AI 大模型架构师
之面向目标架构案例实践

讲师：李希沅

目录

1 面向目标架构案例业务背景

01、大模型 API 开发范式

面向对象开发

```
[ ]: from openai import OpenAI
     openai.api_key = os.getenv("OPENAI_API_KEY")
     client = OpenAI(api_key=openai.api_key)
```

```
[ ]: messages=[
         {"role": "system", "content": "你是一个乐于助人的智能AI小助手"},
         {"role": "user", "content": "你好，请你介绍一下你自己"}
       ]
```

1. 构建Prompt

```
[ ]: completion = client.chat.completions.create(
       model="gpt-3.5-turbo",
       messages= messages)
```

2. 模型的调用

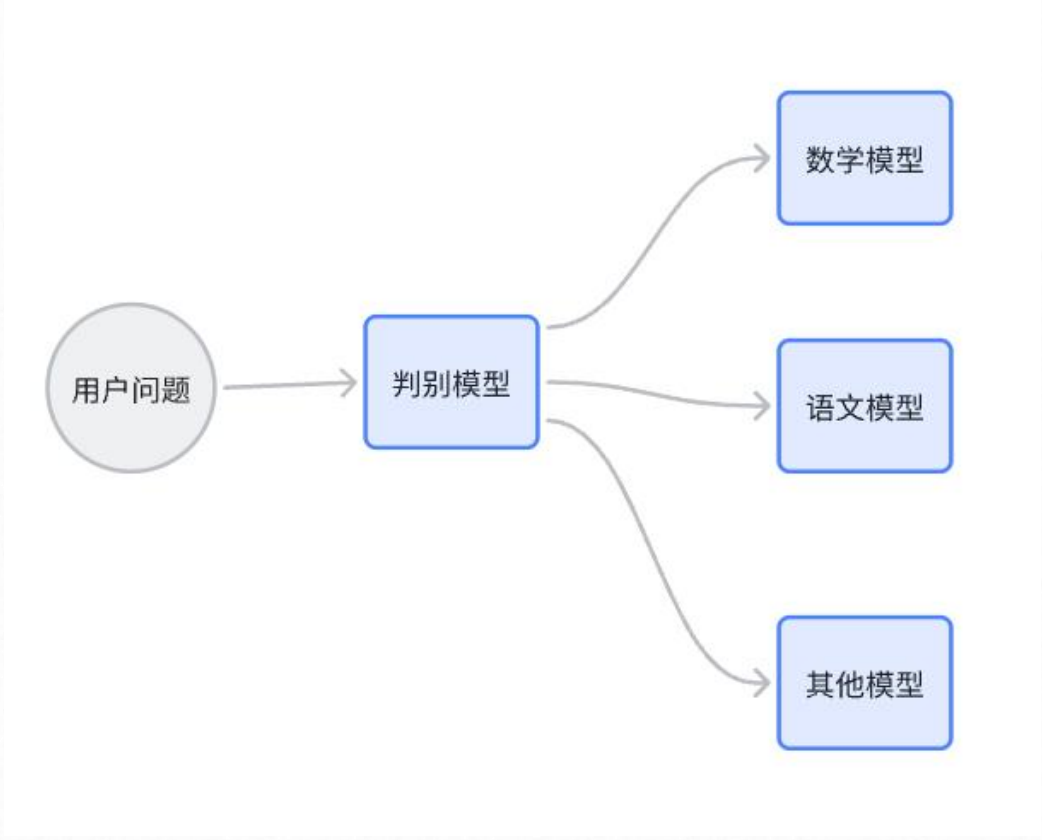
```
[ ]: print(completion.choices[0].message.content)
```

