）
df = results.to_pandas()
print(df)

# 关闭TableEnvironment
table_env.close()
```

08、代码落地

代码落地

09、作业思考题



用微调代替知识库

关注视频号：玄姐谈AGI

助力数字化人才提升 AIGC 能力