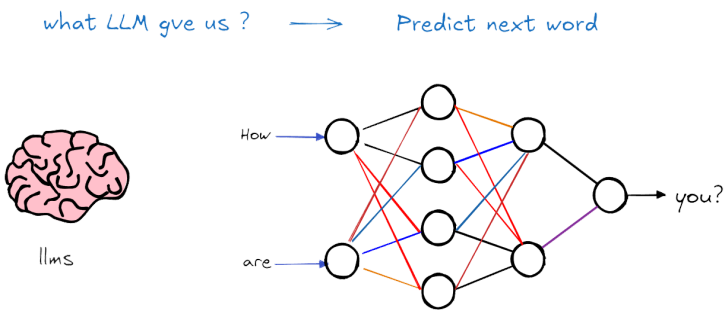
大模型 AI Agent 开发实战

Ch.2 AI Agent应用类型及Function Calling开发实战

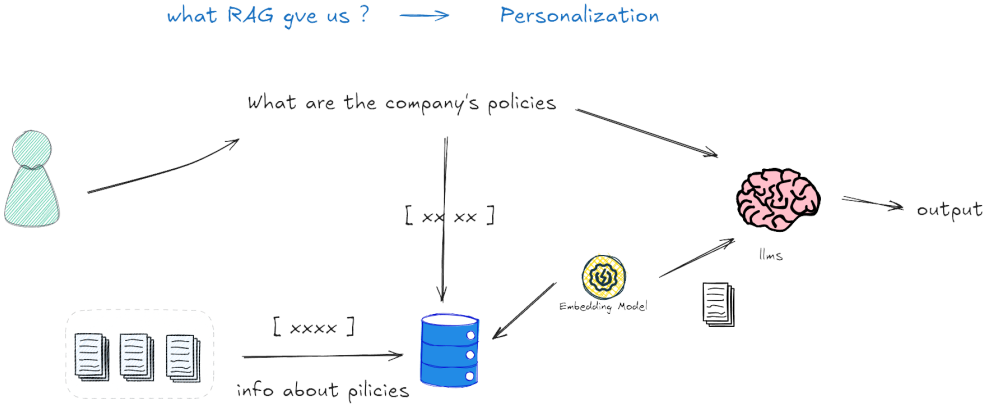
在上一节课中，我们介绍了近两年大模型技术的迅速发展及其技术演进，这包括从大模型自身的能力持续突破（原生能力和涌现能力），基本的函数调用功能，到引入 RAG（检索增强生成）技术，再到当前的 AI Agent（代理）技术。如果说 2023 年是检索增强生成年，那么 2024 年毫无疑问就是代理年。全球各地的公司都在探索使用机器人代理。究其原因还是在于虽然大模型结合RAG技术已经突破了语言生成的可能性界限，但是其存在着极大的局限性。

首先我们来看大语言模型自身的特性。它能够类似于人类大脑那样处理和生成语言，其功能可以就像一个庞大的信息库，里面存储着丰富的语言知识和数据。这使得大模型不仅能理解复杂的语言结构，还能创造出流畅的文本输出。👇



如上图所示展示了一个神经网络的简化示意图。输入词 `how` 和 `are` 通过多层神经网络，其中的每个节点代表神经网络中的一个 `神经元`，而连接线代表不同神经元之间的连接。这些连接（权重）会根据训练数据进行调整，以便大模型能够更好地预测输出。其终点指出大模型的输出是“you?”，这是在给定输入“how are”后，大模型预测的下一个最可能的词。

RAG技术通过查找相关信息来改进大模型的输出，使其能够根据特定的信息进行响应生成。👇



可以看到，当大模型和RAG技术组合在一起，重点仍然是**知识和文本生成。**它们侧重于根据训练数据中的模式生成类似人类的文本，主要处理单个输入并据此提供响应，这就导致了这样的架构组合，一定是**缺乏以灵活、明智的方式设定和追求特定目标的能力。**人工智能代理的一个重要方面是它们的学习和适应能力。通过集成大语言模型和RAG等技术，它们在交互的基础上不断提高性能，随着时间的推移演变成更复杂、更智能的助手。通过不同场景的对比分析可以非常明确的感受到：

- **面向目标的行为**：大模型 和 RAG 模型主要通过模仿训练数据中的模式来生成类似人类的文本。然而，它们通常缺乏以灵活且明智的方式设定和追求具体目标的能力。与之相比，人工智能代理能被设计来拥有明确的目标，并通过计划及行动来实现这些目标。

- **记忆和状态跟踪**：大多数现有的大语言模型不具备持久记忆或状态跟踪能力，通常将每个输入视为孤立事件处理。相反，人工智能代理能维持记忆状态，随时间积累知识，并利用这些状态信息支持未来的决策和行动。
- **与环境的交互**：大模型仅在文本层面上操作，与物理世界无直接互动。人工智能代理则能感知并与其所处的环境互动，无论是在数字领域、机器人系统，还是通过传感器和执行器的物理环境中。
- **迁移和泛化能力**：尽管大模型 在处理与训练数据相似的语言任务方面表现出色，它们往往难以将所学知识迁移到全新的领域或任务上。人工智能代理则展现了在新情况下学习、推理和计划的能力，可能更有效地迁移和泛化。
- **持续学习**：大部分大语言模型在训练完成后便固定下来，不再更新。而人工智能代理能够在与新环境和情况交互时持续学习和调整，不断优化其知识和技能。
- **多任务能力**：大模型 通常专注于特定的语言处理任务。而人工智能代理被设计为能处理多种任务的通用系统，可以整合语言处理、推理、感知和控制等多方面技能，以解决复杂的问题。

通过上述对比分析，相信大家已经对大模型、RAG 和 AI Agent 三者在大模型的应用技术领域中的不同之处有了初步的认识。而接下来，我们将探讨这三者之间的紧密联系和相互作用。

1. 聊天机器人

我们在【Ch.1 大模型应用发展及Agent前沿技术趋势】的最后一部分内容中，借助 AI Agent 代理框架汇总 介绍了目前多元化的AI Agent 构建框架，除了单独的开源项目，`LangGraph` 和 `LlamaIndex Workflows` 等框架也正在帮助世界各地的开发人员构建结构化代理。但**目前的现状是：尽管智能体很受欢迎，但它们尚未在人工智能生态系统之外引起轰动**。很少有成熟的代理能够在消费者或企业用户中取得成功。团队如何驾驭新框架和新代理方向？有哪些工具可用，我们应该使用哪些工具来构建应用程序?

思考这个问题很关键，因为`AI Agent`是一个涵盖范围极广的概念。从简单的智能问答，到复杂的工作流程，均可被归类为 `AI Agent`，但不同的场景需求，它们背后所依赖的技术和框架可能相差甚远。我们可以像定义人工智能发展阶段的方式一样来区分狭义和广义的 `AI Agent`。但无论采用何种称呼，大家非常容易陷入的误区就是：**简单任务、复杂实现**。目前市面上的大模型智能应用产品，如 `ChatGPT`、`智谱清言` 和 `Coze`等，如果从背后技术栈的角度来看，可以被分为三类：**聊天机器人、人工智能助手和人工智能代理**。这三类应用代表着不同的技术应用体系，也是最容易导致大家出现**简单任务复杂化**的主要原因。例如，有时候仅需大模型加上 RAG 就能解决的问题，却非要引入外部函数调用；或者在函数调用足以应对的场景中，却过度构建代理框架。这种现象的根本原因在于对这三种应用类别的理解不够透彻，或对各种技术栈的潜在效能理解不清。所以，在深入探讨构建 AI Agent 的不同开发框架之前，我们将通过三个实际案例，在介绍AI Agent必须掌握的基础知识的同时，分别来实际的构建**聊天机器人、人工智能助手和人工智能代理**三个场景，**让大家深入理解这些概念，同时帮助大家在接收到实际的开发任务时，能够迅速做出最正确的的技术选型。**

首先，我们从用户的角度直观的感受这三种智能应用带给我们的使用体验：

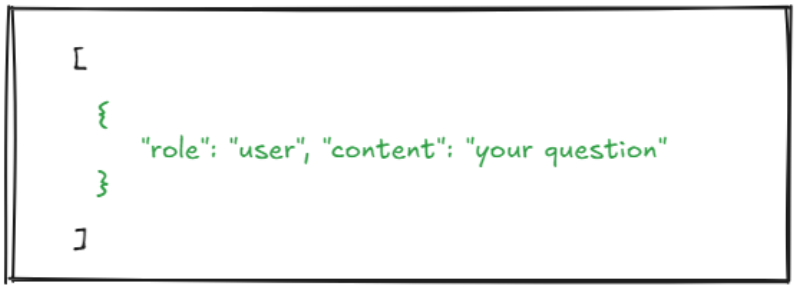
- 聊天机器人主要设计用于进行对话，回答简单问题或提供客户服务。这些机器人有的非常基础，仅依靠预设的回复；而更先进的聊天机器人可以靠RAG等技术来提升对话质量，尽管如此，它们处理的问题范围通常相对有限。
- 人工智能助手，这些工具的功能更为全面和复杂。它们不仅能搜索信息和生成内容，还能管理家中的智能设备等。通过使用人工智能理解上下文和用户偏好，这些助手就像个人定制的私人助理。
- 人工智能代理，这些系统是所有智能应用中功能最强大的。它们被设计来自主执行多种任务、做出决策和解决问题，并且能够与现实世界互动。例如，它们能为您预订航班、在线购物，甚至在合同谈判中代表您行事。

在这里，大家可能会有疑问：**像 `ChatGPT` 这样的应用，在对话过程中也能执行搜索信息等任务，为什么它们还是被认为是聊天机器人？**这正是我们接下来要解释的问题。事实上，**这些应用具备人工智能助手的功能，只不过是通过聊天机器人的形式为我们提供服务。**通过即将展示的案例，大家将能更深刻地理解这一点。接下来，我们就以 `ChatGPT` 应用为例来进行第一个案例：聊天机器人的实现。

`ChatGPT` 于 2022 年 11 月推出，建立在 OpenAI 的 GPT-3（最初是 GPT-3.5模型 ）大语言模型系列之上，并使用监督学习和强化学习技术进行微调。此后，`ChatGPT` 引发了人工智能热潮，推出仅两个月就拥有 1 亿用户。除了使用其官方提供的应用服务（ChatGPT），对于开发人员来说，我们可以通过底层的开发语言将大模型集成到我们个人的应用程序中。这里我们选择使用 Python 进行演示。

> 对于OpenAI GPT模型的API调用的基本介绍，大家可以参考我们之前课程的内容，在《大模型技术实战课》的 Ch 1 至 Ch 8 中有详细的解析。

当我们通过大模型的API进行会话交互时，数据交换的格式如下图所示 👇



上图中展示的 `JSON` 结构是API交互中常用的数据格式。不仅仅是 `OpenAI` 的 `GPT` 模型，其他在线模型或者开源模型均遵循这种规范。在这个例子中，我们看到有一个对象包含两个键值对："role": "user"和"content": "your question"。这表示交互中用户的角色被标识为"user"，并且用户的输入内容为"your question"。这种格式化的数据结构允许API清晰地解析出发送者的角色和对话内容，从而进行有效的处理和响应。