3. 执行结果的解析

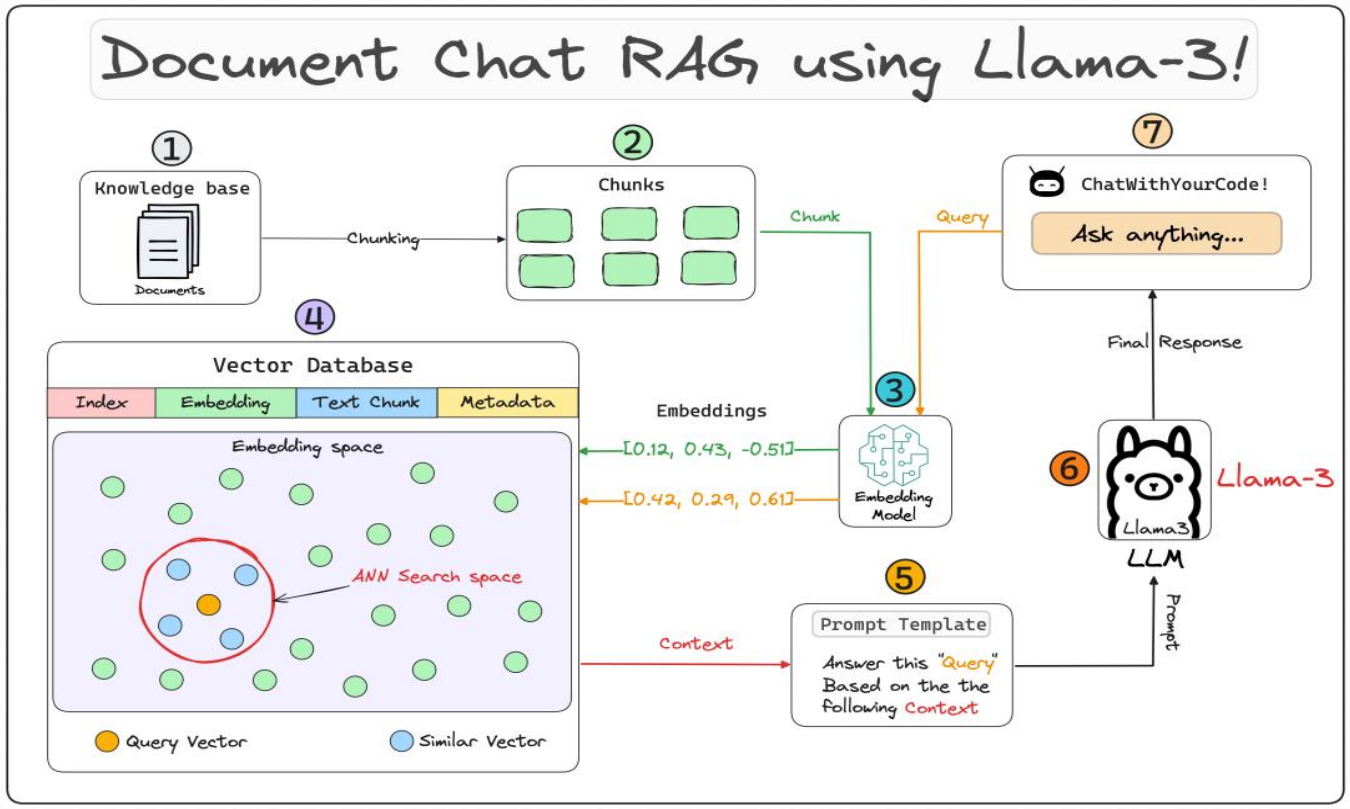
对话机器人

02、大模型 API 开发范式

面向过程开发



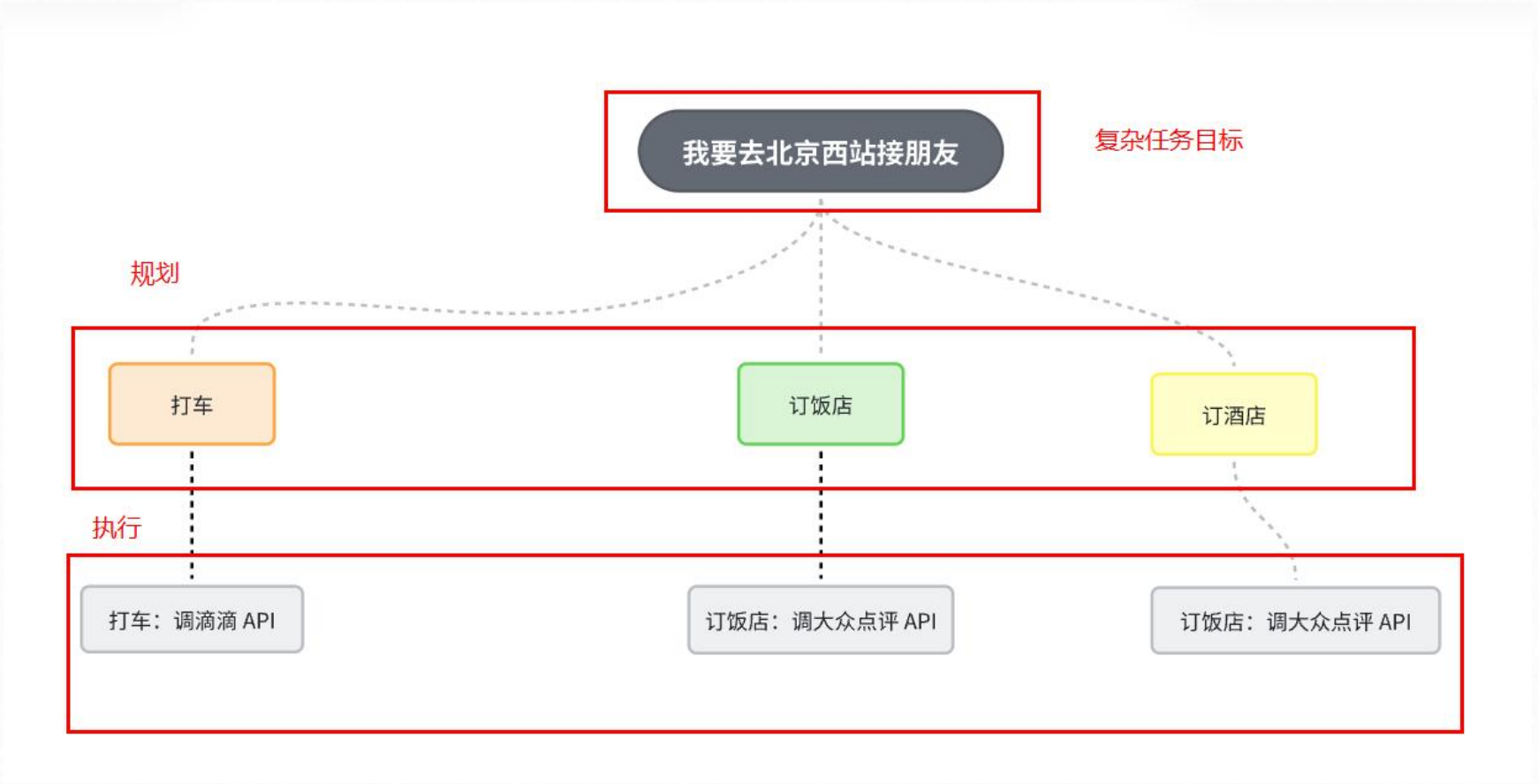
Chain



RAG（知识库）

03、大模型 API 开发范式

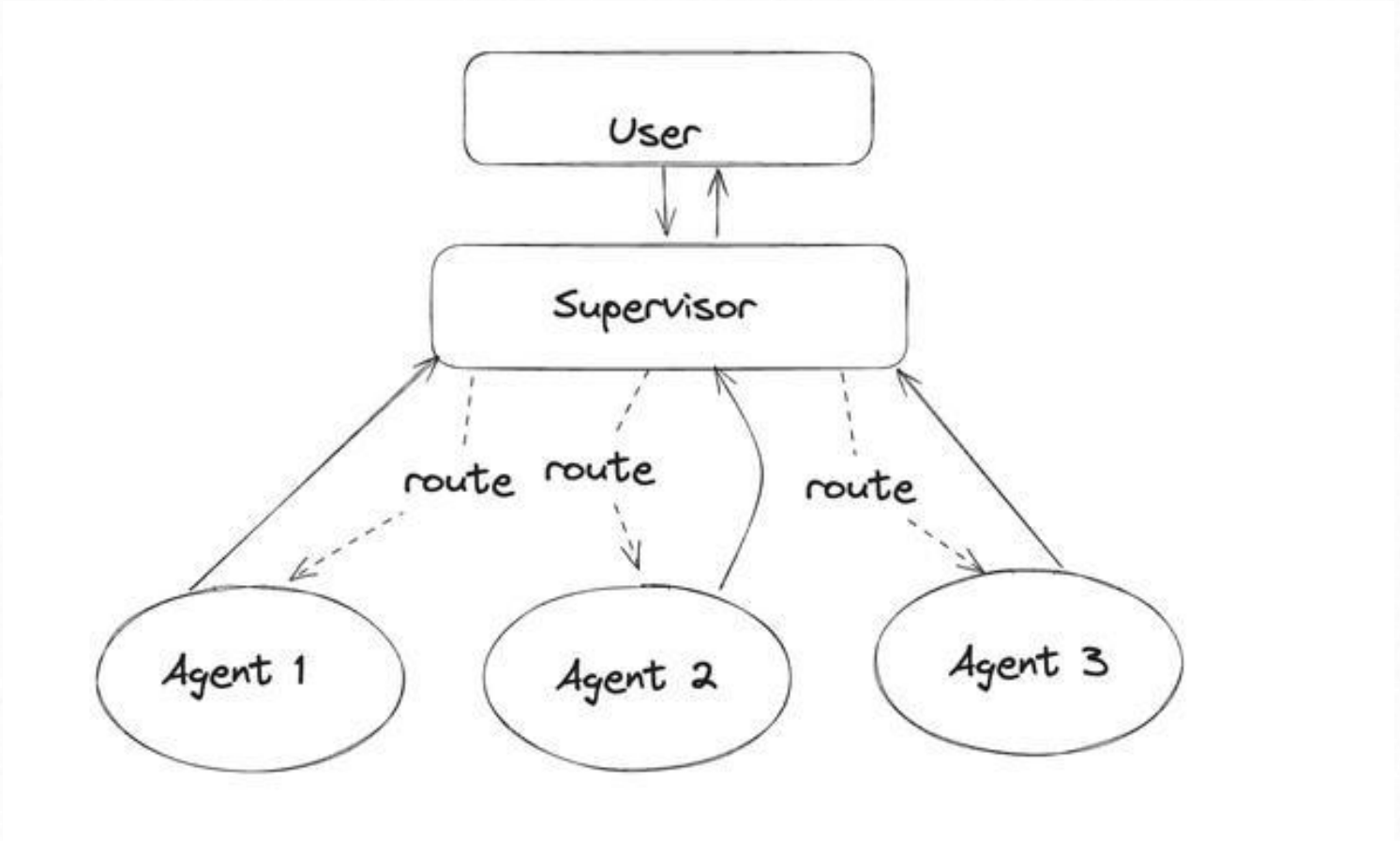
面向 Agent 开发 → 面向目标开发



智能差旅

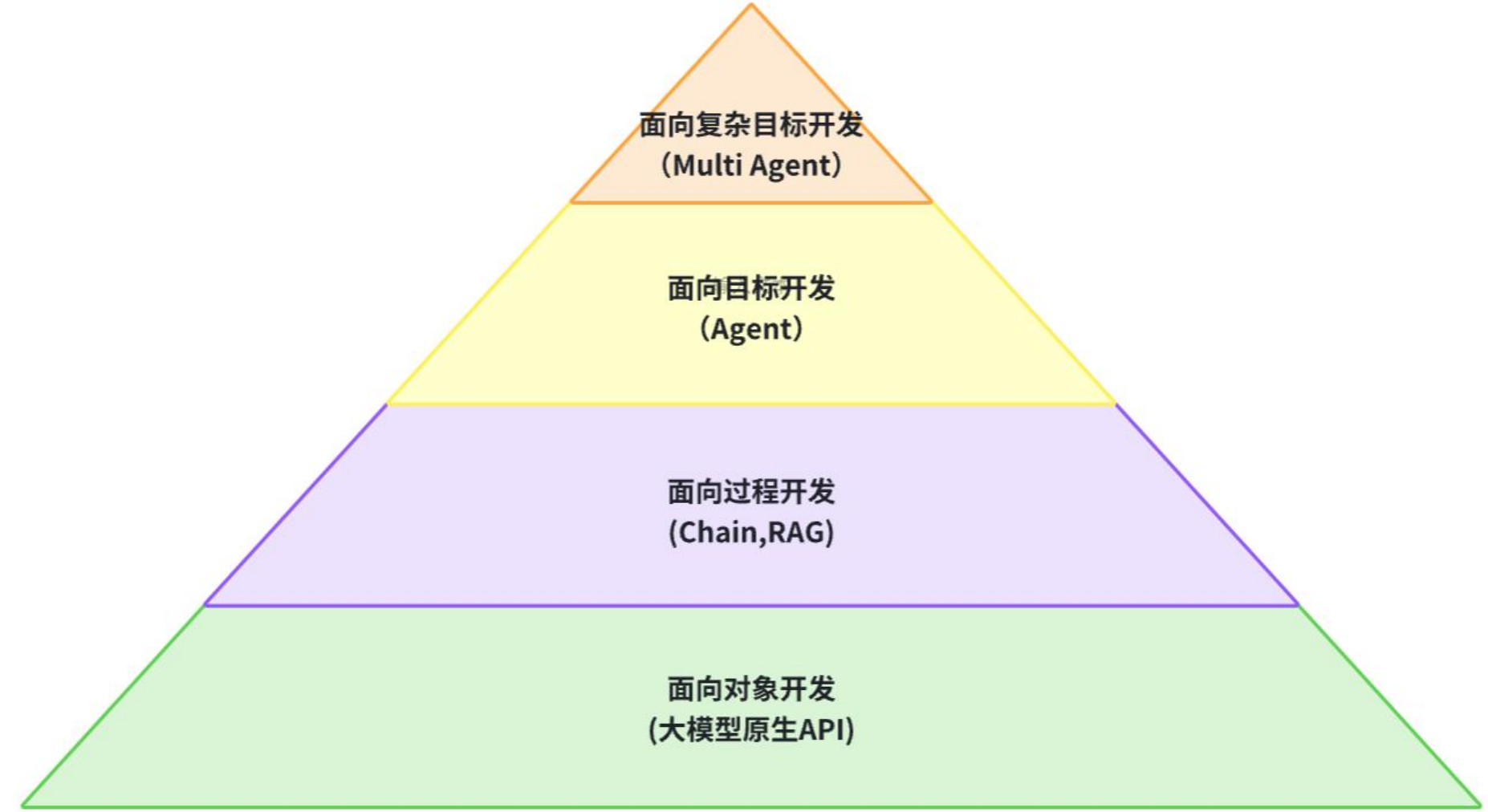
04、大模型 API 开发范式

面向多Agent开发 → 面向复杂目标开发



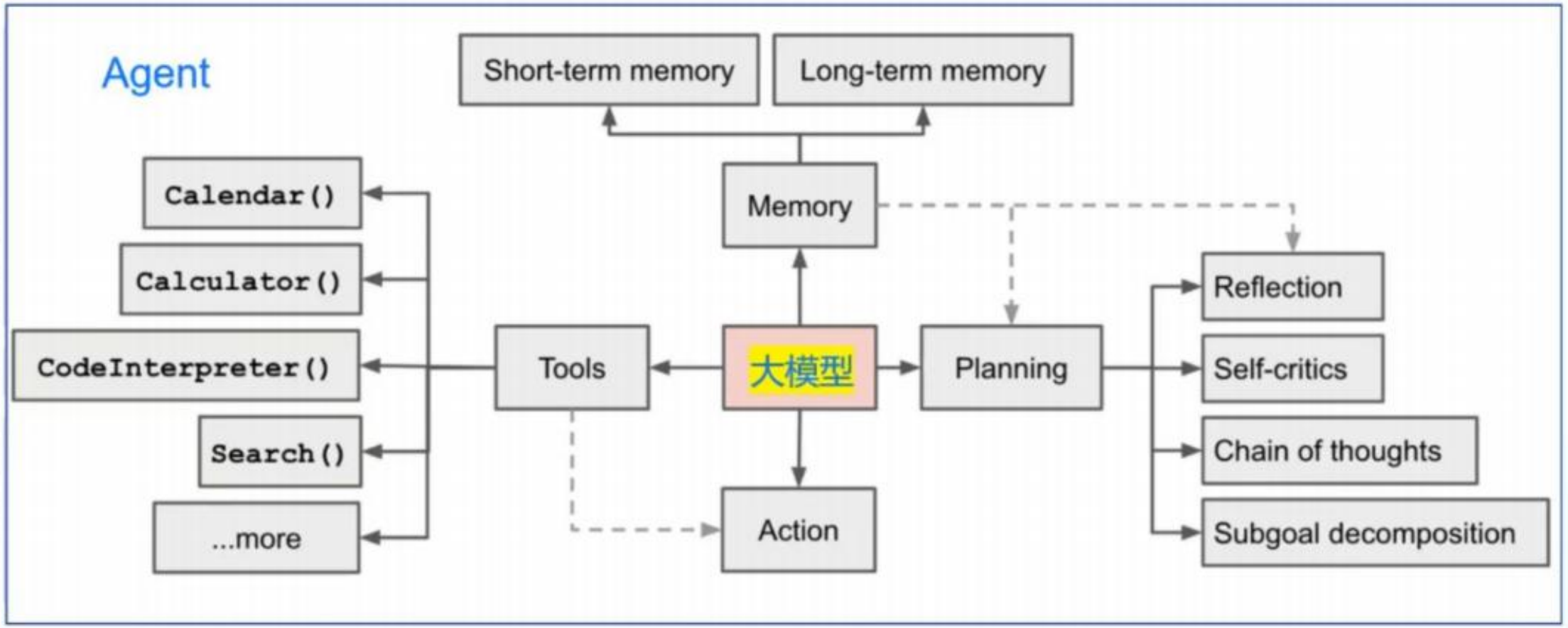
低代码开发平台

05、大模型 API 开发范式



06、Agent 架构设计范式

Agent 的设计理念



07、Agent 架构设计范式

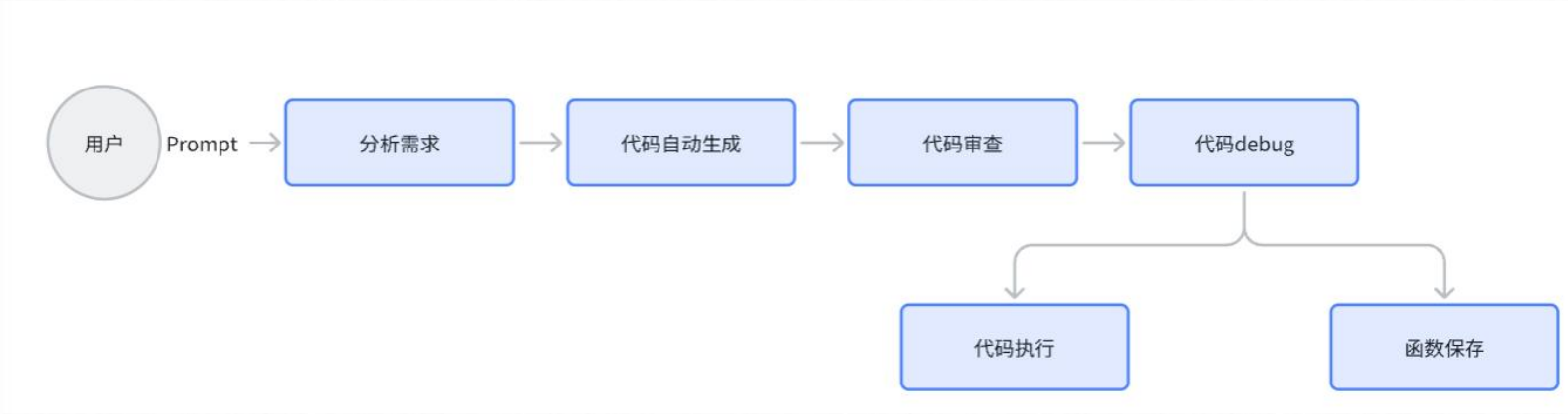
重要结论

结论1：Agent的思路是面向目标开发的思路，但是不是说面向目标就是Agent开发。

结论2：面向目标核心指的是：1. 大模型能自动帮我们规划流程（实现业务逻辑） 2. 自动的调用外部工具函数

结论3：我们这个项目想要探索的方向是：1. 大模型能自动帮我们规划流程（实现业务逻辑）2. 自动的调用外部工具函数 3. 自动生成外部函数。

