Deepseek企业级Agent项目开发实战

Part 2. Ollama REST API - api/generate 接口详解

Ollama 服务启动后会提供一系列原生 `REST API` 端点。通过这些 `Endpoints` 可以在代码环境下与 `ollama` 启动的大模型进行交互、管理模型和获取相关信息。其中两个 `endpoint` 是最重要的，分别是：

- **POST /api/generate**
- **POST /api/chat**

其他端点情况：

- POST /api/create
- POST /api/tags
- POST /api/show
- POST /api/copy
- DELETE /api/delete
- POST /api/pull
- POST /api/push
- POST /api/embed
- GET /api/ps

---

1. /api/generate 接口参数概览

该接口使用提供的模型为给定提示生成响应。这是一个流式端点，因此会有一系列响应。最终响应对象将包括统计信息和请求中的其他数据。其中比较重要的参数我做了标红处理，大家重点理解。

常规参数

参数名	类型	描述
model	*(必需)*	模型名称，必须遵循 `model:tag` 格式，如果不提供，则将默认为 `latest`。
prompt	*(必需)*	用于生成响应的提示。
suffix	*(可选)*	模型响应后的文本。
images	*(可选)*	base64 编码图像的列表（适用于多模态模型，如 llava）。

高级参数（可选）

参数名	类型	描述
format	*(可选)*	返回响应的格式。格式可以是 `json` 或 JSON 模式。最主要的问题是避免产生大量空格
options	*(可选)*	文档中列出的其他模型参数，例如 `temperature`。
system	*(可选)*	系统消息，用于覆盖 Modelfile 中定义的内容。
template	*(可选)*	要使用的提示模板，覆盖 Modelfile 中定义的内容。
stream	*(可选)*	如果为 `false`，响应将作为单个响应对象返回，而不是对象流。
raw	*(可选)*	如果为 `true`，则不会对提示应用格式。
keep_alive	*(可选)*	控制模型在请求后保持加载的时间（默认：5分钟）。
context	*(可选)*	*(已弃用)* 从先前请求返回的上下文参数，用于保持简短的对话记忆。

其中，Options参数详细解释如下，同样我对重点参数做了标红处理，大家重点理解。

参数名	描述	值类型	示例用法
mirostat	启用 Mirostat 采样以控制困惑度。（默认：0，0 = 禁用，1 = Mirostat，2 = Mirostat 2.0）	int	mirostat 0
mirostat_eta	影响算法对生成文本反馈的响应速度。较低的学习率会导致调整较慢，而较高的学习率会使算法更具响应性。（默认：0.1）	float	mirostat_eta 0.1
mirostat_tau	控制输出的连贯性和多样性之间的平衡。较低的值会导致更集中和连贯的文本。（默认：5.0）	float	mirostat_tau 5.0
num_ctx	设置用于生成下一个标记的上下文窗口大小。（默认：2048），影响的是模型可以一次记住的最大 token 数量。	int	num_ctx 4096
repeat_last_n	设置模型回溯的范围以防止重复。（默认：64，0 = 禁用，-1 = num_ctx）	int	repeat_last_n 64
repeat_penalty	设置惩罚重复的强度。较高的值（例如 1.5）会更强烈地惩罚重复，而较低的值（例如 0.9）会更宽松。（默认：1.1）	float	repeat_penalty 1.1
temperature	模型的温度。增加温度会使模型的回答更具创造性。（默认：0.8）	float	temperature 0.7
seed	设置用于生成的随机数种子。将其设置为特定数字将使模型对相同提示生成相同的文本。（默认：0）	int	seed 42
stop	设置使用的停止序列。当遇到此模式时，LLM 将停止生成文本并返回。可以通过在 modelfile 中指定多个单独的停止参数来设置多个停止模式。	string	stop "AI assistant:"
num_predict	生成文本时要预测的最大标记数。（默认：-1，无限生成），影响模型最大可以生成的 token 数量。	int	num_predict 42
top_k	降低生成无意义文本的概率。较高的值（例如 100）会给出更多样化的答案，而较低的值（例如 10）会更保守。（默认：40）	int	top_k 40
top_p	与 top-k 一起工作。较高的值（例如 0.95）会导致更具多样性的文本，而较低的值（例如 0.5）会生成更集中和保守的文本。（默认：0.9）	float	top_p 0.9
min_p	top_p 的替代方案，旨在确保质量和多样性之间的平衡。参数 p 表示考虑标记的最小概率，相对于最可能标记的概率。例如，p=0.05 时，最可能的标记概率为 0.9，值小于 0.045 的 logits 会被过滤掉。（默认：0.0）	float	min_p 0.05