要发送到 `GPT` 模型的问题必须包含在字典列表中。我们使用OpenAI的Python客户端库来调用GPT模型以获取聊天回复，来看代码实现：

```
from openai import OpenAI
client = OpenAI()

response = client.chat.completions.create(
  model="gpt-4o-mini",
  messages=[
    {"role": "user", "content": "请你详细的介绍一下：什么是人工智能？"},
  ]
)
```

```
--------------------------------------------------------------------------

ConnectError                              Traceback (most recent call last)

File ~\anaconda3\envs\agent\Lib\site-packages\httpx\_transports\default.py:72, in
map_httpcore_exceptions()
     71 try:
---> 72     yield
     73 except Exception as exc:
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\httpx\_transports\default.py:236, in
HTTPTransport.handle_request(self, request)
    235 with map_httpcore_exceptions():
--> 236     resp = self._pool.handle_request(req)
    238 assert isinstance(resp.stream, typing.Iterable)
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\httpcore\_sync\connection_pool.py:216,
in ConnectionPool.handle_request(self, request)
    215     self._close_connections(closing)
--> 216     raise exc from None
    218 # Return the response. Note that in this case we still have to manage
    219 # the point at which the response is closed.
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\httpcore\_sync\connection_pool.py:196,
in ConnectionPool.handle_request(self, request)
    194 try:
    195     # Send the request on the assigned connection.
--> 196     response = connection.handle_request(
    197         pool_request.request
    198     )
    199 except ConnectionNotAvailable:
    200     # In some cases a connection may initially be available to
    201     # handle a request, but then become unavailable.
    202     #
    203     # In this case we clear the connection and try again.
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\httpcore\_sync\http_proxy.py:317, in
TunnelHTTPConnection.handle_request(self, request)
    316 with Trace("start_tls", logger, request, kwargs) as trace:
--> 317     stream = stream.start_tls(**kwargs)
    318     trace.return_value = stream
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\httpcore\_sync\http11.py:383, in
HTTP11UpgradeStream.start_tls(self, ssl_context, server_hostname, timeout)
    377 def start_tls(
    378     self,
    379     ssl_context: ssl.SSLContext,
    380     server_hostname: Optional[str] = None,
    381     timeout: Optional[float] = None,
    382 ) -> NetworkStream:
--> 383     return self._stream.start_tls(ssl_context, server_hostname, timeout)
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\httpcore\_backends\sync.py:152, in
SyncStream.start_tls(self, ssl_context, server_hostname, timeout)
    148 exc_map: ExceptionMapping = {
    149     socket.timeout: ConnectTimeout,
    150     OSError: ConnectError,
    151 }
--> 152 with map_exceptions(exc_map):
    153     try:
```

```
File ~\anaconda3\envs\agent\Lib\contextlib.py:155, in
_GeneratorContextManager.__exit__(self, typ, value, traceback)
    154 try:
--> 155     self.gen.throw(typ, value, traceback)
    156 except StopIteration as exc:
    157     # Suppress StopIteration *unless* it's the same exception that
    158     # was passed to throw().  This prevents a StopIteration
    159     # raised inside the "with" statement from being suppressed.
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\httpcore\_exceptions.py:14, in
map_exceptions(map)
     13     if isinstance(exc, from_exc):
---> 14         raise to_exc(exc) from exc
     15 raise
```

```
ConnectError: [WinError 10054] 远程主机强迫关闭了一个现有的连接。
```

```
The above exception was the direct cause of the following exception:
```

```
ConnectError                              Traceback (most recent call last)

File ~\anaconda3\envs\agent\Lib\site-packages\openai\_base_client.py:973, in
SyncAPIClient._request(self, cast_to, options, remaining_retries, stream,
stream_cls)
    972 try:
--> 973     response = self._client.send(
    974         request,
    975         stream=stream or self._should_stream_response_body(request=request),
    976         **kwargs,
    977     )
    978 except httpx.TimeoutException as err:
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\httpx\_client.py:926, in
Client.send(self, request, stream, auth, follow_redirects)
    924 auth = self._build_request_auth(request, auth)
--> 926 response = self._send_handling_auth(
    927     request,
    928     auth=auth,
    929     follow_redirects=follow_redirects,
    930     history=[],
    931 )
    932 try:
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\httpx\_client.py:954, in
Client._send_handling_auth(self, request, auth, follow_redirects, history)
    953 while True:
--> 954     response = self._send_handling_redirects(
    955         request,
    956         follow_redirects=follow_redirects,
    957         history=history,
    958     )
    959     try:
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\httpx\_client.py:991, in
Client._send_handling_redirects(self, request, follow_redirects, history)
    989     hook(request)
--> 991 response = self._send_single_request(request)
    992 try:
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\httpx\_client.py:1027, in
Client._send_single_request(self, request)
   1026 with request_context(request=request):
-> 1027     response = transport.handle_request(request)
   1029 assert isinstance(response.stream, SyncByteStream)
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\httpx\_transports\default.py:235, in
HTTPTransport.handle_request(self, request)
    223 req = httpcore.Request(
    224     method=request.method,
    225     url=httpcore.URL(
   (...)
    233     extensions=request.extensions,
    234 )
--> 235 with map_httpcore_exceptions():
    236     resp = self._pool.handle_request(req)
```

```
File ~\anaconda3\envs\agent\Lib\contextlib.py:155, in
_GeneratorContextManager.__exit__(self, typ, value, traceback)
    154 try:
--> 155     self.gen.throw(typ, value, traceback)
    156 except StopIteration as exc:
    157     # Suppress StopIteration *unless* it's the same exception that
    158     # was passed to throw().  This prevents a StopIteration
    159     # raised inside the "with" statement from being suppressed.
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\httpx\_transports\default.py:89, in
map_httpcore_exceptions()
     88 message = str(exc)
---> 89 raise mapped_exc(message) from exc
```

```
ConnectError: [WinError 10054] 远程主机强迫关闭了一个现有的连接。
```

```
The above exception was the direct cause of the following exception:
```

```
APIConnectionError                        Traceback (most recent call last)

Cell In[1], line 4
      1 from openai import OpenAI
      2 client = OpenAI()
----> 4 response = client.chat.completions.create(
      5   model="gpt-4o-mini",
      6   messages=[
      7     {"role": "user", "content": "请你详细的介绍一下：什么是人工智能？"},
      8   ]
      9 )
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\openai\_utils\_utils.py:274, in
required_args.<locals>.inner.<locals>.wrapper(*args, **kwargs)
    272             msg = f"Missing required argument: {quote(missing[0])}"
    273     raise TypeError(msg)
--> 274 return func(*args, **kwargs)
```

```
File ~\anaconda3\envs\agent\Lib\site-
packages\openai\resources\chat\completions.py:668, in Completions.create(self,
messages, model, frequency_penalty, function_call, functions, logit_bias, logprobs,
max_tokens, n, parallel_tool_calls, presence_penalty, response_format, seed,
service_tier, stop, stream, stream_options, temperature, tool_choice, tools,
top_logprobs, top_p, user, extra_headers, extra_query, extra_body, timeout)
    633 @required_args(["messages", "model"], ["messages", "model", "stream"])
    634 def create(
    635     self,
   (...)
    665     timeout: float | httpx.Timeout | None | NotGiven = NOT_GIVEN,
    666 ) -> ChatCompletion | Stream[ChatCompletionChunk]:
    667     validate_response_format(response_format)
--> 668     return self._post(
    669         "/chat/completions",
    670         body=maybe_transform(
    671             {
    672                 "messages": messages,
    673                 "model": model,
    674                 "frequency_penalty": frequency_penalty,
    675                 "function_call": function_call,
    676                 "functions": functions,
    677                 "logit_bias": logit_bias,
    678                 "logprobs": logprobs,
    679                 "max_tokens": max_tokens,
    680                 "n": n,
    681                 "parallel_tool_calls": parallel_tool_calls,
    682                 "presence_penalty": presence_penalty,
    683                 "response_format": response_format,
    684                 "seed": seed,
    685                 "service_tier": service_tier,
    686                 "stop": stop,
    687                 "stream": stream,
    688                 "stream_options": stream_options,
    689                 "temperature": temperature,
    690                 "tool_choice": tool_choice,
    691                 "tools": tools,
    692                 "top_logprobs": top_logprobs,
    693                 "top_p": top_p,
    694                 "user": user,
    695             },
    696             completion_create_params.CompletionCreateParams,
    697         ),
    698         options=make_request_options(
```

```
    699             extra_headers=extra_headers, extra_query=extra_query,
extra_body=extra_body, timeout=timeout
    700         ),
    701         cast_to=ChatCompletion,
    702         stream=stream or False,
    703         stream_cls=Stream[ChatCompletionChunk],
    704     )
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\openai\_base_client.py:1260, in
SyncAPIClient.post(self, path, cast_to, body, options, files, stream, stream_cls)
   1246 def post(
   1247     self,
   1248     path: str,
   (...)
   1255     stream_cls: type[_StreamT] | None = None,
   1256 ) -> ResponseT | _StreamT:
   1257     opts = FinalRequestOptions.construct(
   1258         method="post", url=path, json_data=body,
files=to_httpx_files(files), **options
   1259     )
-> 1260     return cast(ResponseT, self.request(cast_to, opts, stream=stream,
stream_cls=stream_cls))
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\openai\_base_client.py:937, in
SyncAPIClient.request(self, cast_to, options, remaining_retries, stream, stream_cls)
    928 def request(
    929     self,
    930     cast_to: Type[ResponseT],
   (...)
    935     stream_cls: type[_StreamT] | None = None,
    936 ) -> ResponseT | _StreamT:
--> 937     return self._request(
    938         cast_to=cast_to,
    939         options=options,
    940         stream=stream,
    941         stream_cls=stream_cls,
    942         remaining_retries=remaining_retries,
    943     )
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\openai\_base_client.py:997, in
SyncAPIClient._request(self, cast_to, options, remaining_retries, stream,
stream_cls)
    994 log.debug("Encountered Exception", exc_info=True)
    996 if retries > 0:
--> 997     return self._retry_request(
    998         input_options,
    999         cast_to,
   1000         retries,
   1001         stream=stream,
   1002         stream_cls=stream_cls,
   1003         response_headers=None,
   1004     )
   1006 log.debug("Raising connection error")
   1007 raise APIConnectionError(request=request) from err
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\openai\_base_client.py:1075, in
SyncAPIClient._retry_request(self, options, cast_to, remaining_retries,
response_headers, stream, stream_cls)
   1071 # In a synchronous context we are blocking the entire thread. Up to the
library user to run the client in a
   1072 # different thread if necessary.
   1073 time.sleep(timeout)
-> 1075 return self._request(
   1076     options=options,
   1077     cast_to=cast_to,
   1078     remaining_retries=remaining,
   1079     stream=stream,
   1080     stream_cls=stream_cls,
   1081 )
```


```
File ~\anaconda3\envs\agent\Lib\site-packages\openai\_base_client.py:997, in
SyncAPIClient._request(self, cast_to, options, remaining_retries, stream,
stream_cls)
    994 log.debug("Encountered Exception", exc_info=True)
    996 if retries > 0:
--> 997     return self._retry_request(
    998         input_options,
    999         cast_to,
   1000         retries,
   1001         stream=stream,
   1002         stream_cls=stream_cls,
   1003         response_headers=None,
   1004     )
   1006 log.debug("Raising connection error")
   1007 raise APIConnectionError(request=request) from err
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\openai\_base_client.py:1075, in
SyncAPIClient._retry_request(self, options, cast_to, remaining_retries,
response_headers, stream, stream_cls)
   1071 # In a synchronous context we are blocking the entire thread. Up to the
library user to run the client in a
   1072 # different thread if necessary.
   1073 time.sleep(timeout)
-> 1075 return self._request(
   1076     options=options,
   1077     cast_to=cast_to,
   1078     remaining_retries=remaining,
   1079     stream=stream,
   1080     stream_cls=stream_cls,
   1081 )
```

```
File ~\anaconda3\envs\agent\Lib\site-packages\openai\_base_client.py:1007, in
SyncAPIClient._request(self, cast_to, options, remaining_retries, stream,
stream_cls)
    997         return self._retry_request(
    998             input_options,
    999             cast_to,
   (...)
   1003             response_headers=None,
   1004         )
   1006     log.debug("Raising connection error")
-> 1007     raise APIConnectionError(request=request) from err
   1009 log.debug(
   1010     'HTTP Response: %s %s "%i %s" %s',
   1011     request.method,
   (...)
   1015     response.headers,
   1016 )
   1017 log.debug("request_id: %s", response.headers.get("x-request-id"))
```

```
APIConnectionError: Connection error.
```

当大模型处理完用户的输入，它会返回一个响应对象。此时我们便可以从响应体中大提取模型生成的消息内容。

```
response
```

```
ChatCompletion(id='chatcmpl-A6cQpOOt3XTBWZYDlr1Hv8tL6U7iX', choices=
[Choice(finish_reason='stop', index=0, logprobs=None,
message=ChatCompletionMessage(content='人工智能（Artificial Intelligence, AI）是计算机科学的一个分支，旨在模拟和实现人类智能的特征和功能。它涉及到创造能够执行通常需要人类智能的任务的计算机系统，包括学习、推理、问题解决、理解自然语言和感知等。人工智能可以分为以下几个主要领域：\n\n1. **机器学习（Machine Learning）**：这是人工智能的一个子领域，通过算法和统计模型使计算机能够从数据中学习并作出预测或决策，而无需明确编程。机器学习的关键在于利用大量数据训练模型，使其能够识别模式和规律。\n\n2. **深度学习（Deep Learning）**：深度学习是机器学习的一个特殊分支，使用类似于人脑神经网络的多层结构来处理和建模复杂数据。深度学习在图像识别、自然语言处理和语音识别等领域取得了显著进展。\n\n3. **自然语言处理（Natural Language Processing, NLP）**：这是人工智能的一个领域，旨在使计算机理解、生成和操作人类语言。NLP的应用包括语言翻译、情感分析、聊天机器人和语音助手等。\n\n4. **计算机视觉（Computer Vision）**：计算机视觉是使计算机能够“看”和理解图像和视频的领域。它涉及图像识别、物体检测和图像生成等任务，广泛应用于监控、自动驾驶和医疗成像等领域。\n\n5. **专家系统（Expert Systems）**：这些系统模拟人类专家的决策能力，依靠知识库和推理引擎来解决特定领域的问题。虽然在某些领域（如医疗诊断和金融）使用较多，但通常局限于特定任务。\n\n6. **智能机器人（Intelligent Robotics）**：机器人技术与人工智能的结合，赋予机器人完成复杂任务的能力，如自动化生产线上的操作、家庭助理、外科手术等。\n\n人工智能的应用范围非常广泛，包括金融、医疗、运输、教育、娱乐等多个领域。随着计算能力的增强和数据量的爆炸性增长，人工智能技术也在不断发展。尽管人工智能带来了许多便利和效率，但也伴随着一系列的伦理、隐私和安全问题，这些都需要在技术进步的同时加以关注和解决。\n\n未来，人工智能有可能在更广泛的领域内发挥重要作用，改变我们生活和工作的方式，但它的发展必须伴随着对其潜在影响的深入思考和审慎管理。', refusal=None, role='assistant', function_call=None, tool_calls=None))],
created=1726141063, model='gpt-4o-mini-2024-07-18', object='chat.completion',
service_tier=None, system_fingerprint='fp_483d39d857',
usage=CompletionUsage(completion_tokens=568, prompt_tokens=19, total_tokens=587))
```