08、面向目标架构案例业务背景



09、项目目标和意义

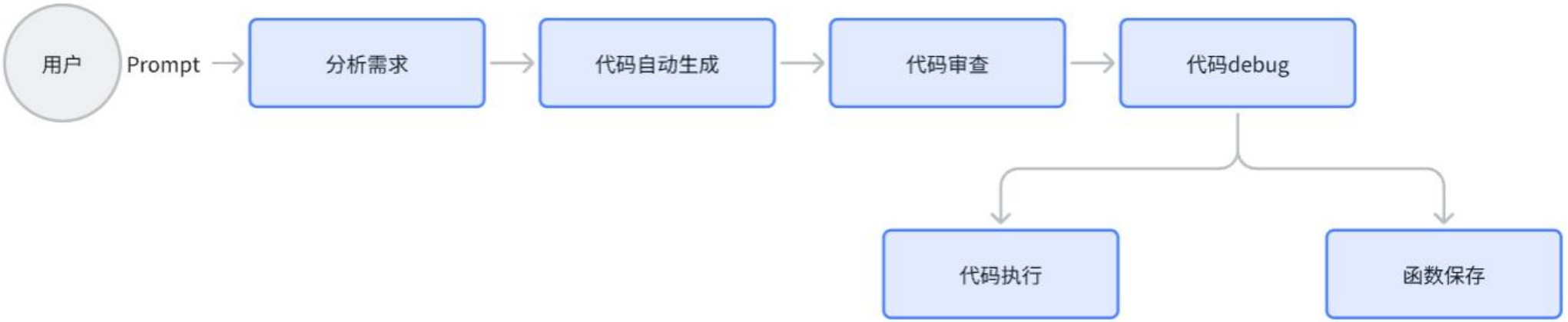
面向目标开发是个趋势，我们先探索

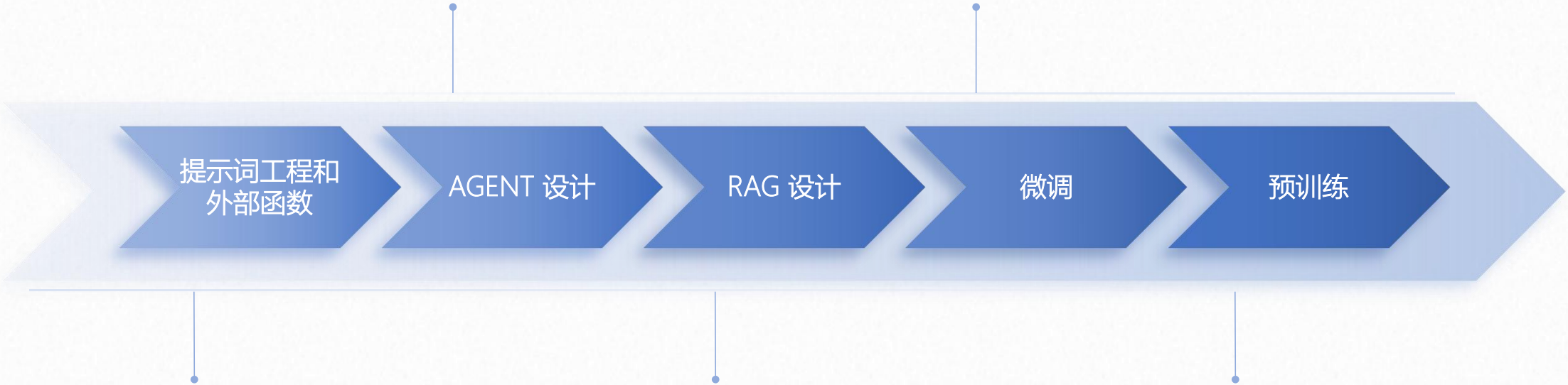
深度掌握 Function Call 使用

深度掌握提示词工程技巧

❷ 面向目标架构案例技术选型

01、面向目标架构案例技术选型



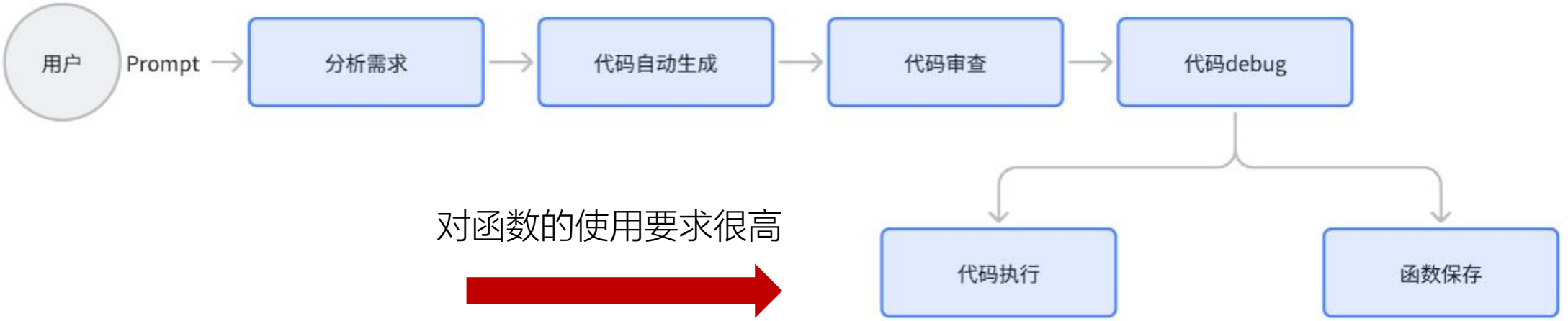


02、面向目标架构案例技术选型

技术	选择理由
提示词工程	是我们核心掌握的能力
Function Call	是我们核心掌握的能力
GLM-4模型	1. GLM-4的能力是中上能力的模型 2. 注册使用方便
开发框架	意义不大/LangChain

3 面向目标架构案例之Function Call进阶

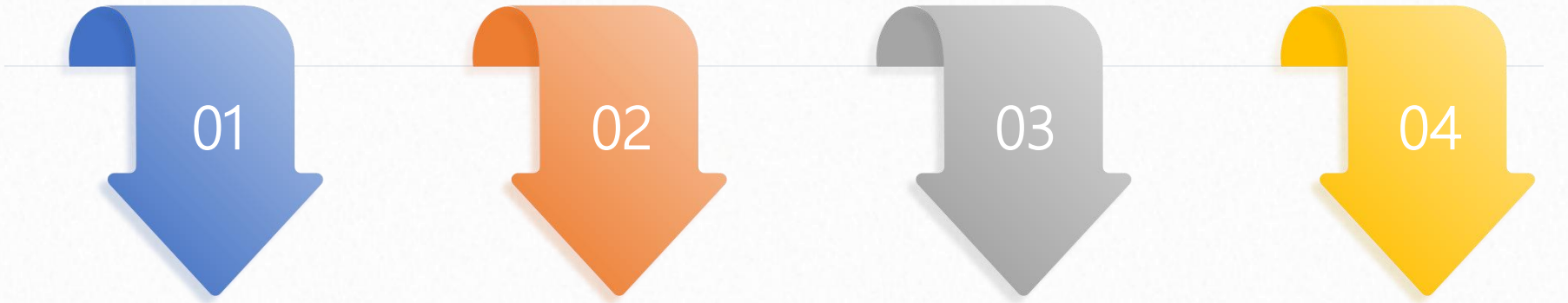
00、为什么要深度掌握 Function Call 技巧



所有方案都有可能需要Function Call

01、Function Call实战中的四大挑战

四大常见挑战



意图识别

如何精准识别出应该调用哪个函数?

海量函数

超多外部函数绑定，超过了上下文长度限制怎么办?

并发调用

一个Prompt调用不止一个函数如何解决?