对于`endpoints`来说，如果使用代码调用，常规的调用方式是通`requests`库进行调用。如下所示：

```
import requests # type: ignore
import json

# 设置 API 端点
generate_url = "http://192.168.110.131:11434/api/generate"     # 这里需要根据实际情况进行修改
```

```
# 示例数据
generate_payload = {
    "model": "deepseek-r1:7b",    # 这里需要根据实际情况进行修改
    "prompt": "请生成一个关于人工智能的简短介绍。",  # 这里需要根据实际情况进行修改
    "stream": False,        # 默认使用的是True，如果设置为False，则返回的是一个完整的响应，而不是一个流式响应
}

# 调用生成接口
response_generate = requests.post(generate_url, json=generate_payload)
if response_generate.status_code == 200:
    generate_response = response_generate.json()
    print("生成响应:", json.dumps(generate_response, ensure_ascii=False, indent=2))
else:
    print("生成请求失败:", response_generate.status_code, response_generate.text)
```

```
生成请求失败: 404 {"error":"model 'deepseek-r1:7b' not found"}
```

返回的响应中包含以下参数，其对应的描述如下：

响应参数

参数名	描述
total_duration	单次响应花费的总时间
load_duration	加载模型花费的时间
prompt_eval_count	提示中的token数
prompt_eval_duration	评估提示所花费的时间（以纳秒为单位）
eval_count	响应中的token数
eval_duration	生成响应的时间（以纳秒为单位）
context	在此响应中使用的对话的编码，可以在下一个请求中发送以保持对话记忆
response	空响应是流的，如果未流式传输，则将包含完整的响应

返回的响应体中重点关注以下几个参数：

2. response 参数格式化解析

`response` 字段指的是模型生成的实际输出内容。对于 `DeepSeek-R1` 模型来说，`response` 字段中包含 标签和正常文本， 标签用于表示模型的思考过程或内部推理，而正常的文本则是模型生成的实际输出内容。注意：非推理类模型的返回结果中没有标识。

```
generate_response["response"]
```

```
'<think>\n好，用户让我生成一个关于人工智能的简短介绍。我需要先回想一下人工智能的基本概念和核心内容。\n\n首先，AI就是我们常说的自动化技术，它利用计算机系统来模拟人类智能。那主要可以分为机器学习、深度学习这些核心技术吗？\n\n接下来，AI在哪些领域应用广泛呢？比如自然语言处理，这可能涉及到聊天机器人或者翻译工具；computer vision 看图片识别，这样在医疗、制造业里面都有用处；还有智能推荐系统，比如Google Flu Trends这样的数据集。\n\n然后，我可以提到一些具体的应用案例，比如自动驾驶汽车的开发，或者智能家居系统的优化。这样内容会更具体，更有吸引力。\n\n最后，总结一下AI的重要性，它不仅改变了我们的生活，也在不断推动着科技创新和经济的发展中起作用。这样一来整个介绍就比较全面了，既有定义，又有具体的应用和未来的影响。\n</think>\n\n人工智能（Artificial Intelligence, AI）是一种基于计算机系统的技术，通过模拟人类智能来实现信息处理、学习和决策的能力。AI的核心是机器学习和深度学习等核心技术，在各个领域如自然语言处理、 computer vision、推荐系统等领域广泛应用。从汽车、智能家居到医疗诊断，AI正在深刻改变我们的生活方式和社会发展。'
```

可以通过简单的字符串操作来分离 标签中的思考内容和正常的文本内容，代码如下：

```python
# 提取 <think> 标签中的内容
think_start = generate_response["response"].find("<think>")
think_end = generate_response["response"].find("</think>")

if think_start != -1 and think_end != -1:
    think_content = generate_response["response"][think_start + len("
<think>"):think_end].strip()
else:
    think_content = "No think content found."

# 提取正常的文本内容
normal_content = generate_response["response"][think_end + len("</think>"):].strip()

# 打印结果
print("思考内容:\n", think_content)
print("\n正常内容:\n", normal_content)
```

```
思考内容:
 好，我现在需要帮用户生成一段关于人工智能的简短介绍。首先，我得弄清楚用户的使用场景和身份是什么。可能这是一个学生、老师，或者是对AI感兴趣的一般读者。他们想要的是简洁明了的信息，适合快速了解。

接下来，我要考虑用户的需求，明确他们需要涵盖哪些内容。通常，一个简短的介绍应该包括定义、主要特点、应用领域以及伦理和社会影响。这样可以让读者有一个全面但不过于深入的理解。

然后，我会思考如何组织这些信息。先从定义开始，人工智能是模拟人类智能的技术，包括学习和推理等能力。接着提到机器学习和深度学习作为关键技术，并举一些实际的应用例子，比如语音助手、图像识别和自动驾驶，这样更具体易懂。

最后，考虑到伦理和社会影响部分，这是现代讨论AI时的重要方面，所以加入隐私保护和算法偏见等内容，可以让介绍更加全面且有深度。整体语言要简洁明了，避免专业术语过多，确保读者容易理解。

正常内容:
 人工智能（Artificial Intelligence, AI）是一种模拟人类智能的技术，通过计算机系统实现学习、推理、感知和自主决策等能力。它涵盖多个领域，如机器学习、自然语言处理和机器人技术。AI应用广泛，包括语音助手、图像识别、自动驾驶等，正在深刻改变生活与工作方式。然而，其发展也带来伦理和社会挑战，如隐私保护和算法偏见等问题。
```