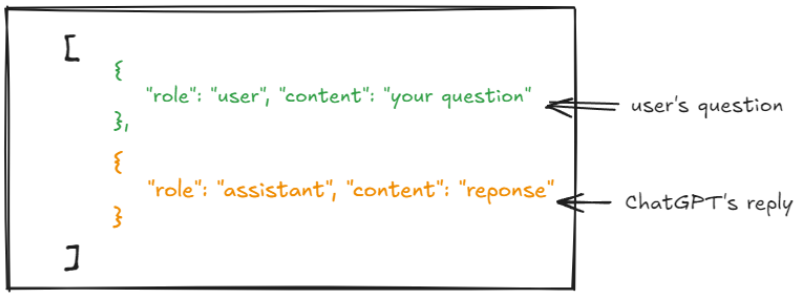
```
response.choices[0].message
```

```
ChatCompletionMessage(content='人工智能（Artificial Intelligence, AI）是计算机科学的一个分支，旨在模拟和实现人类智能的特征和功能。它涉及到创造能够执行通常需要人类智能的任务的计算机系统，包括学习、推理、问题解决、理解自然语言和感知等。人工智能可以分为以下几个主要领域：\n\n1. **机器学习（Machine Learning）**：这是人工智能的一个子领域，通过算法和统计模型使计算机能够从数据中学习并作出预测或决策，而无需明确编程。机器学习的关键在于利用大量数据训练模型，使其能够识别模式和规律。\n\n2. **深度学习（Deep Learning）**：深度学习是机器学习的一个特殊分支，使用类似于人脑神经网络的多层结构来处理和建模复杂数据。深度学习在图像识别、自然语言处理和语音识别等领域取得了显著进展。\n\n3. **自然语言处理（Natural Language Processing, NLP）**：这是人工智能的一个领域，旨在使计算机理解、生成和操作人类语言。NLP的应用包括语言翻译、情感分析、聊天机器人和语音助手等。\n\n4. **计算机视觉（Computer Vision）**：计算机视觉是使计算机能够“看”和理解图像和视频的领域。它涉及图像识别、物体检测和图像生成等任务，广泛应用于监控、自动驾驶和医疗成像等领域。\n\n5. **专家系统（Expert Systems）**：这些系统模拟人类专家的决策能力，依靠知识库和推理引擎来解决特定领域的问题。虽然在某些领域（如医疗诊断和金融）使用较多，但通常局限于特定任务。\n\n6. **智能机器人（Intelligent Robotics）**：机器人技术与人工智能的结合，赋予机器人完成复杂任务的能力，如自动化生产线上的操作、家庭助理、外科手术等。\n\n人工智能的应用范围非常广泛，包括金融、医疗、运输、教育、娱乐等多个领域。随着计算能力的增强和数据量的爆炸性增长，人工智能技术也在不断发展。尽管人工智能带来了许多便利和效率，但也伴随着一系列的伦理、隐私和安全问题，这些都需要在技术进步的同时加以关注和解决。\n\n未来，人工智能有可能在更广泛的领域内发挥重要作用，改变我们生活和工作的方式，但它的发展必须伴随着对其潜在影响的深入思考和审慎管理。',
refusal=None, role='assistant', function_call=None, tool_calls=None)
```

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

```
人工智能（Artificial Intelligence，AI）是计算机科学的一个分支，旨在模拟和实现人类智能的特征和功能。它涉及到创造能够执行通常需要人类智能的任务的计算机系统，包括学习、推理、问题解决、理解自然语言和感知等。人工智能可以分为以下几个主要领域：

1. **机器学习（Machine Learning）**：这是人工智能的一个子领域，通过算法和统计模型使计算机能够从数据中学习并作出预测或决策，而无需明确编程。机器学习的关键在于利用大量数据训练模型，使其能够识别模式和规律。

2. **深度学习（Deep Learning）**：深度学习是机器学习的一个特殊分支，使用类似于人脑神经网络的多层结构来处理和建模复杂数据。深度学习在图像识别、自然语言处理和语音识别等领域取得了显著进展。

3. **自然语言处理（Natural Language Processing，NLP）**：这是人工智能的一个领域，旨在使计算机理解、生成和操作人类语言。NLP的应用包括语言翻译、情感分析、聊天机器人和语音助手等。

4. **计算机视觉（Computer Vision）**：计算机视觉是使计算机能够“看”和理解图像和视频的领域。它涉及图像识别、物体检测和图像生成等任务，广泛应用于监控、自动驾驶和医疗成像等领域。

5. **专家系统（Expert Systems）**：这些系统模拟人类专家的决策能力，依靠知识库和推理引擎来解决特定领域的问题。虽然在某些领域（如医疗诊断和金融）使用较多，但通常局限于特定任务。

6. **智能机器人（Intelligent Robotics）**：机器人技术与人工智能的结合，赋予机器人完成复杂任务的能力，如自动化生产线上的操作、家庭助理、外科手术等。

人工智能的应用范围非常广泛，包括金融、医疗、运输、教育、娱乐等多个领域。随着计算能力的增强和数据量的爆炸性增长，人工智能技术也在不断发展。尽管人工智能带来了许多便利和效率，但也伴随着一系列的伦理、隐私和安全问题，这些都需要在技术进步的同时加以关注和解决。

未来，人工智能有可能在更广泛的领域内发挥重要作用，改变我们生活和工作的方式，但它的发展必须伴随着对其潜在影响的深入思考和审慎管理。
```

上述的交互过程如下图所示 👇



接下来，围绕这种一问一答的逻辑，我们就可以构建非常简单的While循环来生成一个持续对话的交互程序，代码如下所示：

```
from openai import OpenAI
client = OpenAI()
```

```
while True:
    prompt = input('\n用户提问：')
    if prompt == "退出":
        break  # 如果输入的是“退出”，则结束循环
    completion = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[{"role": "user", "content": prompt}],
        max_tokens=1024,
        temperature=0.8)
    message = completion.choices[0].message.content
    print(f"模型回复:{message}")
```

用户提问： 你好，请你介绍一下你自己

模型回复:你好！我是一个人工智能助手，旨在提供信息和回答问题。我可以帮助你解决各种问题，包括知识查询、学习辅导、生活建议等。如果你有任何具体的问题或需要了解的内容，请随时告诉我！

用户提问： 请你介绍一下什么是人工智能

模型回复:人工智能（Artificial Intelligence，简称AI）是计算机科学的一个分支，旨在使机器具备模拟人类智能的能力。它包括多个子领域，如机器学习、自然语言处理、计算机视觉、机器人技术等。人工智能的目标是开发能够执行通常需要人类智能的任务的系统，比如理解语言、识别图像、解决问题和进行决策。

人工智能可以分为两大类：

1. **窄域人工智能（Narrow AI）**：也称弱人工智能，指专门为了完成特定任务而设计的系统。例如，语音识别助手（如Siri、Alexa）、推荐系统和自动驾驶车辆等都属于窄域人工智能。

2. **通用人工智能（General AI）**：也称强人工智能，指具有人类水平的智能，能够理解、学习和应用知识以解决各种问题。当前，通用人工智能仍然是一个理论概念，尚未实现。

人工智能的应用非常广泛，包括医疗诊断、金融分析、智能客服、自动化生产等领域。随着技术的发展，人工智能正逐渐改变各个行业的运作方式，并对社会产生深远的影响。

用户提问： 你好，我叫木羽，很高兴认识你

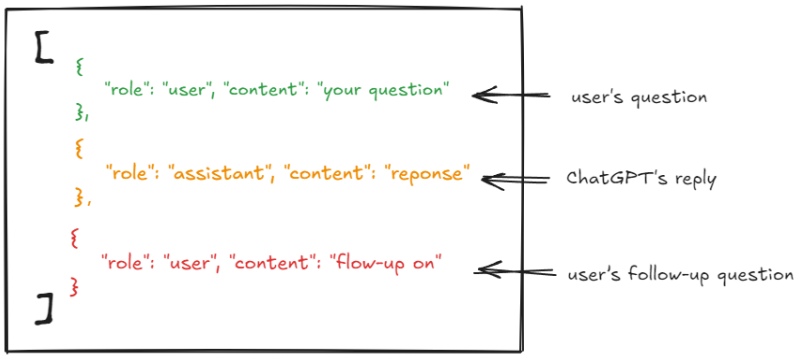
模型回复:你好，木羽！很高兴认识你！有什么我可以帮助你的吗？

用户提问： 请问我叫什么？

```
模型回复:抱歉，我无法知道您的名字。如果您愿意，可以告诉我您的名字。
```

```
用户提问：  退出
```

如上所示的例子：正如我们之前所讨论的，单个大模型并不具备记忆和状态跟踪的能力，它仅将每个输入视为一个独立的事件来处理。这在多轮对话中带来的实际的问题就是：**大模型无法根据先前的对话内容进行综合性的回应。**以这个例子来说，我在一轮对话中告诉模型我的名字是木羽，然后在下一轮询问我的名字，模型却无法记起这一信息。这种缺乏连贯性的表现与我们日常的交互方式明显不符。因此，为了与大模型进行有意义的对话，我们需要做的是：将对话记录不断地反馈给大模型，也就是如下所述的形式 👇



如上图所示，我们需要将每次的对话和大模型的回复， 不断地追加到传递给 API 的消息字典列表中。这样，大模型就能够知道我们之前提出的问题及其提供的答复。所以代码更新如下：

```python
from openai import OpenAI
client = OpenAI()

# 创建一个会话列表来不断地追加历史对话纪律
messages = []

while True:
    prompt = input('\n用户提问：  ')
    if prompt == "退出":
        break  # 如果输入的是“退出”，则结束循环
    messages.append(
        {
            'role':'user',
            'content':prompt
        })
    completion = client.chat.completions.create(
        model="gpt-4o-mini",
        messages = messages)

    response = completion.choices[0].message.content
    print(f"模型回复:{response}")
    messages.append(
        {
```

```
            'role':'assistant',
            'content':response
        })
```

用户提问： 你好，请你介绍一下你自己

模型回复:你好！我是一个人工智能助手，旨在回答你的问题和提供信息。我可以帮助你了解各种主题，比如科技、历史、文化、语言等。如果你有任何具体的问题或需要帮助的地方，请随时告诉我！

用户提问： 我叫木羽，很高兴认识你。

模型回复:你好，木羽！很高兴认识你！如果你有什么问题或想讨论的话题，请随时告诉我。我很乐意帮忙！

用户提问： 你知道我叫什么吗？

模型回复:当然，你告诉我你叫木羽。如果你希望我用其他的称呼或者有其他的问题，请随时告诉我！

用户提问： 退出

```
messages
```

```
[{'role': 'user', 'content': '你好，请你介绍一下你自己'},
 {'role': 'assistant',
  'content': '你好！我是一个人工智能助手，旨在回答你的问题和提供信息。我可以帮助你了解各种主题，比如科技、历史、文化、语言等。如果你有任何具体的问题或需要帮助的地方，请随时告诉我！'},
 {'role': 'user', 'content': '我叫木羽，很高兴认识你。'},
 {'role': 'assistant',
  'content': '你好，木羽！很高兴认识你！如果你有什么问题或想讨论的话题，请随时告诉我。我很乐意帮忙！'},
 {'role': 'user', 'content': '你知道我叫什么吗？'},
 {'role': 'assistant', 'content': '当然，你告诉我你叫木羽。如果你希望我用其他的称呼或者有其他的问题，请随时告诉我！'}]
```

如上案例所示，**大模型可以通过不断追加会话消息列表来模拟记忆功能，这是聊天机器人应用的一种常见实践。**借助于大模型的处理能力和这种短期记忆机制，构造出来的问答系统能够与用户进行交互并帮助解答问题。当然，为了解决大模型知识的局限性问题，我们经常采用RAG技术来增强其信息检索能力。但需要明确的是，**尽管结合了RAG技术，LLM和RAG的组合主要还是用于知识问答。**因此，如果没有其他技术的结合，这种系统还是属于聊天机器人的范畴。所以，如果大家需求仅限于让大模型回答问题或者基于本地数据提供回复，那么只需要大模型和RAG技术栈，并不涉及任何更复杂的代理（Agent）相关技术。

> 关于LLMs + RAG 复杂聊天机器人的实现，大家可以参考我们课程的内容：《大模型RAG技术企业项目实战》 Week1 ~ Week3 的内容。

让我们先回顾一下聊天机器人、人工智能助手和人工智能代理之间的区别，以便您清楚地了解：

聊天机器人是那些基本上被设计用来进行对话、回答小问题或提供客户服务帮助的小朋友。有些非常简单，有些则通过机器学习更先进，但它们的范围是有限的。

然后我们还有人工智能助手，这完全是另一回事了。这些小工具可以处理大量不同的任务，例如搜索信息、生成内容，甚至管理家中的智能设备。他们使用人工智能来了解上下文并了解您的偏好，就像您的小私人助理一样！

但真正成为炸弹的是人工智能特工。这些才是真正的交易。它们旨在自主执行各种任务、做出决策、解决问题，最重要的是与现实世界互动！想象一下他们为您预订航班、购物，甚至谈判合同。

1.1 聊天机器人

在你应该已经听说过ChatGPT（ Chat Generative Pre-trained Transformer ）——一个由OpenAI开发的聊天机器人。 ChatGPT 于 2022 年 11 月推出，建立在 OpenAI 的 GPT-3（特别是GPT-3.5 ）大型语言模型 (LLMs) 系列之上，并使用监督学习和强化学习技术进行微调。此后，ChatGPT 引发了人工智能热潮，推出仅两个月就拥有 1 亿用户。