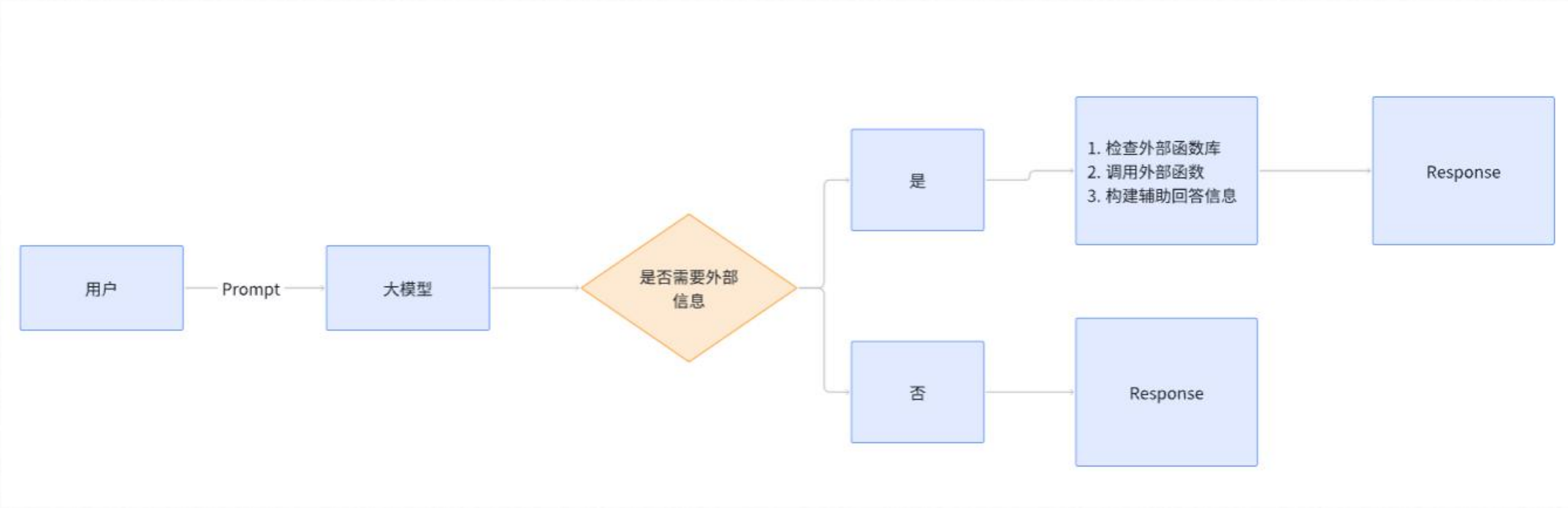
响应太慢

流程消耗时间太长怎么办?

这是常见的4大挑战，但不限于此

02、Function Call实战中的四大挑战

意图识别



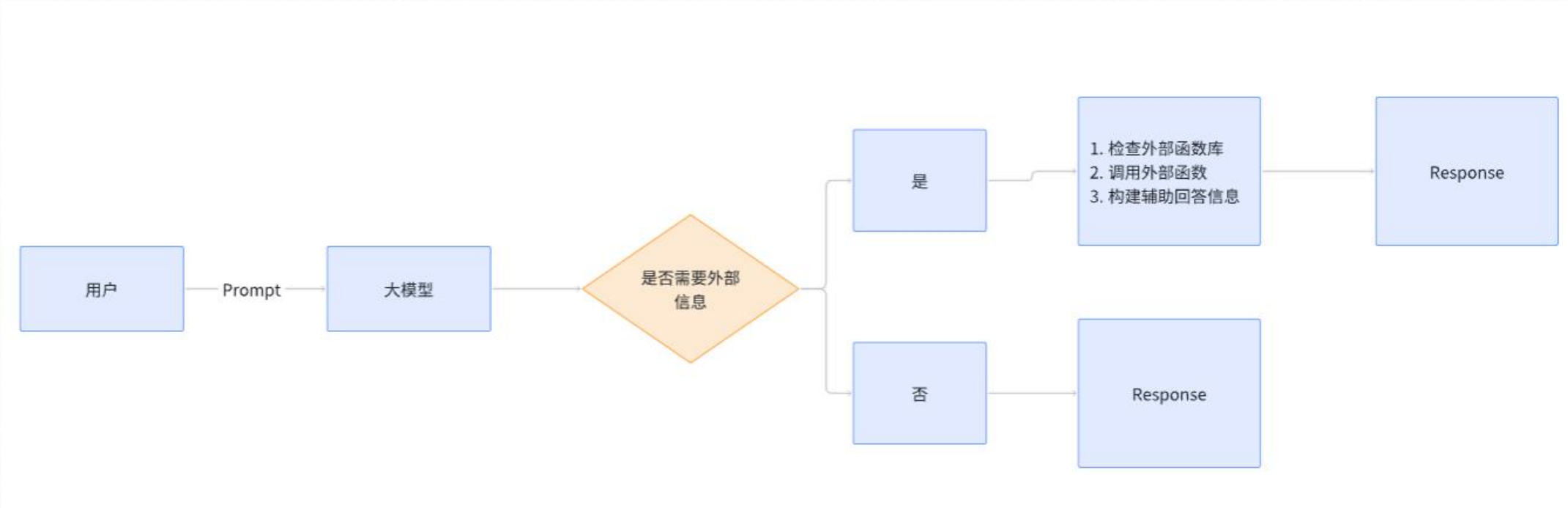
1. 判断是否需要外部函数帮助时，可能出现误判

2. 选择外部函数时，相似的函数名称可能导致混淆，导致误选

3. 无法正确地整合外部函数的结果，即便获得了正确答案，依然无法完整整合信息

03、Function Call实战中的四大挑战

海量函数

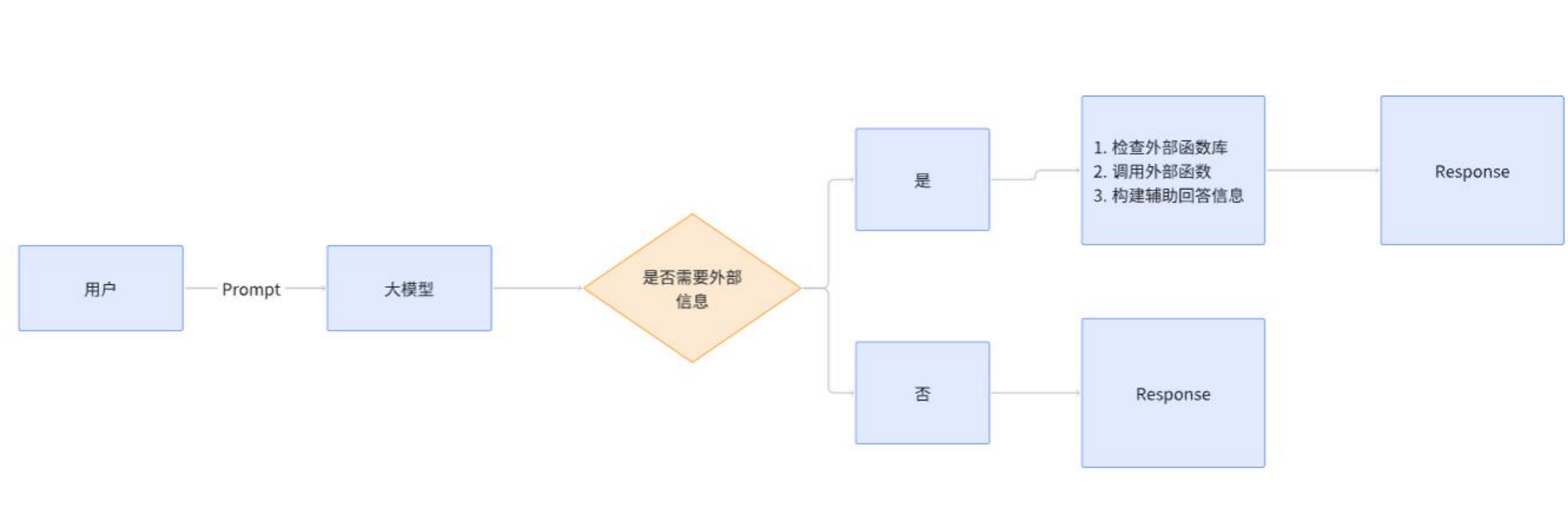


1. 在function call使用过程中，我们输入给模型的所有信息（包括但不限于prompt、函数的JSON Schema、设置的system背景信息、全部的上下文等等）都会被记录在Token消耗当中。