当然也可以用相同的方式提取返回的响应中所有参数的值：

```python
# 打印每个参数的值
print("Model:", generate_response["model"])
print("Created At:", generate_response["created_at"])
print("Response:", generate_response["response"])
print("Done:", generate_response["done"])
print("Done Reason:", generate_response["done_reason"])
print("Context:", generate_response["context"])
print("Total Duration:", generate_response["total_duration"])
print("Load Duration:", generate_response["load_duration"])
print("Prompt Eval Count:", generate_response["prompt_eval_count"])
print("Prompt Eval Duration:", generate_response["prompt_eval_duration"])
print("Eval Count:", generate_response["eval_count"])
print("Eval Duration:", generate_response["eval_duration"])
```

```
Model: deepseek-r1:32b
Created At: 2025-02-13T10:23:02.289673324Z
Response: <think>
好，我现在需要帮用户生成一段关于人工智能的简短介绍。首先，我得弄清楚用户的使用场景和身份是什么。可能这是一个学生、老师，或者是对AI感兴趣的一般读者。他们想要的是简洁明了的信息，适合快速了解。

接下来，我要考虑用户的需求，明确他们需要涵盖哪些内容。通常，一个简短的介绍应该包括定义、主要特点、应用领域以及伦理和社会影响。这样可以让读者有一个全面但不过于深入的理解。

然后，我会思考如何组织这些信息。先从定义开始，人工智能是模拟人类智能的技术，包括学习和推理等能力。接着提到机器学习和深度学习作为关键技术，并举一些实际的应用例子，比如语音助手、图像识别和自动驾驶，这样更具体易懂。

最后，考虑到伦理和社会影响部分，这是现代讨论AI时的重要方面，所以加入隐私保护和算法偏见等内容，可以让介绍更加全面且有深度。整体语言要简洁明了，避免专业术语过多，确保读者容易理解。
</think>

人工智能（Artificial Intelligence, AI）是一种模拟人类智能的技术，通过计算机系统实现学习、推理、感知和自主决策等能力。它涵盖多个领域，如机器学习、自然语言处理和机器人技术。AI应用广泛，包括语音助手、图像识别、自动驾驶等，正在深刻改变生活与工作方式。然而，其发展也带来伦理和社会挑战，如隐私保护和算法偏见等问题。
Done: True
Done Reason: stop
```

```
Context: [151644, 14880, 43959, 46944, 101888, 104455, 9370, 98237, 99534, 100157, 1773,
151645, 151648, 198, 52801, 3837, 107520, 85106, 99663, 20002, 43959, 104383, 101888,
104455, 9370, 98237, 99534, 100157, 1773, 101140, 3837, 35946, 49828, 102115, 101222,
107494, 37029, 102122, 33108, 101294, 102021, 1773, 87267, 105464, 99720, 5373, 101049,
3837, 105471, 32664, 15469, 103198, 99774, 99791, 104785, 1773, 99650, 103945, 100146,
110485, 30858, 34187, 105427, 3837, 100231, 101098, 99794, 1773, 198, 198, 104326, 3837,
104515, 101118, 20002, 104378, 3837, 100692, 99650, 85106, 102994, 102224, 43815, 1773,
102119, 3837, 46944, 98237, 99534, 9370, 100157, 99730, 100630, 91282, 5373, 99558,
100772, 5373, 99892, 100650, 101034, 112811, 106640, 99564, 1773, 99654, 107366, 104785,
104133, 100011, 77288, 100632, 34204, 100403, 108894, 1773, 198, 198, 101889, 3837,
105351, 104107, 100007, 99877, 100001, 27369, 1773, 60726, 45181, 91282, 55286, 3837,
104455, 20412, 105717, 103971, 100168, 105535, 3837, 100630, 100134, 33108, 113272,
49567, 99788, 1773, 102524, 104496, 102182, 100134, 33108, 102217, 100134, 100622,
114876, 3837, 62926, 99357, 101883, 99912, 106736, 103358, 3837, 101912, 105761, 110498,
5373, 107553, 102450, 33108, 109044, 3837, 99654, 33126, 100398, 86744, 100272, 1773,
198, 198, 100161, 3837, 106350, 112811, 106640, 99564, 99659, 3837, 100346, 100390,
104075, 15469, 13343, 101945, 99522, 3837, 99999, 101963, 107120, 100153, 33108, 107018,
99835, 88970, 112223, 3837, 107366, 100157, 101896, 100011, 100136, 18830, 102217, 1773,
101932, 102064, 30534, 110485, 30858, 34187, 3837, 101153, 99878, 116925, 106071, 3837,
103944, 104785, 100047, 101128, 1773, 198, 151649, 198, 198, 104455, 9909, 9286, 16488,
21392, 11, 15235, 7552, 101158, 105717, 103971, 100168, 105535, 3837, 67338, 104564,
72448, 101884, 100134, 5373, 113272, 5373, 108272, 33108, 100842, 102041, 49567, 99788,
1773, 99652, 102994, 101213, 100650, 3837, 29524, 102182, 100134, 5373, 99795, 102064,
54542, 33108, 104354, 99361, 1773, 15469, 99892, 100789, 3837, 100630, 105761, 110498,
5373, 107553, 102450, 5373, 109044, 49567, 3837, 96555, 101295, 101933, 99424, 57218,
99257, 75768, 1773, 103968, 3837, 41146, 99185, 74763, 100393, 112811, 106640, 104036,
3837, 29524, 107120, 100153, 33108, 107018, 99835, 88970, 105108, 1773]
Total Duration: 10020507026
Load Duration: 63824681
Prompt Eval Count: 13
Prompt Eval Duration: 28000000
Eval Count: 301
Eval Duration: 9926000000
```