作为一名开发人员，我没有在https://chat.openai.com/chat上尝试 ChatGPT，而是发现能够将其集成到我的应用程序中更令人兴奋。因此，在本文中，我将向您展示如何使用 Python 与 ChatGPT 进行交互。


```
from openai import OpenAI
client = OpenAI()

response = client.chat.completions.create(
  model="gpt-4o-mini",
  messages=[
    {"role": "system", "content": "你是一位乐于助人的智能助理"},
    {"role": "user", "content": "帮我介绍一下什么是人工智能"},
  ]
)
```

```
response.choices[0].message
```

```
ChatCompletionMessage(content='人工智能（Artificial Intelligence，简称AI）是计算机科学的一个分支，旨在创建能够执行通常需要人类智能才能完成的任务的系统和程序。这些任务包括学习、推理、问题解决、感知、理解自然语言以及决策等。\n\n人工智能可以分为两大类：\n\n1. **弱人工智能（Narrow AI）**：也称为狭义人工智能，指的是为特定任务设计的系统，例如语音识别、图像识别、推荐系统等。这类AI在特定领域内表现出色，但无法超出其编程范围。\n\n2. **强人工智能（General AI）**：指的是一种能够理解、学习和应用智能于任何领域的人工智能系统。这种技能力尚处于理论研究阶段，目前尚无实例。\n\n人工智能的主要技术包括：\n\n- **机器学习（Machine Learning）**：利用数据和算法使计算机系统自动学习并改进其性能。\n- **深度学习（Deep Learning）**：机器学习的一个子集，使用神经网络模型来处理大量数据，以达到更高的学习效果。\n- **自然语言处理（Natural Language Processing，NLP）**：使计算机能够理解和解释人类语言的内容、情感和意图。\n- **计算机视觉（Computer Vision）**：使计算机能够从图像或视频中识别和处理视觉信息。\n\n人工智能应用广泛，涵盖了各个领域，如医疗、金融、交通、教育、娱乐等，为我们的生活带来了巨大的便利和变化。随着技术的发展，人工智能的潜力仍在不断被挖掘。', refusal=None, role='assistant', function_call=None, tool_calls=None)
```

```
```

要发送到 ChatGPT 的问题必须包含在字典列表中。例如，上述问题按照以下格式发送到ChatGPT：

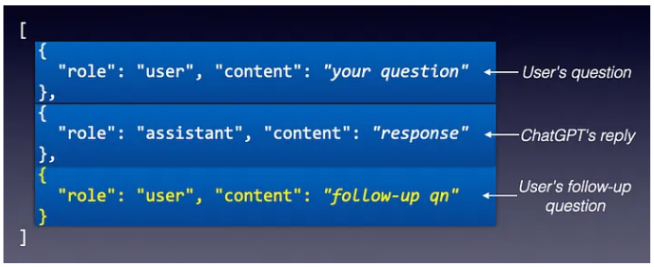
```
while True:
    prompt = input('\n提出一个问题：')
    if prompt == "退出":
        break  # 如果输入的是“退出”，则结束循环
    completion = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[{"role": "user", "content": prompt}],
        max_tokens=1024,
        temperature=0.8)
    message = completion.choices[0].message.content
    print(message)
```

ChatGPT 不记得您之前的问题。因此，为了与它进行有意义的对话，您需要将对话反馈给 API。还记得需要传递给 API 的字典列表吗？

要将之前的对话反馈给 ChatGPT，您首先将 ChatGPT 的回复附加到列表中：

Then, you append your follow-up question:

然后，您附加您的后续问题：



这样，ChatGPT 就能够知道您之前提出的问题及其提供的答复。以下是更新后的代码，允许用户与 ChatGPT 进行有意义的对话：

```
messages = []

while True:
    prompt = input('\n提出一个问题：  ')
    if prompt == "退出":
        break  # 如果输入的是“退出”，则结束循环
    messages.append(
        {
            'role':'user',
            'content':prompt
        })
    completion = client.chat.completions.create(
        model="gpt-4o-mini",
        messages = messages)

    response = completion.choices[0].message.content
    print(response)
    messages.append(
        {
            'role':'assistant',
            'content':response
        })
```

存在的问题是： 输出 无法给到下游！！！！！！！ 要解决的根本两点问题就是：

1. 如何让 大模型产生结构化的输出
2. 如何让 大模型在一次对话中，自动执行多个步骤。

这就提出了 函数调用功能。为什么要这么说，我们来看下面的这个例子：

2. 人工智能助手

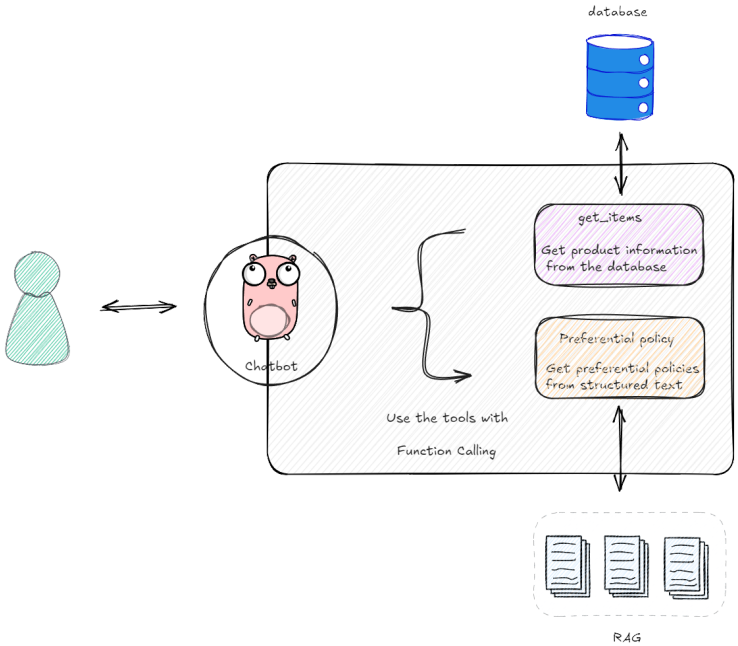
第二类智能应用是人工智能助手。这类应用不仅能生成内容，还具备搜索信息、管理设备等能力。可以理解上下文和用户偏好，它们表现得就像个人定制的私人助理，能够帮助我们实际的完成一些需要人工接入的既定工作。起到的**主要作用是替代手工执行特定的任务。**比如这样的一个智能客服助手的案例：电商平台上，用户经常有关于商品信息查询和优惠政策提问的需求。一个智能客服系统可以自动化这些流程，提高响应速度和准确性，这种场景下智能助手具体要做的是：

- 功能：用户询问特定商品时，如"你们家有没有XX商品？"智能助手可以自动在商品数据库中查询并回复是否有货、价格等信息。
- 实现：大模型接收到商品查询请求后，从内置的商品数据表中检索信息，并以文本形式向用户提供反馈。
- 功能：当用户提出关于购买时某些商品的实时优惠政策等问题时，智能助手能从预设的非结构化文本中提取相关信息并回答。
- 实现：模型利用预先设定的关键词和语境从非结构化数据（如优惠政策的手册或厂家的API返回的文本）中提取答案。

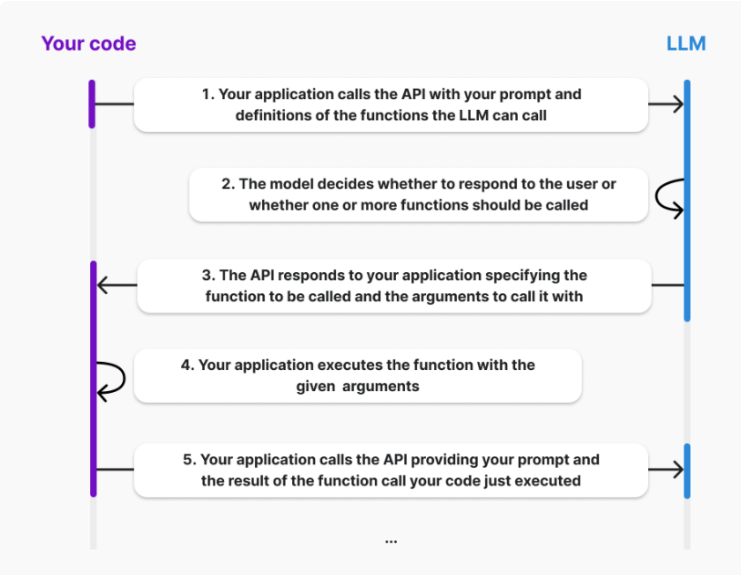
这样复杂一点的场景在第一个聊天机器人的案例中很难实现，其主要存在的问题是：**类似智能客服的整个问答流程涉及两个阶段，第一阶段是接收用户的输入，去查询数据库。第二阶段是根据数据库返回的结果，结合用户的问题，生成最终的答案。而这样的阶段性操作，其根本要解决的两点问题是：**

1. 如何让 大模型产生结构化的输出
2. 如何让 大模型在一次对话中，自动执行多个步骤。

由此才有了 函数调用（Function Calling）的技术应用。为什么要这么说，我们来看下面的这个例子：当用户询问 `你们家都卖哪些产品` 时，人工智能助手需要在它可以生成对用户的响应之前，先从内部系统获取最新的商家数据，就变成了下面这样：



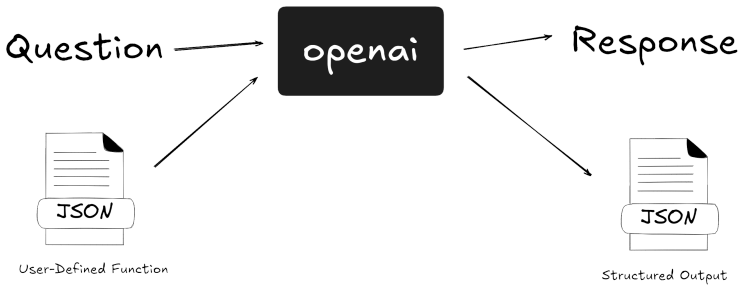
人工智能助手的核心技术就是函数调用，因为它是借助函数调用的功能，通过提示（Prompt）告诉大模型确定应该调用哪些函数来满足用户的需求。其生命周期如下图所示：



如上图所示，描述了使用大语言模型（LLM）通过API进行功能调用的完整生命周期。具体过程如下：

1. **API调用**：应用程序向API发送调用请求，附带具体的提示信息和可供LLM调用的函数定义。
2. **模型决策**：LLM评估接收到的输入，并决定是否直接回应用户，或是需要调用一个或多个外部函数以提供更合适的回答。
3. **API响应**：API向应用程序返回响应，指明需要执行的函数以及执行这些函数所需的参数。
4. **执行函数**：应用程序根据API的指示执行指定的函数，使用提供的参数。
5. **调用结果处理**：完成函数执行后，应用程序再次调用API，传递先前的提示信息和函数执行的结果，以便LLM可以利用这些新数据生成最终的用户响应。

可以看到，当涉及到函数调用的时候，其生命周期相较于直接让大模型生成响应回复，需要额外增加的操作是：其一：我们要给它配备对应的外部工具；其二：我们要执行外部工具，得到外部工具的返回结果；其三：把工具的执行结果拼接到用户的原始输入中，从而得到最终的回复。其过程如下图所示：



为了帮助大家更好的理解通过函数调用（function calling）实现人工智能助手的过程，我们实操一个智能客服的案例来详细介绍中间环节的详细处理流程。

首先，我们先构造一个商家后台商品信息的模拟数据。这里使用 `Python` 连接 `sqlite3` 库进行存储。代码如下所示：

```python
import sqlite3

def create_and_populate_database():
    # 连接到本地数据库文件
    conn = sqlite3.connect('SportsEquipment.db')  # 指定文件名来保存数据库
```

```
    cursor = conn.cursor()

    # 检查是否存在名为 'products' 的表，如果不存在则创建
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND
name='products';")
    if cursor.fetchone() is None:
        # 创建表
        cursor.execute('''
        CREATE TABLE products (
            product_id TEXT,
            product_name TEXT,
            description TEXT,
            specifications TEXT,
            usage TEXT,
            brand TEXT,
            price REAL,
            stock_quantity INTEGER
        )
        ''')
        # 数据列表，用于插入表中
        products = [
            ('001', '足球', '高品质职业比赛用球，符合国际标准', '圆形，直径22 cm', '职业比赛、
学校体育课', '耐克', 120, 50),
            ('002', '羽毛球拍', '轻量级，适合初中级选手，提供优秀的击球感受', '碳纤维材质，重量
85 g', '业余比赛、家庭娱乐', '尤尼克斯', 300, 30),
            ('003', '篮球', '室内外可用，耐磨耐用，适合各种天气条件', '皮质，标准7号球', '学
校、社区运动场', '斯伯丁', 200, 40),
            ('004', '跑步鞋', '适合长距离跑步，舒适透气，提供良好的足弓支撑', '多种尺码，透气网
布', '长跑、日常训练', '阿迪达斯', 500, 20),
            ('005', '瑜伽垫', '防滑材料，厚度适中，易于携带和清洗', '长180cm，宽60cm，厚5mm',
'瑜伽、普拉提', '曼达卡', 150, 25),
            ('006', '速干运动衫', '吸汗快干，适合各种户外运动，持久舒适', 'S/M/L/XL，多色可
选', '运动、徒步、旅游', '诺斯脸', 180, 60),
            ('007', '电子计步器', '精确计步，带心率监测功能，蓝牙连接手机应用', '可充电，续航7
天', '日常健康管理、运动', 'Fitbit', 250, 15),
            ('008', '乒乓球拍套装', '包括两只拍子和三个球，适合家庭娱乐和业余训练', '标准尺寸，
拍面防滑处理', '家庭、社区', '双鱼', 160, 35),
            ('009', '健身手套', '抗滑耐磨，保护手部，适合各种健身活动', '多种尺码，通风设计',
'健身房、户外运动', 'Under Armour', 120, 50),
            ('010', '膝盖护具', '减少运动伤害，提供良好的支撑和保护，适合篮球和足球运动', '弹性
织物，可调节紧度', '篮球、足球及其他运动', '麦克戴维', 220, 40)
        ]

        # 插入数据到表中
        cursor.executemany('''
        INSERT INTO products (product_id, product_name, description, specifications,
usage, brand, price, stock_quantity)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?)
        ''', products)

        # 提交更改以确保数据被保存在文件中
        conn.commit()
```

```
    # 检索并打印所有记录以验证插入
    cursor.execute('SELECT * FROM products')
    all_rows = cursor.fetchall()

    conn.close()  # 关闭连接以释放资源

    return all_rows

# 执行函数并打印结果
create_and_populate_database()
```

```
[('001',
  '足球',
  '高品质职业比赛用球，符合国际标准',
  '圆形，直径22 cm',
  '职业比赛、学校体育课',
  '耐克',
  120.0,
  50),
 ('002',
  '羽毛球拍',
  '轻量级，适合初中级选手，提供优秀的击球感受',
  '碳纤维材质，重量85 g',
  '业余比赛、家庭娱乐',
  '尤尼克斯',
  300.0,
  30),
 ('003',
  '篮球',
  '室内外可用，耐磨耐用，适合各种天气条件',
  '皮质，标准7号球',
  '学校、社区运动场',
  '斯伯丁',
  200.0,
  40),
 ('004',
  '跑步鞋',
  '适合长距离跑步，舒适透气，提供良好的足弓支撑',
  '多种尺码，透气网布',
  '长跑、日常训练',
  '阿迪达斯',
  500.0,
  20),
 ('005',
  '瑜伽垫',
  '防滑材料，厚度适中，易于携带和清洗',
  '长180cm，宽60cm，厚5mm',
  '瑜伽、普拉提',
```

```
  '曼达卡',
  150.0,
  25),
 ('006',
  '速干运动衫',
  '吸汗快干，适合各种户外运动，持久舒适',
  'S/M/L/XL，多色可选',
  '运动、徒步、旅游',
  '诺斯脸',
  180.0,
  60),
 ('007',
  '电子计步器',
  '精确计步，带心率监测功能，蓝牙连接手机应用',
  '可充电，续航7天',
  '日常健康管理、运动',
  'Fitbit',
  250.0,
  15),
 ('008',
  '乒乓球拍套装',
  '包括两只拍子和三个球，适合家庭娱乐和业余训练',
  '标准尺寸，拍面防滑处理',
  '家庭、社区',
  '双鱼',
  160.0,
  35),
 ('009',
  '健身手套',
  '抗滑耐磨，保护手部，适合各种健身活动',
  '多种尺码，通风设计',
  '健身房、户外运动',
  'Under Armour',
  120.0,
  50),
 ('010',
  '膝盖护具',
  '减少运动伤害，提供良好的支撑和保护，适合篮球和足球运动',
  '弹性织物，可调节紧度',
  '篮球、足球及其他运动',
  '麦克戴维',
  220.0,
  40)]
```