2. 函数越多，大模型精准识别的能力就越弱

04、Function Call实战中的四大挑战

并发调用

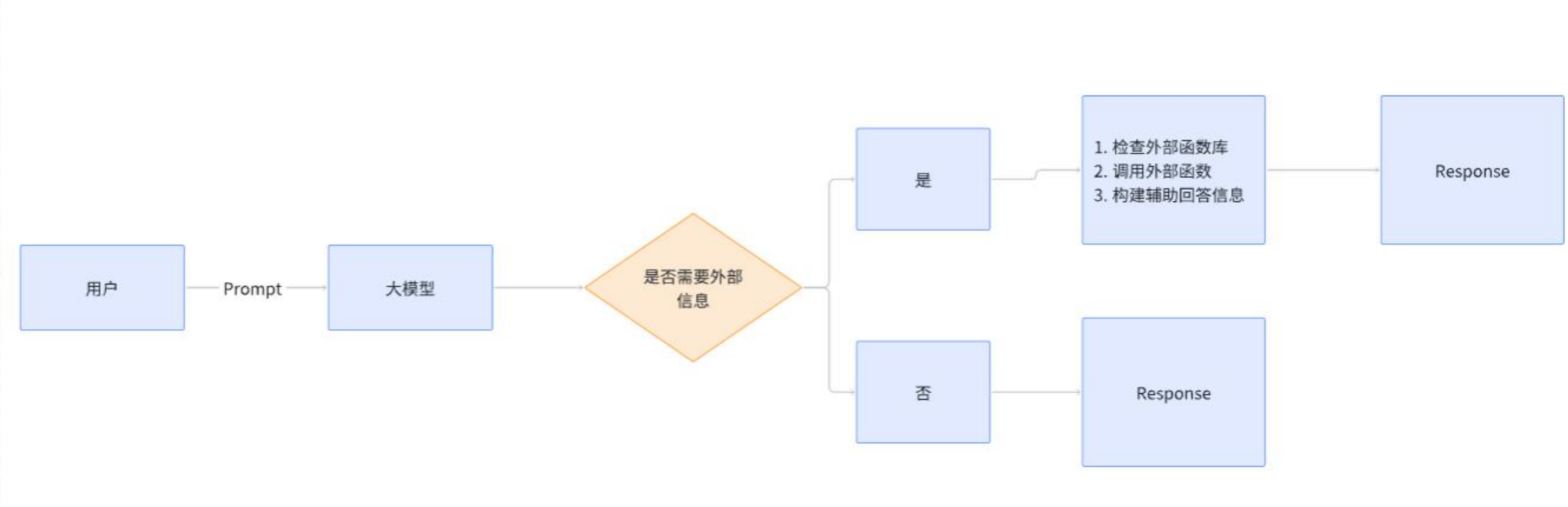


1. 一个Prompt里面需要多个外部函数串行执行。

2. 一个Prompt里面需要多个外部函数并行执行。

05、Function Call实战中的四大挑战

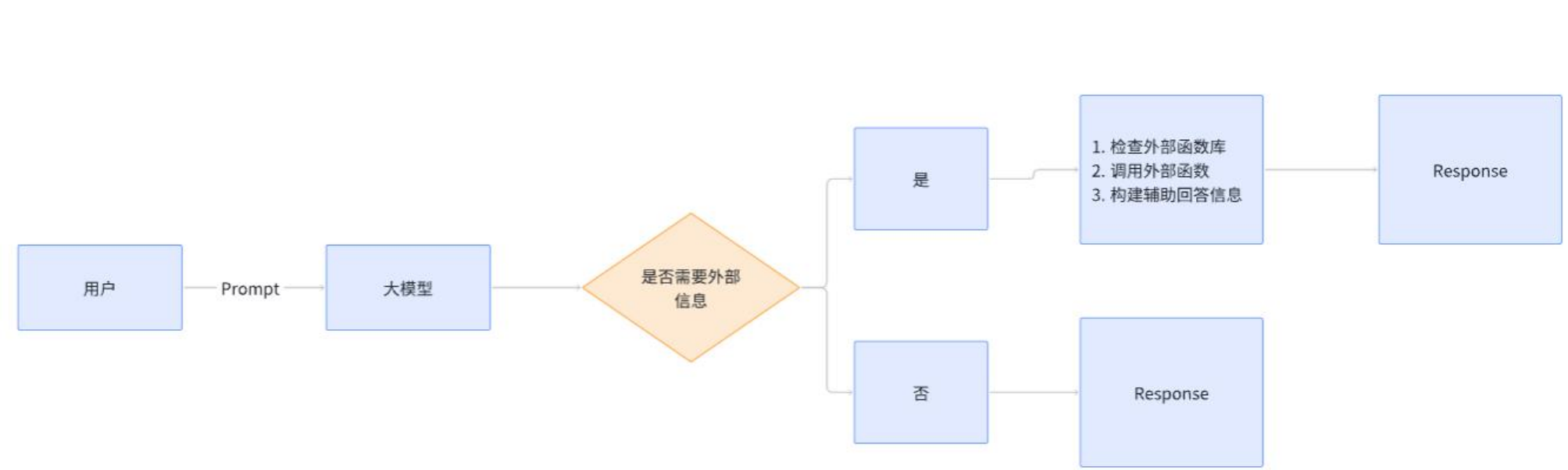
响应太慢



1. 整个外部函数调用的流程，比较冗长，响应速度太慢怎么办？

06、Function Call实战进阶技巧

第一个挑战：意图识别



1. 判断是否需要外部函数帮助时，可能出现误判

2. 选择外部函数时，相似的函数名称可能导致混淆，导致误选

3. 无法正确地整合外部函数的结果，即便获得了正确答案，依然无法完整整合信息

07、Function Call实战进阶技巧

```python
def machine_learning_1():
    """
    解释什么是机器学习
    """

    answer = """机器学习是人工智能的一个分支，它使计算机能够从数据中学习并做出决策，\
    而无需明确编程。通过分析大量数据，机器学习算法可以识别数据模式和特征，用于预测或决策。\
    这种方法包括监督学习、无监督学习和强化学习等类型，广泛应用于分类、回归、聚类和推荐系统等领域，\
    旨在提高工作效率和决策质量。"""

    return answer

def machine_learning_2():

    """机器学习和深度学习的区别"""

    answer = """机器学习是让计算机从数据中学习的一般方法，包括多种算法，如线性回归、决策树等，\
    通常需要人工进行特征提取。深度学习是机器学习的一种特殊形式，使用多层神经网络，能够自动从数据\
    中学习更复杂的特征表示，适用于处理大规模、复杂数据，尤其在图像识别、语音识别等领域表现优越。"""

    return answer
```

Prompt	测试结果
测试：什么是机器学习?	调用了machine_learning_1函数
测试：请问深度学习和机器学习的区别?	调用了machine_learning_2函数
测试：请问什么是深度学习?	调用了machine_learning_1函数

08、Function Call实战进阶技巧

提升JSON Schame的质量

方式	具体操作
关键字	Prompt的提示词与函数名的关系很大 大模型是对Prompt提示词和函数名进行Embedding，然后通过相似度来匹配函数的。
特定词语对函数分类	在函数名称前添加前缀：user_，test_，admin_
给函数增加权重	为使用更频繁的函数增加primary、prior、core等有助于模型理解其重要性的名称，也可以在进行编号或者函数的描述时写上“非常重要，经常使用，核心业务”等等描述。
使用否定句，帮助模型进行函数辨别	当用户的问题中，包括xxx时，该函数不能用。

09、Function Call实战进阶技巧

干预模型选择

使用自查Prompt进行干预

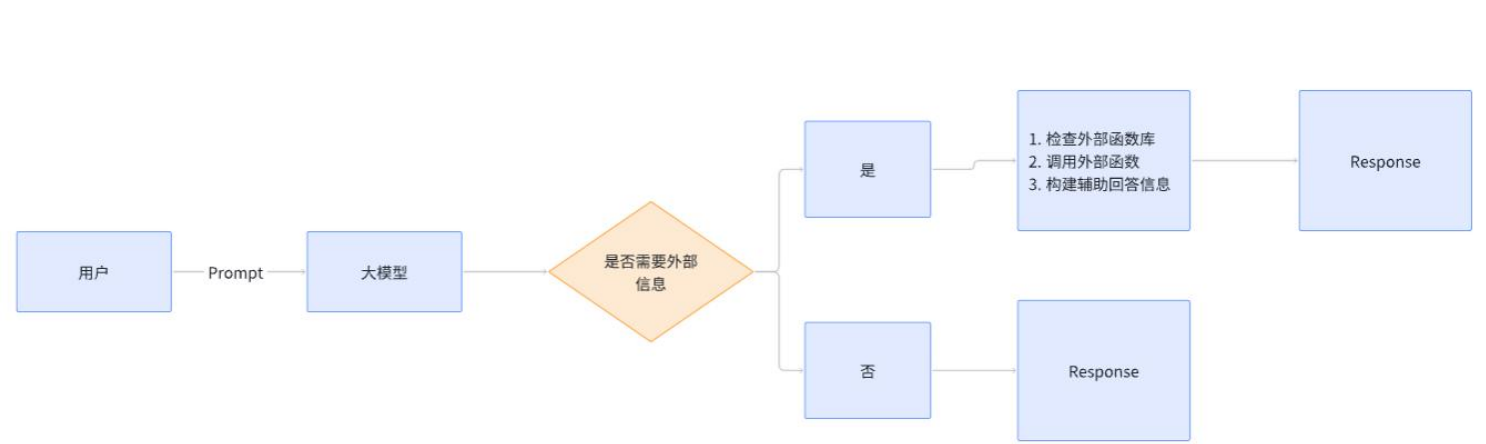
```
def function_call(prompt,tools_):
    prompt = prompt
    judge_words = "这个问题的答案是否在你的语料库里？\
        请回答“这个问题答案在我的语料库里”或者“这个问题的答案不在我的语料库里”\
        不要回答其他额外的文字"
    message = [
        {"role": "user", "content": prompt + judge_words}
    ]

    response1 = client.chat.completions.create(
        model="glm-4",
        messages=message
        )

    if "这个问题答案在我的语料库里" in response1.choices[0].message.content:
        message = [
        {"role": "user", "content": prompt}
    ]
        response2 = client.chat.completions.create(
            model="glm-4",
            messages=message,
        )

        print(response2.choices[0].message.content)

    else:
        message = [
        {"role": "user", "content": prompt}
    ]
        response2 = client.chat.completions.create(
            model="glm-4",  # 填写需要调用的模型名称
            messages=message,
            tools = tools_,
            tool_choice = "auto"
            )
        print(response2.choices[0].message.tool_calls[0].function.name)
```

依赖匹配好于依赖理解

```
def function_call(prompt,tools_):
    prompt = prompt
    judge_words =["我要点菜",
                  "我要下单",
                  "我要吃饭",
                  "我看看菜单"
                  "......"
    ]


    if prompt not in judge_words
        message = [
        {"role": "user", "content": prompt}]

        response1 = client.chat.completions.create(
            model="glm-4",
            messages=message,
        )

        print(response1.choices[0].message.content)

    else:
        message = [
        {"role": "user", "content": prompt}]
        response2 = client.chat.completions.create(
            model="glm-4",  # 填写需要调用的模型名称
            messages=message,
            tools = tools_,
            tool_choice = "auto"
            )
```

10、Funcation Call实战进阶技巧

函数信息精准返回




```python
def multiply(a: float, b: float) -> float:
    """
    计算两个浮点数的乘积。

    参数:
    a (float): 第一个乘数。
    b (float): 第二个乘数。

    返回:
    float: 两个参数的乘积。
    """
    return a * b
```

Prompt	测试结果
第一次	函数正确调用，结果正确。
第二次	函数正确调用，结果正确。
第三次	函数正确调用，结果不正确。
第四次	函数正确调用，结果正确。
第五次	函数正确调用，结果不正确。

很多大模型的Funcation Call的能力还不够强

11、Function Call实战进阶技巧

函数信息精准返回

OpenAI风格

```
message.append({
            "role": "tool",
            "name": response_f.choices[0].message.tool_calls[0].function.name,
            "content": prompt + str(function_response),
            })

response_a = client.chat.completions.create(
    model="glm-4",  # 填写需要调用的模型名称
    messages=message,
    tools = tools_,
    tool_choice = response_f.choices[0].message.tool_calls[0].function.name,
        )

print(response_a.choices[0].message.content)
```

GLM-4官网代码样例

```
messages.append({
        "role": "tool",
        "content": f"{json.dumps(function_result)}",
        "tool_call_id":tool_call.id
    })

response = client.chat.completions.create(
    model="glm-4",  # 填写需要调用的模型名称
    messages=messages,
    tools=tools,
)
print(response.choices[0].message)
messages.append(response.choices[0].message.model_dump())
```

升级优化

```
message = []
message.append({
            "role": "assistant",
            "content": "你使用了tools工具，最终获得的答案是" + str(function_response),
    })
message.append({"role": "user", "content": prompt + "请仅仅回答答案的数字，不要包括其他描述"})

response_a = client.chat.completions.create(
        model="glm-4",  # 填写需要调用的模型名称
        messages=message
        )

print(response_a.choices[0].message.content)
```

Prompt	测试结果
第一次	函数正确调用，结果正确。
第二次	函数正确调用，结果正确。
第三次	函数正确调用，结果正确。
第四次	函数正确调用，结果正确。
第五次	函数正确调用，结果正确。

012、Function Call实战进阶技巧

第二个挑战：海量函数

GLM-4

提供了更强大的问答和文本生成能力。适合于复杂的对话交互和深度内容创作设计的场景。

模型编码	描述	上下文长度
glm-4	最新的 GLM-4 、最大支持 128k 上下文、支持 Function Call 、Retreival。	128k tokens
glm-4v	实现了视觉语言特征的深度融合，支持视觉问答、图像字幕、视觉定位、复杂目标检测等各类多模态理解任务。	2k tokens