执行完上述函数后，会在本地的当前路径下生成一个`SportsEquipment.db`文件，这个表有多个字段，包括产品ID、名称、描述、规格、用途、品牌、价格和库存数量。

- **第一步：构建大模型能够调用的函数**

对于电商智能客服这个场景，如果我们希望它能够在回答用户的问题之前，先去查询上述创建的商品后台数据库，那么就需要出创建一个函数来做这件事。比如用户输入：`你们家都有什么球在卖`？ 那么理想的状态是这个函数可以根据 `球` 这个关键字向后台数据库执行查询操作。所以我们构建了如下所示的 `query_by_product_name` 函数：

测试一下函数功能：

```
import sqlite3

def query_by_product_name(product_name):
    # 连接 SQLite 数据库
    conn = sqlite3.connect('SportsEquipment.db')
    cursor = conn.cursor()

    # 使用SQL查询按名称查找产品。'%'符号允许部分匹配。
    cursor.execute("SELECT * FROM products WHERE product_name LIKE ?", ('%' +
product_name + '%',))

    # 获取所有查询到的数据
    rows = cursor.fetchall()

    # 关闭连接
    conn.close()

    return rows
```

```
matching_products = query_by_product_name('球')

print("Matching Products:")

for product in matching_products:
    print(product)
```

```
Matching Products:
('001', '足球', '高品质职业比赛用球，符合国际标准', '圆形，直径22 cm', '职业比赛、学校体育课',
'耐克', 120.0, 50)
('002', '羽毛球拍', '轻量级，适合初中级选手，提供优秀的击球感受', '碳纤维材质，重量85 g', '业余
比赛、家庭娱乐', '尤尼克斯', 300.0, 30)
('003', '篮球', '室内外可用，耐磨耐用，适合各种天气条件', '皮质，标准7号球', '学校、社区运动场',
'斯伯丁', 200.0, 40)
('008', '乒乓球拍套装', '包括两只拍子和三个球，适合家庭娱乐和业余训练', '标准尺寸，拍面防滑处理',
'家庭、社区', '双鱼', 160.0, 35)
```

- **第 二 步：向大模型描述这个函数，以便大模型知道如何调用它，以及在什么情况下需要调用它**

根据场景设计，我们能够明确的是我们允许大模型调用什么函数，但是大模型什么时候去调用它，如何调用它，则需要我们进一步创建一个“函数定义”来向模型描述该函数。该定义描述了该函数的作用（以及可能何时调用该函数）以及调用该函数需要哪些参数。而函数定义的 `parameters` 部分应使用 `JSON` 架构进行描述。一旦模型生成函数调用，它将使用此信息根据提供的要求生成参数。

> 对于Json Schema 结构的详细介绍，大家可以参考我们之前课程的内容，在《大模型技术实战课》的 Ch 8 课程。

```
{
    "name": "query_by_product_name",
    "description": "Query the database to retrieve a list of products that match or contain the specified product name. This function can be used to assist customers in finding products by name via an online platform or customer support interface.",   # 查询数据库以检索匹配或包含指定产品名称的产品列表。此功能可用于帮助客户通过在线平台或客户支持界面按名称查找产品。
    "parameters": {
        "type": "object",
        "properties": {
            "product_name": {
                "type": "string",
                "description": "The name of the product to search for. The search is case-insensitive and allows partial matches."
            }
        },
        "required": ["product_name"]
    }
}
```

这段`Json Schema`描述的关键要素如下：

1. **name 字段**：明确指示大模型需要调用的具体函数名称。
2. **description 字段**：向大模型说明在哪些用户需求下应当触发这个函数，即在何种场景下调用。
3. **parameters**：详细描述函数所接受的参数，使大模型能够从类似“你家都卖什么球”的自然语言查询中提取出“球”作为关键搜索词。其中的`required`字段指明哪些参数是必需的，确保函数能在缺少这些参数时提示或防止执行。

这种表述不仅清晰地阐明了各字段的作用，也帮助理解如何结构化这些信息以便模型能够有效地解析和响应用户的具体需求。

- **第 三 步：将函数定义作为可用工具以及消息传递给大模型**

接下来，当涉及到函数调用的时候，我们需要在调用聊天完成 API 时，额外的传递一个`tools`参数，以告知大模型：你在当前的会话过程中，可以调用`query_by_product_name`参数。

```
tools = [
    {
        "type": "function",
        "function": {
            "name": "query_by_product_name",
            "description": "Query the database to retrieve a list of products that match or contain the specified product name. This function can be used to assist customers in finding products by name via an online platform or customer support interface.",
            "parameters": {
                "type": "object",
                "properties": {
```

```
                    "product_name": {
                        "type": "string",
                        "description": "The name of the product to search for. The
search is case-insensitive and allows partial matches."
                    }
                },
                "required": ["product_name"]
            }
        }

    }
]
```

```
messages = [
    {"role": "user", "content": "老板，在吗"}
]
```

注意：这里通过`tools`参数进行工具传递。

```
from openai import OpenAI
client = OpenAI()

response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=messages,
    tools=tools,  # 这里是添加
)
```

- **第 四 步： 接收并处理大模型响应**

这里需要注意的是：**尽管传递了`tools`工具，但并不意味着大模型一定会进行调用**，我们之前提到了：对每个工具我们会提供`description`描述，它的意义在于如果大模型判断用户的输入并没有这方面的意图，则会像`聊天机器人`应用一样直接进行响应。
如果模型不生成函数调用，则响应将包含以聊天完成的正常方式直接回复用户。例如，在这种情况下 chat_response.choices[0].message 可能包含：

```
response
```

```
ChatCompletion(id='chatcmpl-A6dCCr7pmOgJumt9IjI1xlVRQ6wmo', choices=
[Choice(finish_reason='stop', index=0, logprobs=None,
message=ChatCompletionMessage(content='在的，有什么我可以帮助您的吗？', refusal=None,
role='assistant', function_call=None, tool_calls=None))], created=1726144000,
model='gpt-4o-mini-2024-07-18', object='chat.completion', service_tier=None,
system_fingerprint='fp_483d39d857', usage=CompletionUsage(completion_tokens=11,
prompt_tokens=104, total_tokens=115))
```

```
response.choices[0].message.content
```

```
'在的，有什么我可以帮助您的吗？'
```

而假设用户的问题中提及了关于商品的提问，才会触发工具调用的流程：

```
messages = [
    {"role": "user", "content": "你好，你家都卖什么球？"}
]
```

```
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=messages,
    tools=tools,
)
```

如果模型生成函数调用，它将生成调用的参数（基于我们提供的parameters定义）。其响应体如下所示：

```
response
```

```
ChatCompletion(id='chatcmpl-A6dF0yyo1SYXN1hTs8wje4lWOH8M6', choices=
[Choice(finish_reason='tool_calls', index=0, logprobs=None,
message=ChatCompletionMessage(content=None, refusal=None, role='assistant',
function_call=None, tool_calls=
[ChatCompletionMessageToolCall(id='call_Npn8AYJHvGjXAdUFklksZ6UO',
function=Function(arguments='{"product_name":"球"}', name='query_by_product_name'),
type='function')]))], created=1726144174, model='gpt-4o-mini-2024-07-18',
object='chat.completion', service_tier=None, system_fingerprint='fp_483d39d857',
usage=CompletionUsage(completion_tokens=17, prompt_tokens=109, total_tokens=126))
```

这里的区别是：当触发函数调用，content字段的对应的值将会是None，而tool_calls字段中的内容，将会按照我们对`query_by_product_name`的Json Schema描述中`required`字段的要求来返回值。如上所示为：arguments='{"product_name":"球"}', name='query_by_product_name')。当拿到这样的参数后，按照我们期望的逻辑应该是把`球`作为关键词，执行数据库查询，拿到商品的详细信息，如下图所示：

```
tool_call = response.choices[0].message.tool_calls[0]
tool_call
```

```
ChatCompletionMessageToolCall(id='call_Npn8AYJHvGjXAdUFklksZ6UO',
function=Function(arguments='{"product_name":"球"}', name='query_by_product_name'),
type='function')
```

```
arguments = json.loads(tool_call.function.arguments)
arguments
```

```
{'product_name': '球'}
```

```
product_name = arguments.get('product_name')
product_name
```

```
'球'
```

```
final_res = query_by_product_name(product_name)
final_res
```

```
[('001',
  '足球',
  '高品质职业比赛用球，符合国际标准',
  '圆形，直径22 cm',
  '职业比赛、学校体育课',
  '耐克',
  120.0,
  50),
 ('002',
  '羽毛球拍',
  '轻量级，适合初中级选手，提供优秀的击球感受',
  '碳纤维材质，重量85 g',
  '业余比赛、家庭娱乐',
  '尤尼克斯',
  300.0,
  30),
 ('003',
```

```
  '篮球',
  '室内外可用，耐磨耐用，适合各种天气条件',
  '皮质，标准7号球',
  '学校、社区运动场',
  '斯伯丁',
  200.0,
  40),
 ('008',
  '乒乓球拍套装',
  '包括两只拍子和三个球，适合家庭娱乐和业余训练',
  '标准尺寸，拍面防滑处理',
  '家庭、社区',
  '双鱼',
  160.0,
  35)]
```

- **第 五 步：将函数调用结果提供回大模型**

经过上面的手动流程，现在我们已经在本地执行了函数调用，我们需要将此函数调用的结果提供回聊天 API，以便大模型可以生成用户应该看到的实际响应。参考`聊天机器人`案例中构建大模型记忆的过程，我们只需要维护一个 message 列表，把函数调用的信息追加进去即可。整个过程如下所示：

```
response
```

```
ChatCompletion(id='chatcmpl-A6dF0yyo1SYXN1hTs8wje4lWOH8M6', choices=
[Choice(finish_reason='tool_calls', index=0, logprobs=None,
message=ChatCompletionMessage(content=None, refusal=None, role='assistant',
function_call=None, tool_calls=
[ChatCompletionMessageToolCall(id='call_Npn8AYJHvGjXAdUFklksZ6UO',
function=Function(arguments='{"product_name":"球"}', name='query_by_product_name'),
type='function')]))], created=1726144174, model='gpt-4o-mini-2024-07-18',
object='chat.completion', service_tier=None, system_fingerprint='fp_483d39d857',
usage=CompletionUsage(completion_tokens=17, prompt_tokens=109, total_tokens=126))
```

这里我们需要在使用`json.dumps()`添加 ensure_ascii=False 参数。这样，JSON 库就会保留中文字符而不是将它们转换为 Unicode 转义序列，如下所示：

```
content = json.dumps({
    "product_name": product_name,
    "query_by_product_name":final_res
}, ensure_ascii=False),
```

```
content
```

```
('{"product_name": "球", "query_by_product_name": [["001", "足球", "高品质职业比赛用球，符合国际标准", "圆形，直径22 cm", "职业比赛、学校体育课", "耐克", 120.0, 50], ["002", "羽毛球拍", "轻量级，适合初中级选手，提供优秀的击球感受", "碳纤维材质，重量85 g", "业余比赛、家庭娱乐", "尤尼克斯", 300.0, 30], ["003", "篮球", "室内外可用，耐磨耐用，适合各种天气条件", "皮质，标准7号球", "学校、社区运动场", "斯伯丁", 200.0, 40], ["008", "乒乓球拍套装", "包括两只拍子和三个球，适合家庭娱乐和业余训练", "标准尺寸，拍面防滑处理", "家庭、社区", "双鱼", 160.0, 35]]}',)
```

```
response.choices[0].message.tool_calls[0].id
```

```
'call_Npn8AYJHvGjXAdUFklksZ6U0'
```