ChatGLM-Turbo

适用于对知识量、推理能力、创造力要求较高的场景，比如广告文案、小说写作、知识类写作、代码生成等。

模型编码	描述	上下文长度
glm-3-turbo	最新的glm-3-turbo、最大支持 128k上下文、支持 Function Call、Retreival。	128k tokens
chatglm_turbo		32k tokens

128K成为了顶级大模型的标配

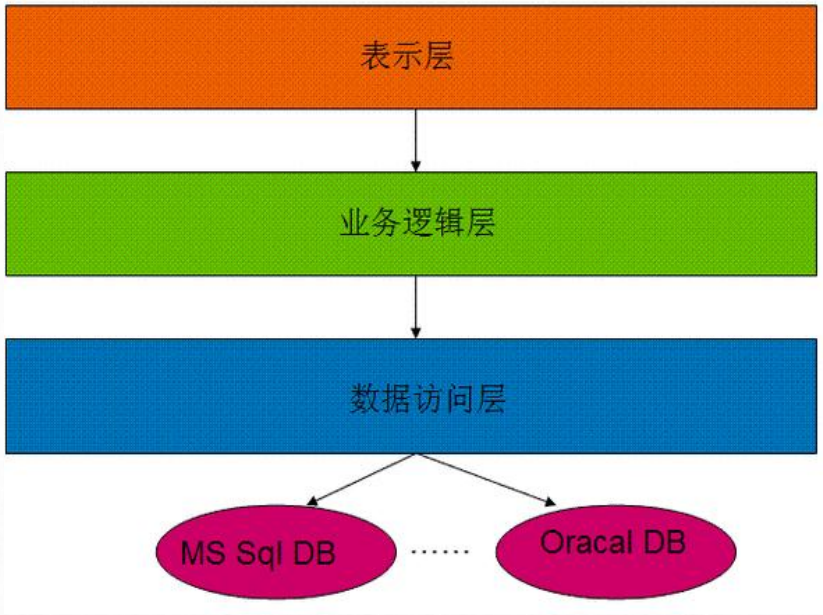
Token限制的问题

计算延迟的问题

意图识别的问题

13、Function Call实战进阶技巧

分层设计->粗略对齐->精准对齐



让模型进行意图识别，识别出来，应该是哪个类型的业务，就定位到哪个集合的工具

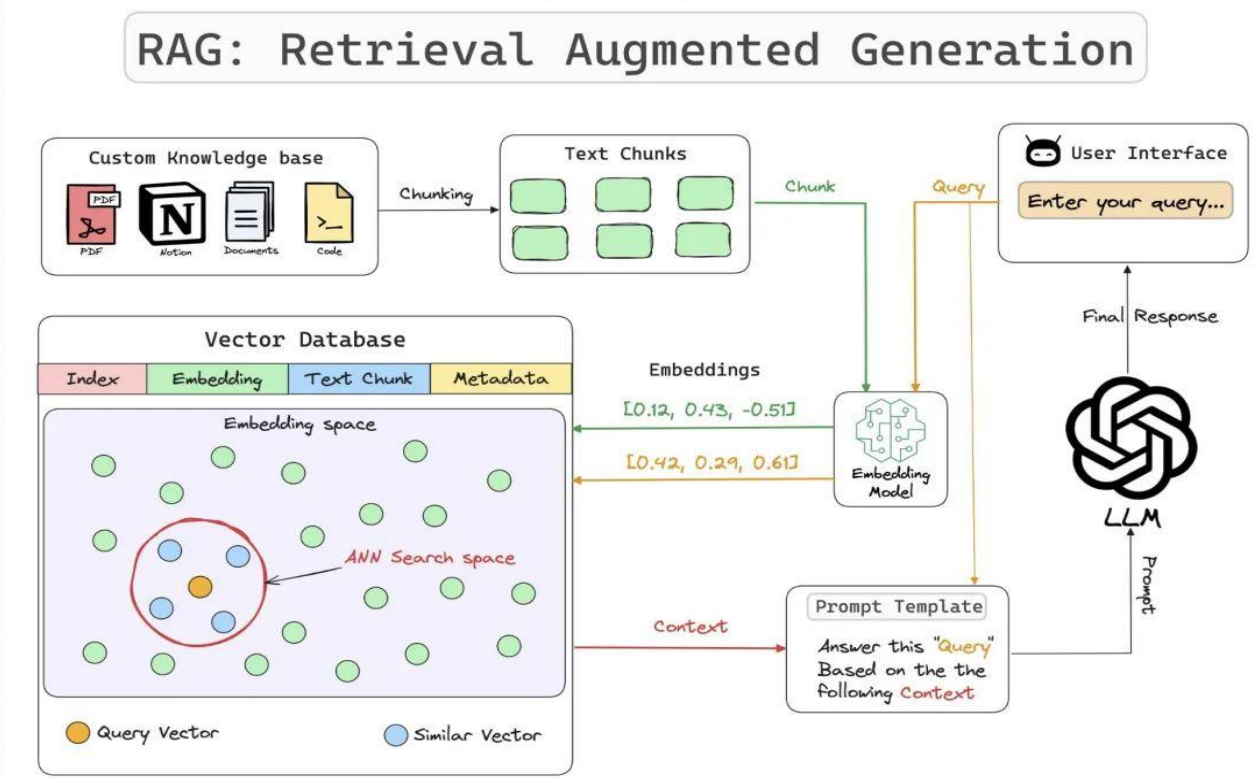
精准函数调用（流程跟前面一样）

按照功能，按照规律，按照业务，
按照生命周期等进行拆分

14、Function Call实战进阶技巧

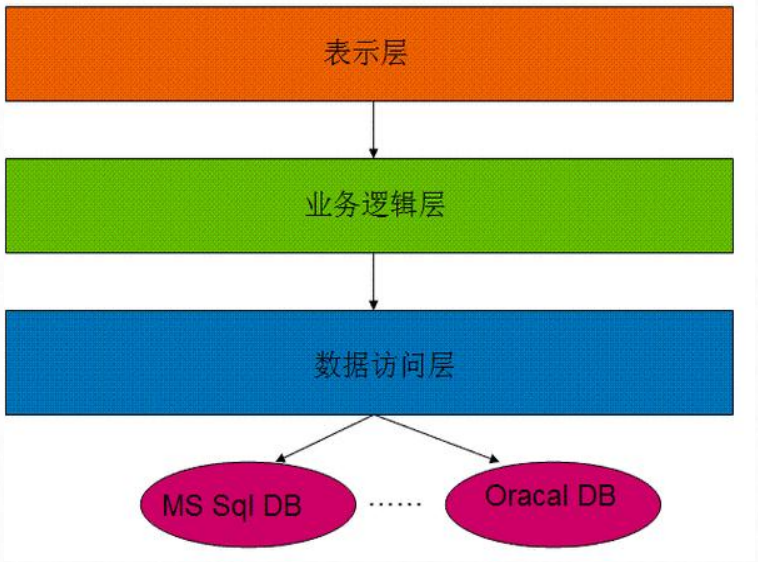
分层设计->粗略对齐->精准对齐

```
"""
    相比于之前的版本，进行了如下调整——
    首先，先对用户的意图进行识别，判断是哪一类问题
    """
    prompt = prompt
    judge_prompt = "请问上述问题最接近哪一类功能问题？\
    请回应数据访问层、业务逻辑层、表示层，或其他。\
    不要回应别的任何文字。"
    message = [
        {"role": "user", "content": prompt + judge_prompt}
    ]

    #进行初次意图识别
    response_first_align = client.chat.completions.create(
    model="glm-4",
    messages=message #此时其实最好的方式是使用function_call而不是prompt
    )

    #根据意图识别出的数据类别，定位相应类别的函数群
    #在这里进行关键字匹配
    func_type = response_first_align.choices[0].message.content
    current_function_tools = None

    if "数据访问层" in func_type:
        current_function_tools = function_type["数据访问层"]
    elif "业务逻辑层" in func_type:
        current_function_tools = function_type["业务逻辑层"]
    elif "表示层" in func_type:
        current_function_tools = function_type["表示层"]
    else:
        current_function_tools = function_type["其他"]

    print("初步意图识别结果:{}".format(func_type),"\n")

    #确认函数群后，直接使用原始prompt在函数群中执行function call
    message = [{"role":"assistant","content":"当你被问到功能问题时，\
                                             尝试调用Tools来解决问题。"}]
    message.append({"role": "user", "content": prompt})

    response_f = client.chat.completions.create(
        model="glm-4",
        messages=message,
        tools = current_function_tools,
        tool_choice = "auto"
        )
```



15、Function Call实战进阶技巧

分层设计->粗略对齐->精准对齐



让模型进行意图识别，识别出来，应该是哪个类型的业务，就定位到哪个集合的工具

精准函数调用（流程跟前面一样）

识别出来一组函数以后，发现函数多了以后，就不能精准调用了。

预料太长了后，模型无法区分Prompt，函数库，知识库等。

从提示词中就告诉大模型，只需要给我返回来函数的名称即可，然后我们自己完成函数的调用。

16、Function Call实战进阶技巧

第三个挑战：并行调用

串行调用

prompt = "请问6565656565与3454321的和与积作差等于几？"

并行调用

prompt = """请帮我在用户表中增加如下用户：
用户名：张三，密码：txt_001,
用户名：李四，密码：txt_004,
用户名：王五，密码：txt_006,
用户名：赵六，密码：txt_008,
"""

17、Function Call实战进阶技巧

串行调用

```
prompt = "请问6565656565与3454321的和与积作差等于几？"

message.append({"role": "user", "content": prompt + "解决这个问题要分几步？\
                                                    只告诉我步骤，不要解决问题\
                                                    请按照'1. xxx;2. xxx;...;的格式列举\
                                                    步骤与步骤之间使用半角分号;相隔，整个句子请用半角分号;结尾"})

response_step = client.chat.completions.create(
    model="glm-4",
    messages=message,
    temperature = 0.1
    )
```

```
# 定义正则表达式模式
pattern = r'(.*?;)'

# 使用re.findall提取所有匹配项
steps = re.findall(pattern, str_content)

# 去除最后一个不完整的匹配项
steps = [step.strip() for step in steps if step.strip()]

for step in steps:
    print(step)
```

1. 计算两个数的和
2. 计算两个数的积
3. 对结果进行相减

```
step_answer = ""
# 步骤之间相互独立或者有联系，都可以用
for step in steps:
    message = [{"role":"assistant","content":prompt + "正在分步骤解决问题。已知条件是" + step_answer}]
    message.append({"role":"user","content":"当前步骤是" + step})

    response_per_step = client.chat.completions.create(
        model="glm-4",
        messages=message,
        tools = basic_math_tools,
        tool_choice = True,
        temperature = 0.1
        )

    try:
        function_calls = response_per_step.choices[0].message.tool_calls[0]
        function_to_call = available_math_functions[function_calls.function.name]
        function_args = json.loads(function_calls.function.arguments)
        function_response = function_to_call(**function_args)
    except:
        function_response = response_per_step.choices[0].message.content

    step_answer = step + "答案是" + str(function_response)
    print(step_answer)

#经历完整的步骤、获得答案之后再返回
message = []
message.append({
        "role": "assistant",
        "content": "你使用了tools工具，最终获得的答案是" + str(function_response),
    })
message.append({"role": "user", "content": prompt + "请仅仅回答答案的数字，不要包括其他描述"})

response_a = client.chat.completions.create(
    model="glm-4",  # 填写需要调用的模型名称
    messages=message
    )

print(response_a.choices[0].message.content)
```