这里有一点需要注意的是：除了像常规的会话信息提供 'role' 和 'content'外，函数调用产生的信息还需要传递`tool_call_id`字段：

```
# 创建 function calling 结果的消息历史
function_call_result_message = {"role": "tool", "content": str(content),
"tool_call_id": response.choices[0].message.tool_calls[0].id}
```

```
function_call_result_message
```

```
{'role': 'tool',
 'content': '(\'{"product_name": "球", "query_by_product_name": [["001", "足球", "高品质职业比赛用球，符合国际标准", "圆形，直径22 cm", "职业比赛、学校体育课", "耐克", 120.0, 50], ["002", "羽毛球拍", "轻量级，适合初中级选手，提供优秀的击球感受", "碳纤维材质，重量85 g", "业余比赛、家庭娱乐", "尤尼克斯", 300.0, 30], ["003", "篮球", "室内外可用，耐磨耐用，适合各种天气条件", "皮质，标准7号球", "学校、社区运动场", "斯伯丁", 200.0, 40], ["008", "乒乓球拍套装", "包括两只拍子和三个球，适合家庭娱乐和业余训练", "标准尺寸，拍面防滑处理", "家庭、社区", "双鱼", 160.0, 35]]}\',)',
 'tool_call_id': 'call_Npn8AYJHvGjXAdUFklksZ6U0'}
```

同时，必须把 response.choices[0].message 添加进来，因为标记为 'tool' 角色的消息都必须是对之前的 'tool_calls' 角色消息的响应。这是因为在使用API或系统内部消息传递机制时，其中一个工具（如某种类型的API调用）必须基于之前的调用结果来执行。

```
response.choices[0].message
```

```
ChatCompletionMessage(content=None, refusal=None, role='assistant',
function_call=None, tool_calls=
[ChatCompletionMessageToolCall(id='call_Npn8AYJHvGjXAdUFklksZ6U0',
function=Function(arguments='{"product_name":"球"}', name='query_by_product_name'),
type='function')])
```

构建完整的Messages：

```
messages = [
    {"role": "user", "content": "你好，你家都卖什么球？"},
    response.choices[0].message,
    function_call_result_message
]
```

```
messages
```

```
[{'role': 'user', 'content': '你好，你家都卖什么球？'},
 ChatCompletionMessage(content=None, refusal=None, role='assistant',
function_call=None, tool_calls=
[ChatCompletionMessageToolCall(id='call_Npn8AYJHvGjXAdUFklksZ6U0',
function=Function(arguments='{"product_name":"球"}', name='query_by_product_name'),
type='function')]),
 {'role': 'tool',
  'content': '(\'{"product_name": "球", "query_by_product_name": [["001", "足球", "高品质职业比赛用球，符合国际标准", "圆形，直径22 cm", "职业比赛、学校体育课", "耐克", 120.0, 50], ["002", "羽毛球拍", "轻量级，适合初中级选手，提供优秀的击球感受", "碳纤维材质，重量85 g", "业余比赛、家庭娱乐", "尤尼克斯", 300.0, 30], ["003", "篮球", "室内外可用，耐磨耐用，适合各种天气条件", "皮质，标准7号球", "学校、社区运动场", "斯伯丁", 200.0, 40], ["008", "乒乓球拍套装", "包括两只拍子和三个球，适合家庭娱乐和业余训练", "标准尺寸，拍面防滑处理", "家庭、社区", "双鱼", 160.0, 35]]}\',)',
  'tool_call_id': 'call_Npn8AYJHvGjXAdUFklksZ6U0'}]
```

再次进行调用：

```
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=messages,
)
```

```
# 打印结果
print(response.choices[0].message.content)
```

```
我们店里销售以下几种球：

1. **足球**
   - 描述：高品质职业比赛用球，符合国际标准
   - 规格：圆形，直径22 cm
   - 适用场景：职业比赛、学校体育课
   - 品牌：耐克
   - 价格：120.0元
   - 库存：50

2. **篮球**
   - 描述：室内外可用，耐磨耐用，适合各种天气条件
   - 规格：皮质，标准7号球
   - 适用场景：学校、社区运动场
   - 品牌：斯伯丁
   - 价格：200.0元
   - 库存：40

3. **羽毛球拍**（附带羽毛球）
   - 描述：轻量级，适合初中级选手，提供优秀的击球感受
   - 规格：碳纤维材质，重量85 g
   - 适用场景：业余比赛、家庭娱乐
   - 品牌：尤尼克斯
   - 价格：300.0元
   - 库存：30

4. **乒乓球拍套装**（包括两只拍子和三个球）
   - 描述：适合家庭娱乐和业余训练
   - 规格：标准尺寸，拍面防滑处理
   - 适用场景：家庭、社区
   - 品牌：双鱼
   - 价格：160.0元
   - 库存：35

如果你对某个球感兴趣，欢迎询问！
```

接下来，我们把这个能够调用外部工具的助手，再次以`聊天机器人`的形式来实现。

```
available_functions = {"query_by_product_name": query_by_product_name}
```

```
messages = [
    {"role": "user", "content": "你好，你家都卖什么球？"}
]
```

```
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=messages,
    tools=tools,
)
```

```
response.choices[0]
```

```
Choice(finish_reason='tool_calls', index=0, logprobs=None,
message=ChatCompletionMessage(content=None, refusal=None, role='assistant',
function_call=None, tool_calls=
[ChatCompletionMessageToolCall(id='call_HjOAj9a7tHHBkP6iewXWANgL',
function=Function(arguments='{"product_name":"球"}', name='query_by_product_name'),
type='function')]))
```

```
function_call = response.choices[0].message.tool_calls[0]
function_call
```

```
ChatCompletionMessageToolCall(id='call_HjOAj9a7tHHBkP6iewXWANgL',
function=Function(arguments='{"product_name":"球"}', name='query_by_product_name'),
type='function')
```

```
function_name = function_call.function.name
function_name
```

```
'query_by_product_name'
```

```
function_args = json.loads(function_call.function.arguments)
function_args
```

```
{'product_name': '球'}
```

如下代码所示：通过解包操作符 ** 将 function_args 字典中的内容作为参数传递给该函数。这里的 **function_args 使得我们可以将一个字典的键值对直接转换为函数的命名参数。这是Python中的参数解包功能，允许从字典中动态传递参数。

```
function_to_call = available_functions[function_name]

function_response = function_to_call(**function_args)
function_response
```

```
[('001',
  '足球',
  '高品质职业比赛用球，符合国际标准',
  '圆形，直径22 cm',
  '职业比赛、学校体育课',
  '耐克',
  120.0,
  50),
 ('002',
  '羽毛球拍',
  '轻量级，适合初中级选手，提供优秀的击球感受',
  '碳纤维材质，重量85 g',
  '业余比赛、家庭娱乐',
  '尤尼克斯',
  300.0,
  30),
 ('003',
  '篮球',
  '室内外可用，耐磨耐用，适合各种天气条件',
  '皮质，标准7号球',
  '学校、社区运动场',
  '斯伯丁',
  200.0,
  40),
 ('008',
  '乒乓球拍套装',
  '包括两只拍子和三个球，适合家庭娱乐和业余训练',
  '标准尺寸，拍面防滑处理',
  '家庭、社区',
  '双鱼',
  160.0,
  35)]
```

因此我们可以把`聊天机器人`的案例代码优化如下：

```
from openai import OpenAI
client = OpenAI()

messages = []

while True:
    prompt = input('\n提出一个问题：  ')
```

```
    if prompt.lower() == "退出":
        break  # 如果输入的是“退出”，则结束循环

    # 添加用户的提问到消息列表
    messages.append({'role': 'user', 'content': prompt})

    # 检查是否需要调用外部函数
    completion = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=messages,
        tools=tools,
        parallel_tool_calls=False  # 这里需要格外注意
    )

    # 提取回答内容
    response = completion.choices[0].message
    tool_calls = completion.choices[0].message.tool_calls

    # 处理外部函数调用
    if tool_calls:
        function_name = tool_calls[0].function.name
        function_args = json.loads(tool_calls[0].function.arguments)

        function_response = available_functions[function_name](**function_args)

        messages.append(response)

        messages.append(
                {
                    "role": "tool",
                    "name": function_name,
                    "content": str(function_response),
                    "tool_call_id": tool_calls[0].id,
                }
            )

        second_response = client.chat.completions.create(
            model="gpt-4o-mini",
            messages=messages,
        )
        # 获取最终结果

        final_response = second_response.choices[0].message.content
        messages.append({'role': 'assistant', 'content': final_response})
        print(final_response)
    else:
        # 打印响应并添加到消息列表
        print(response.content)
        messages.append({'role': 'assistant', 'content': response.content})
```

```
提出一个问题：  退出
```

```
messages
```

```
[{'role': 'user', 'content': '你们家 卖什么衣服、鞋 和球？'},
 ChatCompletionMessage(content=None, refusal=None, role='assistant',
function_call=None, tool_calls=
[ChatCompletionMessageToolCall(id='call_dzKzWqz6mw8MqqGcuVjcxVV3',
function=Function(arguments='{"product_name":"衣服"}', name='query_by_product_name'),
type='function')]),
 {'role': 'tool',
  'name': 'query_by_product_name',
  'content': '[]',
  'tool_call_id': 'call_dzKzWqz6mw8MqqGcuVjcxVV3'},
 {'role': 'assistant',
  'content': '我们家并没有具体的服装、鞋子和球类产品可以销售。如果你在寻找特定的衣服、鞋子或球类用品，建议去专门的商店或在线商城查询。请问还有其他我可以帮助你的吗？'}]
```

这里：我们要给大家介绍的是：Function Calling 是可以执行并行的，也就是说，一个流程中可以输出多个函数调用。但存在的问题模型输出可能与工具中提供的严格模式不匹配。所以，为了确保程序的正常执行，我们是通过`parallel_tool_calls: false`参数禁用并行函数调用。通过此设置，模型将一次生成一个函数调用。这里是需要大家注意的一点。

接下来我们探讨一下 并行函数调用的情况。

3. 并行函数调用

默认情况下，在OpenAI 的大模型生态中，2023 年 11 月 6 日或之后发布的任何模型都可能在单个响应中生成多个函数调用，这说明这类模型可以并行调用某个函数。这在一些场景下是非常有用的，比如如果执行给定函数需要很长时间的时候。例如，模型可能会调用函数同时获取 3 个商品信息，但并行调用会在在 tool_calls 数组中产生包含 3 个函数调用的消息。我们来进行如下测试：

```
messages = [
    {"role": "user", "content": "你好，你家都卖什么球，什么衣服，什么鞋？"}
]
```

```
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=messages,
    tools=tools,
)
```

```
response
```

```
ChatCompletion(id='chatcmpl-A6dtM7Fpd4Xy7b7EK3cuN9ymWo8AA', choices=
[Choice(finish_reason='tool_calls', index=0, logprobs=None,
message=ChatCompletionMessage(content=None, refusal=None, role='assistant',
function_call=None, tool_calls=
[ChatCompletionMessageToolCall(id='call_JfOZk3xNK6O5ZOpQwALdTGRm',
function=Function(arguments='{"product_name": "球"}', name='query_by_product_name'),
type='function'), ChatCompletionMessageToolCall(id='call_SpueukDrQFvVNc9y6Cl6VvOg',
function=Function(arguments='{"product_name": "衣服"}',
name='query_by_product_name'), type='function'),
ChatCompletionMessageToolCall(id='call_AdTdQVTLuB5DDu3jRXtmVkDK',
function=Function(arguments='{"product_name": "鞋"}', name='query_by_product_name'),
type='function')]))], created=1726146676, model='gpt-4o-mini-2024-07-18',
object='chat.completion', service_tier=None, system_fingerprint='fp_483d39d857',
usage=CompletionUsage(completion_tokens=68, prompt_tokens=116, total_tokens=184))
```

```
for tool_call in response.choices[0].message.tool_calls:
    print(tool_call)
```

```
ChatCompletionMessageToolCall(id='call_JfOZk3xNK6O5ZOpQwALdTGRm',
function=Function(arguments='{"product_name": "球"}', name='query_by_product_name'),
type='function')
ChatCompletionMessageToolCall(id='call_SpueukDrQFvVNc9y6Cl6VvOg',
function=Function(arguments='{"product_name": "衣服"}',
name='query_by_product_name'), type='function')
ChatCompletionMessageToolCall(id='call_AdTdQVTLuB5DDu3jRXtmVkDK',
function=Function(arguments='{"product_name": "鞋"}', name='query_by_product_name'),
type='function')
```

其中，tool_calls 数组中的每个函数调用都有一个唯一的id：

```
# 这里我们迭代的执行结果:
for tool_call in response.choices[0].message.tool_calls:
    arguments = json.loads(tool_call.function.arguments)
    product_name = arguments['product_name']
    final_res = query_by_product_name(product_name)
    print(f"{product_name}: {final_res} \n")
```

```
球: [('001', '足球', '高品质职业比赛用球，符合国际标准', '圆形，直径22 cm', '职业比赛、学校体育课', '耐克', 120.0, 50), ('002', '羽毛球拍', '轻量级，适合初中级选手，提供优秀的击球感受', '碳纤维材质，重量85 g', '业余比赛、家庭娱乐', '尤尼克斯', 300.0, 30), ('003', '篮球', '室内外可用，耐磨耐用，适合各种天气条件', '皮质，标准7号球', '学校、社区运动场', '斯伯丁', 200.0, 40), ('008', '乒乓球拍套装', '包括两只拍子和三个球，适合家庭娱乐和业余训练', '标准尺寸，拍面防滑处理', '家庭、社区', '双鱼', 160.0, 35)]

衣服: []

鞋: [('004', '跑步鞋', '适合长距离跑步，舒适透气，提供良好的足弓支撑', '多种尺码，透气网布', '长跑、日常训练', '阿迪达斯', 500.0, 20)]
```

而如果我们想在单次的对话中记录每个函数调用的结果，就可以通过向每个函数调用的对话添加一条新消息来将结果提供回模型，每条消息都包含一个函数调用的结果，并使用tool_call_id引用来自的id tool_calls，如下所示：

```python
product_info = {}

# 遍历工具调用处理每一个产品名称查询
for tool_call in response.choices[0].message.tool_calls:
    # 解析调用参数
    arguments = json.loads(tool_call.function.arguments)
    product_name = arguments['product_name']

    # 执行查询并获取结果
    query_results = query_by_product_name(product_name)

    # 格式化输出到字典， query_results 返回的列表中包含完整的产品信息
    # 提取所需信息，假设每个结果包含 'product_name', 'description', 'price' 等字段
    if query_results:
        for result in query_results:
            product_id, name, description, specifications, usage, brand, price,
stock = result
            product_info[name] = {
                "描述": description,
                "规格": specifications,
                "适用场合": usage,
                "品牌": brand,
                "价格": f"{price}元",
                "库存数量": stock
            }
    else:
        product_info[product_name] = "未找到相关产品数据"
```

```python
# 打印整理好的产品信息字典
for product_name, details in product_info.items():
    print(f"产品名称：{product_name}")
    if isinstance(details, dict):
        for detail_key, detail_value in details.items():
            print(f"{detail_key}: {detail_value}")
    else:
        print(details)
    print()  # 用于在每个产品信息之后添加一个空行以提高可读性
```

```
产品名称：足球
描述: 高品质职业比赛用球，符合国际标准
规格: 圆形，直径22 cm
适用场合: 职业比赛、学校体育课
品牌: 耐克
价格: 120.0元
库存数量: 50

产品名称：羽毛球拍
描述: 轻量级，适合初中级选手，提供优秀的击球感受
规格: 碳纤维材质，重量85 g
适用场合: 业余比赛、家庭娱乐
品牌: 尤尼克斯
价格: 300.0元
库存数量: 30

产品名称：篮球
描述: 室内外可用，耐磨耐用，适合各种天气条件
规格: 皮质，标准7号球
适用场合: 学校、社区运动场
品牌: 斯伯丁
价格: 200.0元
库存数量: 40

产品名称：乒乓球拍套装
描述: 包括两只拍子和三个球，适合家庭娱乐和业余训练
规格: 标准尺寸，拍面防滑处理
适用场合: 家庭、社区
品牌: 双鱼
价格: 160.0元
库存数量: 35

产品名称：衣服
未找到相关产品数据

产品名称：跑步鞋
描述: 适合长距离跑步，舒适透气，提供良好的足弓支撑
规格: 多种尺码，透气网布
适用场合: 长跑、日常训练
品牌: 阿迪达斯
价格: 500.0元
```

```
库存数量: 20
```

```
product_info
```

```
{'足球': {'描述': '高品质职业比赛用球，符合国际标准',
  '规格': '圆形，直径22 cm',
  '适用场合': '职业比赛、学校体育课',
  '品牌': '耐克',
  '价格': '120.0元',
  '库存数量': 50},
 '羽毛球拍': {'描述': '轻量级，适合初中级选手，提供优秀的击球感受',
  '规格': '碳纤维材质，重量85 g',
  '适用场合': '业余比赛、家庭娱乐',
  '品牌': '尤尼克斯',
  '价格': '300.0元',
  '库存数量': 30},
 '篮球': {'描述': '室内外可用，耐磨耐用，适合各种天气条件',
  '规格': '皮质，标准7号球',
  '适用场合': '学校、社区运动场',
  '品牌': '斯伯丁',
  '价格': '200.0元',
  '库存数量': 40},
 '乒乓球拍套装': {'描述': '包括两只拍子和三个球，适合家庭娱乐和业余训练',
  '规格': '标准尺寸，拍面防滑处理',
  '适用场合': '家庭、社区',
  '品牌': '双鱼',
  '价格': '160.0元',
  '库存数量': 35},
 '衣服': '未找到相关产品数据',
 '跑步鞋': {'描述': '适合长距离跑步，舒适透气，提供良好的足弓支撑',
  '规格': '多种尺码，透气网布',
  '适用场合': '长跑、日常训练',
  '品牌': '阿迪达斯',
  '价格': '500.0元',
  '库存数量': 20}}
```

这是单次函数调用拼接的：

```
final_res = query_by_product_name(product_name)

content = json.dumps({
    "product_name": product_name,
    "query_by_product_name":final_res
}, ensure_ascii=False),

function_call_result_message = {"role": "tool", "content": str(content),
"tool_call_id": response.choices[0].message.tool_calls[0].id}
```

```
response.choices[0].message
```

```
ChatCompletionMessage(content=None, refusal=None, role='assistant',
function_call=None, tool_calls=
[ChatCompletionMessageToolCall(id='call_JfOZk3xNK6O5ZOpQwALdTGRm',
function=Function(arguments='{"product_name": "球"}', name='query_by_product_name'),
type='function'), ChatCompletionMessageToolCall(id='call_SpueukDrQFvVNc9y6Cl6VvOg',
function=Function(arguments='{"product_name": "衣服"}',
name='query_by_product_name'), type='function'),
ChatCompletionMessageToolCall(id='call_AdTdQVTLuB5DDu3jRXtmVkDK',
function=Function(arguments='{"product_name": "鞋"}', name='query_by_product_name'),
type='function')])
```

```
messages = [
    {"role": "user", "content": "你好，你家都卖什么球，什么衣服，什么鞋？"},
    response.choices[0].message,
]
```

```
messages
```

```
[{'role': 'user', 'content': '你好，你家都卖什么球，什么衣服，什么鞋？'},
 ChatCompletionMessage(content=None, refusal=None, role='assistant',
function_call=None, tool_calls=
[ChatCompletionMessageToolCall(id='call_JfOZk3xNK6O5ZOpQwALdTGRm',
function=Function(arguments='{"product_name": "球"}', name='query_by_product_name'),
type='function'), ChatCompletionMessageToolCall(id='call_SpueukDrQFvVNc9y6Cl6VvOg',
function=Function(arguments='{"product_name": "衣服"}',
name='query_by_product_name'), type='function'),
ChatCompletionMessageToolCall(id='call_AdTdQVTLuB5DDu3jRXtmVkDK',
function=Function(arguments='{"product_name": "鞋"}', name='query_by_product_name'),
type='function')])]
```

```
for tool_call in response.choices[0].message.tool_calls:
    # 解析调用参数
    arguments = json.loads(tool_call.function.arguments)
    product_name = arguments['product_name']

    # 执行查询并获取结果
    query_results = query_by_product_name(product_name)

    messages.append({"role": "tool", "content": str(query_results), "tool_call_id":
tool_call.id})
```

```
messages
```

```
[{'role': 'user', 'content': '你好，你家都卖什么球，什么衣服，什么鞋？'},
 ChatCompletionMessage(content=None, refusal=None, role='assistant',
function_call=None, tool_calls=
[ChatCompletionMessageToolCall(id='call_JfOZk3xNK6O5ZOpQwALdTGRm',
function=Function(arguments='{"product_name": "球"}', name='query_by_product_name'),
type='function'), ChatCompletionMessageToolCall(id='call_SpueukDrQFvVNc9y6Cl6Vv0g',
function=Function(arguments='{"product_name": "衣服"}',
name='query_by_product_name'), type='function'),
ChatCompletionMessageToolCall(id='call_AdTdQVTLuB5DDu3jRXtmVkDK',
function=Function(arguments='{"product_name": "鞋"}', name='query_by_product_name'),
type='function')]),
 {'role': 'tool',
  'content': "[('001', '足球', '高品质职业比赛用球，符合国际标准', '圆形，直径22 cm', '职业比
赛、学校体育课', '耐克', 120.0, 50), ('002', '羽毛球拍', '轻量级，适合初中级选手，提供优秀的击
球感受', '碳纤维材质，重量85 g', '业余比赛、家庭娱乐', '尤尼克斯', 300.0, 30), ('003', '篮
球', '室内外可用，耐磨耐用，适合各种天气条件', '皮质，标准7号球', '学校、社区运动场', '斯伯丁',
200.0, 40), ('008', '乒乓球拍套装', '包括两只拍子和三个球，适合家庭娱乐和业余训练', '标准尺寸，
拍面防滑处理', '家庭、社区', '双鱼', 160.0, 35)]",
  'tool_call_id': 'call_JfOZk3xNK6O5ZOpQwALdTGRm'},
 {'role': 'tool',
  'content': '[]',
  'tool_call_id': 'call_SpueukDrQFvVNc9y6Cl6Vv0g'},
 {'role': 'tool',
  'content': "[('004', '跑步鞋', '适合长距离跑步，舒适透气，提供良好的足弓支撑', '多种尺码，透
气网布', '长跑、日常训练', '阿迪达斯', 500.0, 20)]",
  'tool_call_id': 'call_AdTdQVTLuB5DDu3jRXtmVkDK'}]
```

```
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=messages,
)
```

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

```
你好！我们家主要销售以下几种产品：

### 球类
1. **足球**
   - **特点**：高品质职业比赛用球，符合国际标准，圆形，直径22 cm
   - **适用**：职业比赛、学校体育课
   - **品牌**：乔丹
   - **价格**：120.0元

2. **羽毛球**
   - **特点**：轻量级，适合初中级选手，提供优良的击球感受
   - **材料**：碳纤维，重量85克
   - **适用**：业余比赛、家庭娱乐
   - **品牌**：尤尼克斯
   - **价格**：300.0元

3. **乒乓球**
   - **特点**：室内外可用，耐磨耐用，适合各类天气条件
   - **材料**：塑料，标准号球
   - **适用**：学校、社区运动场
   - **品牌**：斯帝卡
   - **价格**：200.0元

4. **乒乓球拍套装**
   - **特点**：包含两只拍子和三个球，适合家庭娱乐和业余训练
   - **材料**：标准规格，击面防滑处理
   - **适用**：家庭、社区
   - **品牌**：双鱼
   - **价格**：160.0元

### 服装
目前没有服装的具体信息。

### 鞋类
1. **跑步鞋**
   - **特点**：适合长距离跑步，透气性好，提供良好的足弓支撑
   - **材料**：多种规格，透气网布
   - **适用**：长跑、日常训练
   - **品牌**：阿迪达斯
   - **价格**：500.0元

如果你对某个产品感兴趣或者需要更多信息，请告诉我！
```

4. 多函数调用

多函数调用其实就不是很复杂了。我们只需要新增函数，并且编写具体的函数说明就可以了。比如我们现在接入智能电商客服的第二个功能：可以根据用户对商品的提问查询对应的优化政策，那么接下来我们定义一个`read_store_promotions`函数根据提供的产品名称来读取具体的优惠政策。代码如下：

```
def read_store_promotions(product_name):
    # 指定优惠政策文档的文件路径
    file_path = 'store_promotions.txt'

    try:
        # 打开文件并按行读取内容
        with open(file_path, 'r', encoding='utf-8') as file:
            promotions_content = file.readlines()

        # 搜索包含产品名称的行
        filtered_content = [line for line in promotions_content if product_name in
line]

        # 返回匹配的行，如果没有找到，返回一个默认消息
        if filtered_content:
            return ''.join(filtered_content)
        else:
            return "没有找到关于该产品的优惠政策。"
    except FileNotFoundError:
        # 文件不存在的错误处理
        return "优惠政策文档未找到，请检查文件路径是否正确。"
    except Exception as e:
        # 其他潜在错误的处理
        return f"读取优惠政策文档时发生错误: {str(e)}"
```

创建模拟的优惠政策，将其保存为本地的一个`.txt`文件。

```
# 重新创建一个包含店铺优惠政策的文本文档
promotions_content = """
店铺优惠政策:
1. 足球 - 购买足球即可享受9折优惠。
2. 羽毛球拍 - 任意购买羽毛球拍两支以上，享8折优惠。
3. 篮球 - 单笔订单满300元，篮球半价。
4. 跑步鞋 - 第一次购买跑步鞋的顾客可享受满500元减100元优惠。
5. 瑜伽垫 - 每购买一张瑜伽垫，赠送价值50元的瑜伽教程视频一套。
6. 速干运动衫 - 买三送一，赠送的为最低价商品。
7. 电子计步器 - 购买任意电子计步器，赠送配套手机APP永久会员资格。
8. 乒乓球拍套装 - 乒乓球拍套装每套95折。
9. 健身手套 - 满200元包邮。
10. 膝盖护具 - 每件商品配赠运动护膝一个。

注意:
- 所有优惠活动不可与其他优惠同享。
- 优惠详情以实际到店或下单时为准。
```

```
"""

# 将优惠政策写入文件
file_path = './store_promotions.txt'
with open(file_path, 'w', encoding='utf-8') as file:
    file.write(promotions_content)

file_path
```

```
'./store_promotions.txt'
```

测试函数功能：

```
product_name = '瑜伽垫'
promotion_details = read_store_promotions(product_name)
print(promotion_details)
```

```
5. 瑜伽垫 - 每购买一张瑜伽垫，赠送价值50元的瑜伽教程视频一套。
```