18、Function Call实战进阶技巧

并行调用

batch

```
messages1 = [SystemMessage(content="你是一位乐于助人的智能小助手"),
 HumanMessage(content="请帮我介绍一下什么是机器学习"),]
```

```
messages2 = [SystemMessage(content="你是一位乐于助人的智能小助手"),
 HumanMessage(content="请帮我介绍一下什么是AIGC"),]
```

```
messages3 = [SystemMessage(content="你是一位乐于助人的智能小助手"),
 HumanMessage(content="请帮我介绍一下什么是大模型技术"),]
```

```
reponse = chat.batch([messages1,
            messages2,
            messages3,])

reponse
```

```
[AIMessage(content='机器学习是一种人工智能的分支领域，其目标是让计算机系统通过学习数据和模式识别，从而能够自动进行决策和预测。机器学习利用统计学和算法来让计算机系统从数据中学习，改进和发展自身的性能，而无需明确地进行编程。通过机器学习，计算机系统可以通过大量的数据训练和优化自己的模型，以实现各种任务，如图像识别、语音识别、自然语言处理、推荐系统等。机器学习的应用范围非常广泛，正在逐渐改变我们的日常生活和工作方式。', response_metadata={'token_usage': {'completion_tokens': 208, 'prompt_tokens': 47, 'total_tokens': 255}, 'model_name': 'gpt-3.5-turbo', 'system_fingerprint': 'fp_2f57f81c11', 'finish_reason': 'stop', 'logprobs': None}, id='run-e225be84-dbe8-47ec-9207-0bd5113721fa-0'),
 AIMessage(content='AIGC是Artificial Intelligence Graduate Certificate的缩写，即人工智能研究生证书。AIGC是一种专业课程，旨在培养学生在人工智能领域的技能和知识。这个证书课程通常由大学或学院提供，需要学生完成一系列的课程和项目。课程涵盖了人工智能的核心技术，包括机器学习、自然语言处理、图像识别等。获得AIGC证书的学生可以在人工智能领域的职业中取得更好的就业机会。', response_metadata={'token_usage': {'completion_tokens': 179, 'prompt_tokens': 47, 'total_tokens': 226}, 'model_name': 'gpt-3.5-turbo', 'system_fingerprint': None, 'finish_reason': 'stop', 'logprobs': None}, id='run-ce431921-e742-498d-82e9-2b22a98bba95-0'),
 AIMessage(content='大模型技术是指利用大规模的数据和计算资源来训练和部署复杂、庞大的机器学习模型的技术。随着数据量的不断增加和计算能力的提升，大模型技术在人工智能领域变得越来越重要。\n\n大模型技术通常涉及使用大规模的数据集来训练深度神经网络等复杂模型，以提高模型的准确性和泛化能力。同时，大模型技术还需要大量的计算资源来训练这些模型，包括高性能的计算机、GPU加速器等硬件设备。\n\n大模型技术在各种领域都有广泛的应用，包括自然语言处理、计算机视觉、语音识别等。通过大模型技术，研究人员和工程师们可以构建更加复杂和强大的机器学习模型，从而实现更多领域的创新和应用。', response_metadata={'token_usage': {'completion_tokens': 303, 'prompt_tokens': 48, 'total_tokens': 351}, 'model_name': 'gpt-3.5-turbo', 'system_fingerprint': 'fp_2f57f81c11', 'finish_reason': 'stop', 'logprobs': None}, id='run-1d8f25dc-010f-49b3-a51b-86d7549d48ed-0')]
```

19、Function Call实战进阶技巧

并行调用

创建 Batch 文件

Batch 文件的格式应为 .jsonl，其中每个请求占据一行（JSON 对象）。每一行包含 API 单个请求的详细信息。每个请求的定义如下：

glm-4 | glm-4v | cogview-3 | embedding-2

```
{
    "custom_id": "request-1",  #每个请求必须包含custom_id且是唯一的,长度必须为 6 -64 位.用来将结果和输入进行匹配.
    "method": "POST",
    "url": "/v4/chat/completions",
    "body": {
        "model": "glm-4",     #每个batch文件只能包含对单个模型的请求,支持 glm-4、glm-3-turbo.
        "messages": [
            {"role": "system","content": "你是一个意图分类器."},
            {"role": "user", "content": """
            # 任务：对以下用户评论进行情感分类和特定问题标签标注，只输出结果，
            # 评论：review = "订单处理速度太慢，等了很久才发货。"
            # 输出格式：
            {
            "分类标签": " ",
            "特定问题标注": " "
            }
            """
            }
        ],
        "temperature": 0.1
    }
}
```

构建的 .jsonl 文件如下，本示例中包含 10 个请求，单个文件最多支持 50000 个请求且大小不超过 100M：

```
{"custom_id": "request-1", "method": "POST", "url": "/v4/chat/completions", "body": {"model": "glm-4", "messages": [{"role": "syste
{"custom_id": "request-2", "method": "POST", "url": "/v4/chat/completions", "body": {"model": "glm-4", "messages": [{"role": "syste
{"custom_id": "request-3", "method": "POST", "url": "/v4/chat/completions", "body": {"model": "glm-4", "messages": [{"role": "syste
{"custom_id": "request-4", "method": "POST", "url": "/v4/chat/completions", "body": {"model": "glm-4", "messages": [{"role": "syste
{"custom_id": "request-5", "method": "POST", "url": "/v4/chat/completions", "body": {"model": "glm-4", "messages": [{"role": "syste
{"custom_id": "request-6", "method": "POST", "url": "/v4/chat/completions", "body": {"model": "glm-4", "messages": [{"role": "syste
{"custom_id": "request-7", "method": "POST", "url": "/v4/chat/completions", "body": {"model": "glm-4", "messages": [{"role": "syste
{"custom_id": "request-8", "method": "POST", "url": "/v4/chat/completions", "body": {"model": "glm-4", "messages": [{"role": "syste
{"custom_id": "request-9", "method": "POST", "url": "/v4/chat/completions", "body": {"model": "glm-4", "messages": [{"role": "syste
{"custom_id": "request-10", "method": "POST", "url": "/v4/chat/completions", "body": {"model": "glm-4", "messages": [{"role": "syst
```

20、Function Call实战进阶技巧

并行调用

上传文件

通过文件管理上传 Batch 文件并获取文件 ID。

```python
from zhipuai import ZhipuAI

client = ZhipuAI(api_key="") # 请填写您自己的APIKey

result = client.files.create(
    file=open("product_reviews.jsonl", "rb"),
    purpose="batch"
)
print(result.id)
```

创建 Batch 任务

成功上传输入文件后，您可以使Batch 文件的 id 创建 Batch 任务。在本示例中，我们假设文件 ID 为 file_123。

```python
from zhipuai import ZhipuAI

client = ZhipuAI()  # 填写您自己的APIKey

create = client.batches.create(
    input_file_id="file_123",
    endpoint="/v4/chat/completions",
    completion_window="24h", #完成时间只支持 24 小时
    metadata={
        "description": "Sentiment classification"
    }
)
print(create)
```

检查 Batch 状态

Batch 任务将在 24 小时内处理完成，状态为 "completed" 表示任务已完成。

```python
batch_job = client.batches.retrieve("batch_id")
print(batch_job)
```

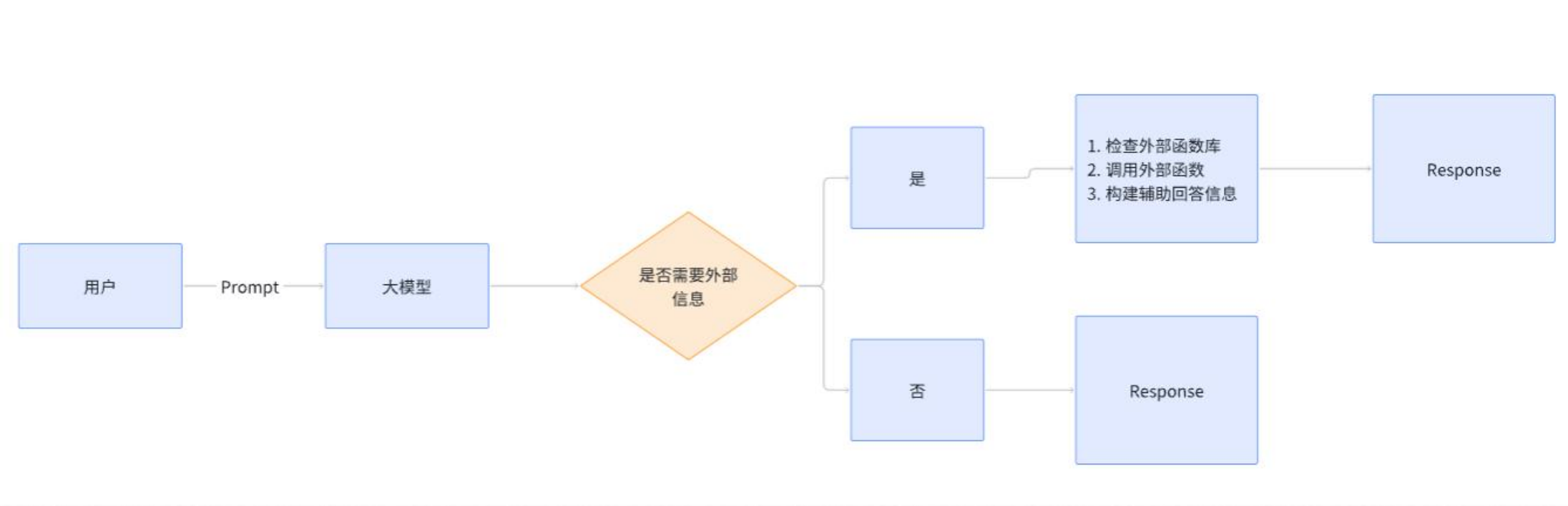
下载 Batch 结果

Batch 任务完成后，您可以使用 Batch 对象中的output_file_id 字段下载结果，并将其保存到本地。

```python
from zhipuai import ZhipuAI

client = ZhipuAI()  # 填写您自己的APIKey
# client.files.content返回 _legacy_response.HttpxBinaryResponseContent实例
content = client.files.content("file-456")

# 使用write_to_file方法把返回结果写入文件
content.write_to_file("write_to_file_batchoutput.jsonl")
```