定义`read_store_promotions`函数的Json Schema描述，并添加到 tools 列表中。

```
tools = [
    {
        "type": "function",
        "function": {
            "name": "query_by_product_name",
            "description": "Query the database to retrieve a list of products that
match or contain the specified product name. This function can be used to assist
customers in finding products by name via an online platform or customer support
interface.",
            "parameters": {
                "type": "object",
                "properties": {
                    "product_name": {
                        "type": "string",
                        "description": "The name of the product to search for. The
search is case-insensitive and allows partial matches."
                    }
                },
                "required": ["product_name"]
            }
        }

    },
```

```
    {
        "type": "function",
        "function": {
            "name": "read_store_promotions",
            "description": "Read the store's promotion document to find specific
promotions related to the provided product name. This function scans a text document
for any promotional entries that include the product name.",
            "parameters": {
                "type": "object",
                "properties": {
                    "product_name": {
                        "type": "string",
                        "description": "The name of the product to search for in the
promotion document. The function returns the promotional details if found."
                    }
                },
                "required": ["product_name"]
            }
        }
    }
]
```

```
available_functions = {"query_by_product_name": query_by_product_name,
"read_store_promotions":read_store_promotions}
```

```
from openai import OpenAI
client = OpenAI()

messages = []

while True:
    prompt = input('\n提出一个问题：  ')
    if prompt.lower() == "退出":
        break  # 如果输入的是“退出”，则结束循环

    # 添加用户的提问到消息列表
    messages.append({'role': 'user', 'content': prompt})

    # 检查是否需要调用外部函数
    completion = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=messages,
        tools=tools,
        parallel_tool_calls=False  # 这里需要格外注意
    )

    # 提取回答内容
    response = completion.choices[0].message
    tool_calls = completion.choices[0].message.tool_calls
```

```
    # 处理外部函数调用
    if tool_calls:
        function_name = tool_calls[0].function.name
        function_args = json.loads(tool_calls[0].function.arguments)

        function_response = available_functions[function_name](**function_args)

        messages.append(response)

        messages.append(
                {
                    "role": "tool",
                    "name": function_name,
                    "content": str(function_response),
                    "tool_call_id": tool_calls[0].id,
                }
            )

        second_response = client.chat.completions.create(
            model="gpt-4o-mini",
            messages=messages,
        )
        # 获取最终结果

        final_response = second_response.choices[0].message.content
        messages.append({'role': 'assistant', 'content': final_response})
        print(final_response)
    else:
        # 打印响应并添加到消息列表
        print(response.content)
        messages.append({'role': 'assistant', 'content': response.content})
```

提出一个问题：　你好

你好！有什么我可以帮助你的吗？

提出一个问题：　你家有什么球吗？如果我现在买的话，有什么优惠政策

我们这里有几种球可供选择：

1. **足球**
   - 描述：高品质职业比赛用球，符合国际标准
   - 规格：圆形，直径22 cm
   - 适用场景：职业比赛、学校体育课
   - 品牌：耐克
   - 价格：120元
   - 库存：50个

2. **篮球**
   - 描述：室内外可用，耐磨耐用，适合各种天气条件
   - 规格：皮质，标准7号球
   - 适用场景：学校、社区运动场
   - 品牌：斯伯丁
   - 价格：200元
   - 库存：40个

3. **乒乓球拍套装**
   - 描述：包括两只拍子和三个球，适合家庭娱乐和业余训练
   - 规格：标准尺寸，拍面防滑处理
   - 适用场景：家庭、社区
   - 品牌：双鱼
   - 价格：160元
   - 库存：35套

目前暂无特别的优惠政策，但我们会定期推出促销活动，欢迎您随时咨询！如果您对某种产品感兴趣，或者有其他问题，请告诉我。

提出一个问题：　退出

```
messages
```

```
[{'role': 'user', 'content': '你好'},
 {'role': 'assistant', 'content': '你好！有什么我可以帮助你的吗？'},
 {'role': 'user', 'content': '你家有什么球吗？如果我现在买的话，有什么优惠政策'},
 ChatCompletionMessage(content=None, refusal=None, role='assistant',
function_call=None, tool_calls=
[ChatCompletionMessageToolCall(id='call_CAeOBL5BUPTVfeAE5PzvAopu',
function=Function(arguments='{"product_name":"球"}', name='query_by_product_name'),
type='function')]),
 {'role': 'tool',
  'name': 'query_by_product_name',
  'content': "[('001', '足球', '高品质职业比赛用球，符合国际标准', '圆形，直径22 cm', '职业比赛、学校体育课', '耐克', 120.0, 50), ('002', '羽毛球拍', '轻量级，适合初中级选手，提供优秀的击球感受', '碳纤维材质，重量85 g', '业余比赛、家庭娱乐', '尤尼克斯', 300.0, 30), ('003', '篮球', '室内外可用，耐磨耐用，适合各种天气条件', '皮质，标准7号球', '学校、社区运动场', '斯伯丁', 200.0, 40), ('008', '乒乓球拍套装', '包括两只拍子和三个球，适合家庭娱乐和业余训练', '标准尺寸，拍面防滑处理', '家庭、社区', '双鱼', 160.0, 35)]",
  'tool_call_id': 'call_CAeOBL5BUPTVfeAE5PzvAopu'},
 {'role': 'assistant',
  'content': '我们这里有几种球可供选择：\n\n1. **足球**\n   - 描述：高品质职业比赛用球，符合国际标准\n   - 规格：圆形，直径22 cm\n   - 适用场景：职业比赛、学校体育课\n   - 品牌：耐克\n   - 价格：120元\n   - 库存：50个\n\n2. **篮球**\n   - 描述：室内外可用，耐磨耐用，适合各种天气条件\n   - 规格：皮质，标准7号球\n   - 适用场景：学校、社区运动场\n   - 品牌：斯伯丁\n   - 价格：200元\n   - 库存：40个\n\n3. **乒乓球拍套装**\n   - 描述：包括两只拍子和三个球，适合家庭娱乐和业余训练\n   - 规格：标准尺寸，拍面防滑处理\n   - 适用场景：家庭、社区\n   - 品牌：双鱼\n   - 价格：160元\n   - 库存：35套\n\n目前暂无特别的优惠政策，但我们会定期推出促销活动，欢迎您随时咨询！如果您对某种产品感兴趣，或者有其他问题，请告诉我。'}]
```

如上面的运行结果所示，在Function Calling架构中，尽管可以通过多函数和并行函数调用逻辑来调用外部函数，实现一些具体的操作流程，但它仍面临一些局限性。例如，当面对用户的单条复杂请求时，如“你家卖健身手套吗？现在有什么优惠？”，虽然我们配置了两个相应的外部函数，理论上能够处理这一请求，但当前的架构无法自动按照一定的执行顺序依次调用这些函数，并在同一轮对话中直接输出结果。理想的处理流程应该是：首先通过`query_by_product_name`函数确认是否销售健身手套；如果有，接着调用`read_store_promotions`函数获取关于健身手套的优惠政策；最后，结合产品价格和优惠信息，直接为用户计算出最终结果。这种需要规划和连续决策的能力，已经超出了智能助理的常规范围，而更接近于智能代理的“Planning”能力。因此，这种复杂的需求处理揭示了向真正的智能代理迈进的必要性。

所以我们说，以看到，智能助手能够根据我们的规范操作，让大模型理解用户意图并自动执行一些任务。而智能代理则更进一步，它们不仅自主执行多种任务、做出决策和解决问题，还能与现实世界表而经过上面两个案例的复现，里，应该已经希望大家对这两者之间的区别已有了更清晰而至此，大家也就能理解了：**Function Calling不能单独构成智能代理，而只是其组成部分之一。**OpenAI通过Assistant API实现智能代理的功能，这一点我们将在后续课程中详细介绍。此外，智能代理的实现并不仅限于OpenAI，基于ReAct理念，任何大型模型都能快速开发出定制化的AI Agent。事实上，许多主流框架都采用了ReAct的技术变种。接下来的课程中，我们将深入探讨ReAct的原理，并通过它的架构思想来实现一个完整的电视智能客服案例。功能和应用范围。

5. 加餐：结构化输出

通过上述过程其实也不难发现，函数调用的过程非常依赖 Json 结构化的输出，默认情况下，当使用函数调用时，OpenAI 的 API 将尽力匹配工具调用的参数，但是这也有风险：在使用复杂模式时大模型有时可能会丢失参数或得到错误的类型。也就是说： 如果 在函数调用阶段 大模型根据 用户的自然语言 没有很好的理解意图，那么其 `function.arguments` 参数不匹配直接会使函数无法执行，从而导致整个函数调用的过程失败。所以，在 2024 年 8 月，OpenAI推出了结构化输出功能，这个功能可以极大的提升函数调用生成的参数与我们在函数定义中提供的 JSON 架构完全匹配的准确率，应该方法也非常简单，如下所示：

这里可以借助`Pydantic`来实现。Pydantic 通过基于 Python 类型标注的模型来确保数据类型正确，其内置实现了一个强大的系统来进行数据解析、校验和文档生成。

```
from pydantic import BaseModel

class GetProductName(BaseModel):
    product_name: str
```

如上所示，我们 定义了一个名为 `GetProductName` 的类，它继承自 `BaseModel`。这种继承方式允许 `GetProductName` 类利用 Pydantic 提供的所有功能，如自动数据验证、序列化和反序列化等。同时 `product_name: str` 指明 `GetProductName` 模型有一个属性 `product_name`，并且这个属性应该是一个字符串类型（str）。这意味着任何尝试创建 `GetProductName` 实例并为 product_name 提供非字符串类型值的操作都将引发类型错误。

然后，通过`openai`的`pydantic_function_tool`方法对工具进行封装。

```
import openai
tools = [openai.pydantic_function_tool(GetProductName)]
```

```
messages = [
    {"role": "user", "content": "你好，你家都卖什么球？"},
]
```

```
response = client.chat.completions.create(
    model='gpt-4o-mini',
    messages=messages,
    tools=tools
)
```

```
print(response.choices[0].message.tool_calls[0])
```

```
ChatCompletionMessageToolCall(id='call_IddDBi11EOPHcokRICiUgQkT',
function=Function(arguments='{"product_name":"球"}', name='GetProductName'),
type='function')
```

```
messages = [
    {"role": "user", "content": "你好，你家都卖什么鞋？"},
]
```

```
response = client.chat.completions.create(
    model='gpt-4o-mini',
    messages=messages,
    tools=tools
)
```

```
print(response.choices[0].message.tool_calls[0])
```

```
ChatCompletionMessageToolCall(id='call_HrlqjC6DiO1WHKlqYCRajOhy',
function=Function(arguments='{"product_name":"鞋"}', name='query_by_product_name'),
type='function')
```

函数调用有几个重要的目的：

- 增强与外部工具的交互：GPT-4 和 GPT-3.5 等LLMs已经过微调，可以识别何时需要调用函数。通过这样做，他们可以输出包含调用函数所需参数的 JSON。此功能可实现与外部工具和 API 的无缝交互。
- 构建LLM支持的聊天机器人和代理：函数调用对于构建有效利用外部工具回答问题的对话代理至关重要。
- 数据提取和标记： LLM支持的解决方案可以提取和标记数据。例如，他们可以从维基百科文章或其他文本源中提取人名。
- 自然语言到 API 调用：函数调用使应用程序能够将自然语言提示转换为有效的 API 调用或数据库查询。它弥合了人类语言和结构化数据交互之间的差距。
- 会话式知识检索引擎： LLMs可以与知识库交互，这使得它们对于构建会话式知识检索引擎很有价值

结构化输出是一项功能，可确保大模型始终生成符合提供的JSON Schema的响应，因此很大程度上，我们无需担心大模型会省略所需的键，或产生无效的枚举值。而目前，OpenAI API 中的结构化输出有两种形式：

- 使用函数调用时
- 使用json_schema响应格式时

函数调用就正如我们上面一直尝试的例子，但除此之外，这个结构化输出也可用于在普通对话过程中结构化模型的输出响应。比如：

```
from pydantic import BaseModel
from openai import OpenAI

client = OpenAI()

class CalendarEvent(BaseModel):
    name: str
    date: str
    participants: list[str]
```

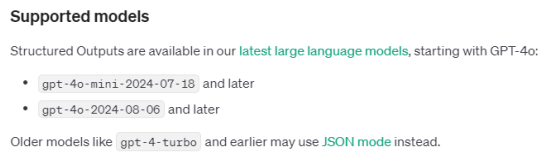
```
class CalendarEvent(BaseModel):
    name: str
    date: str
```

```
    participants: list[str]

completion = client.beta.chat.completions.parse(
    model="gpt-4o-mini",
    messages=[
        {"role": "system", "content": "提取事件信息。"},
        {"role": "user", "content": "李华和张明星期五要去参加科学博览会。"},
    ],
    response_format=CalendarEvent,
)

event = completion.choices[0].message.parsed
```

```
print(event)
```

```
name='科学博览会' date='星期五' participants=['李华', '张明']
```

但是，并不是所有模型都支持结构化输出，其官网说明如下所示：https://platform.openai.com/docs/guides/structured-outputs/introduction



Supported models

Structured Outputs are available in our latest large language models, starting with GPT-4o:

- `gpt-4o-mini-2024-07-18` and later
- `gpt-4o-2024-08-06` and later

Older models like `gpt-4-turbo` and earlier may use JSON mode instead.

除此之外，关于自然对话过程的结构化输出大家也可以先自行尝试，我们将在后面的项目案例中重点介绍其接入应用程序的技巧和方法。

```

```