21、Function Call实战进阶技巧

挑战四：响应太慢



意图识别，海量函数，并行执行

缓存方案设计

前三步配合使用

更多方案大家继续探索

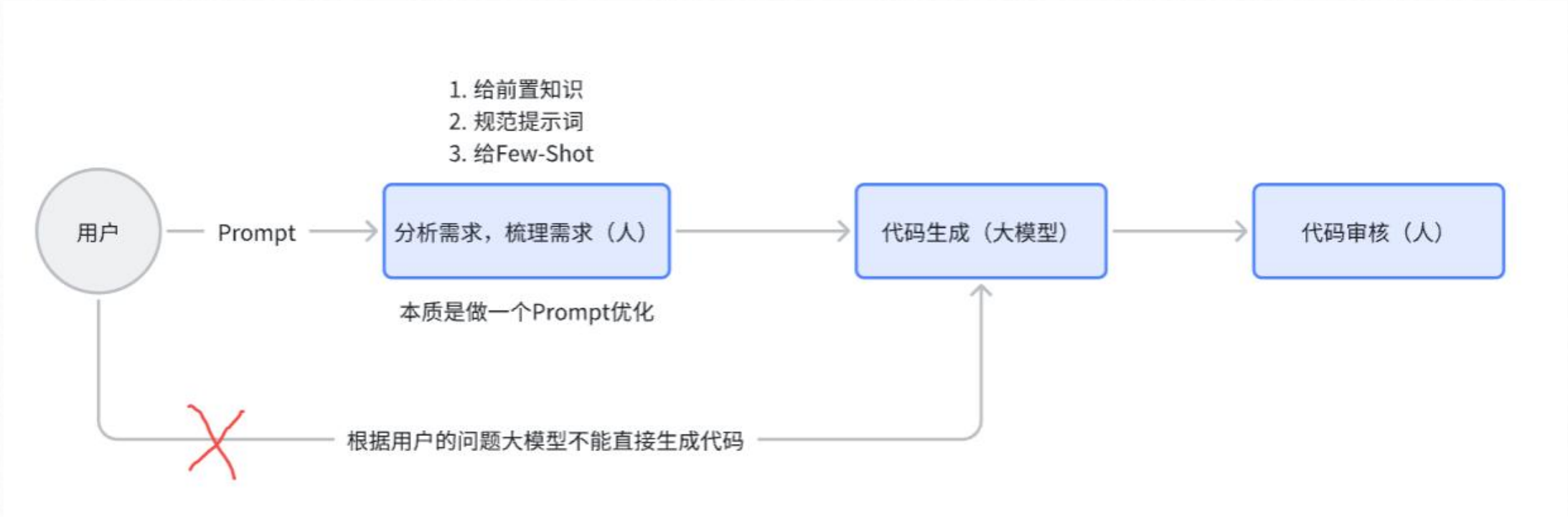
4 面向目标架构案例之代码落地

01、代码落地-阶段1

提示词工程技术

Function Call技术

02、代码落地-阶段1


1. 给前置知识
2. 规范提示词
3. 给Few-Shot
用户
Prompt
分析需求，梳理需求（人）
代码生成（大模型）
代码审核（人）
本质是做一个Prompt优化
根据用户的问题大模型不能直接生成代码

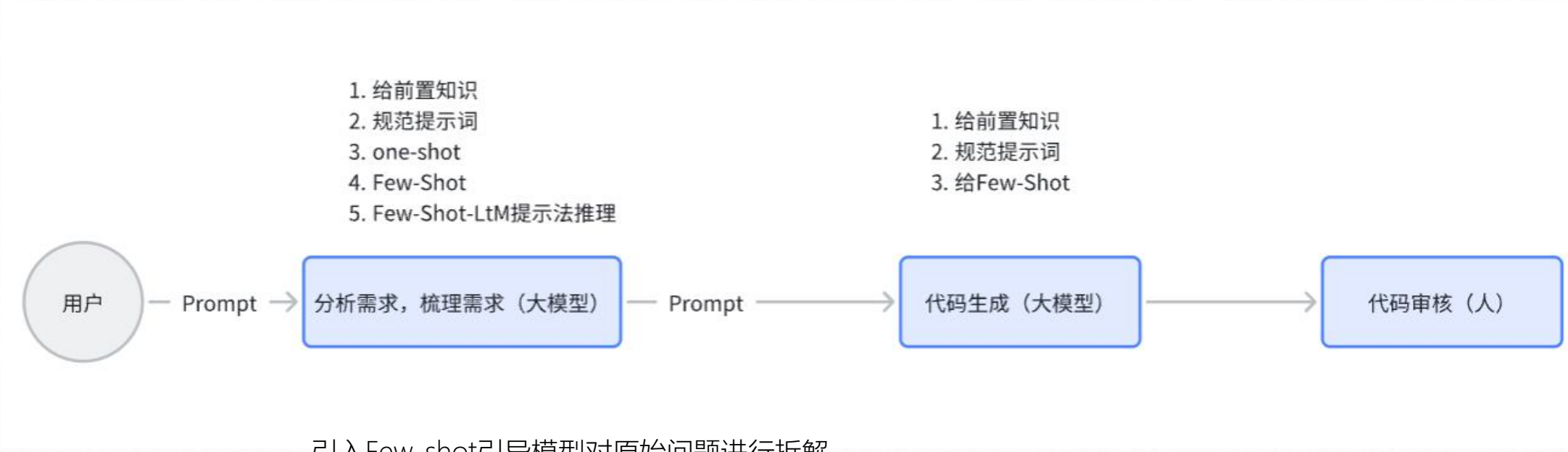
03、代码落地-阶段1

```
def sql_inter(sql_query):
    """
    用于获取iquery数据库中各张表的有关相关信息，\
    核心功能是将输入的SQL代码传输至iquery数据库所在的MySQL环境中进行运行，\
    并最终返回SQL代码运行结果。需要注意的是，本函数是借助pymysql来连接MySQL数据库。
    :param sql_query: 字符串形式的SQL查询语句，用于执行对MySQL中iquery数据库中各张表进行查询，并获得各表中的各类相关信息
    :return: sql_query在MySQL中的运行结果。
    """

    mysql_pw = "iquery_agent"

    connection = pymysql.connect(
            host='localhost',  # 数据库地址
            user='iquery_agent',  # 数据库用户名
            passwd=mysql_pw,  # 数据库密码
            db='iquery',  # 数据库名
            charset='utf8'  # 字符集选择utf8
        )

    try:
        with connection.cursor() as cursor:
            # SQL查询语句
            sql = sql_query
            cursor.execute(sql)

            # 获取查询结果
            results = cursor.fetchall()

    finally:
        connection.close()


    return json.dumps(results)
```

04、代码落地-阶段2


面向目标架构
以人为主，AI为辅
1. 人梳理需求
2. 大模型帮忙写代码
3. 人审核代码
AI和人相互配合
1. 大模型梳理需求
2. 大模型帮忙写代码
3. 人审核代码
AI为主，以人为辅
1. 大模型梳理需求
2. 大模型帮忙写代码
3. 大模型审核代码

04、代码落地-阶段2



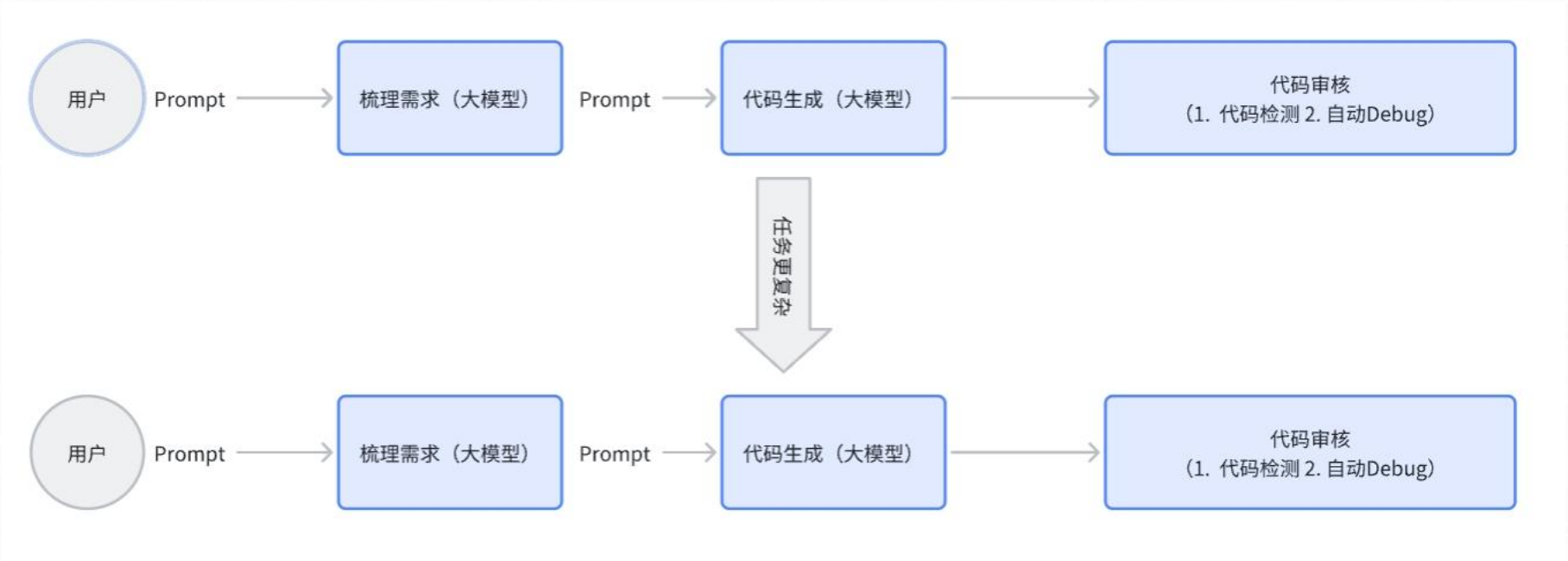
引入Few-shot引导模型对原始问题进行拆解

引入Few-shot引导模型对子问题再次进行多步分解

05、代码落地-阶段3


面向目标架构
以人为主，AI为辅
1. 人梳理需求
2. 大模型帮忙写代码
3. 人审核代码
AI和人相互配合
1. 大模型梳理需求
2. 大模型帮忙写代码
3. 人审核代码
AI为主，以人为辅
1. 大模型梳理需求
2. 大模型帮忙写代码
3. 大模型审核代码

01、AI大模型架构师知识准备



01、AI大模型架构师知识准备

01、AI大模型架构师知识准备

01、AI大模型架构师知识准备

01、AI大模型架构师知识准备

01、AI大模型架构师知识准备

01、AI大模型架构师知识准备

01、AI大模型架构师知识准备

2 AI 大模型架构师课程知识准备

01、AI大模型架构师知识准备

知识点	掌握程度
提示词工程	1. 掌握常见的提示词技巧，如思维链，few-shot等。
外部函数调用	1. 知道外部函数调用的详细流程 2. 熟练编写外部函数调用的代码
RAG	1. 熟练掌握RAG的整个流程 2. 能动手实现简单的RAG流程
微调	1. 知道常见的微调的方式 2. 知道微调的原理