`Ollama` 返回的响应中，采用的时间单位均以纳秒返回。纳秒（nanosecond）和秒（second）之间的关系是：1 秒 = 10^9 纳秒

```python
# 将纳秒转换为秒
total_duration_s = generate_response["total_duration"] / 1_000_000_000
load_duration_s = generate_response["load_duration"] / 1_000_000_000
prompt_eval_duration_s = generate_response["prompt_eval_duration"] / 1_000_000_000
eval_duration_s = generate_response["eval_duration"] / 1_000_000_000

# 打印转换后的秒值
print("单次调用总花费时间:", total_duration_s)
print("加载模型花费时间:", load_duration_s)
print("评估提示所花费的时间:", prompt_eval_duration_s)
print("生成响应的时间:", eval_duration_s)
```

```
单次调用总花费时间: 10.020507026
加载模型花费时间: 0.063824681
评估提示所花费的时间: 0.028
生成响应的时间: 9.926
```

3. num_ctx / num_predict 输入输出控制

`num_ctx` 和 `num_predict` 参数都是需要放置在 `options` 参数中的，其中：

- `num_ctx` 该参数指的是大模型在一次对话中能够"看到"和"记住"的最大上下文长度，默认配置 2048，相当于一次只能向模型输入 2k `token`，超过 2k 模型就无法记住。当 `prompt` 特别长时往往会出现问题。并且现在开源模型往往支持长上下文，默认配置会严重限制本地模型能力。
- `num_predict` 参数指的是模型响应返回的最大 token 数据量。

我们可以这样测试：

```
import requests # type: ignore
import json

# 设置 API 端点
generate_url = "http://192.168.110.131:11434/api/generate"     # 这里需要根据实际情况进行修改

# 示例数据
generate_payload = {
    "model": "deepseek-r1:32b",    # 这里需要根据实际情况进行修改
    "prompt": "请生成一个关于人工智能的简短介绍。",  # 这里需要根据实际情况进行修改
    "stream": False,        # 默认使用的是True，如果设置为False，则返回的是一个完整的响应，而不是一个流式响应
    "options": {
        # "num_ctx": 7,  慎用，可能会导致Ollama服务不稳定，建议选择 1024 及以上
        "num_predict": 10
    }
}

# 调用生成接口
response_generate = requests.post(generate_url, json=generate_payload)
if response_generate.status_code == 200:
    generate_response = response_generate.json()
    print("生成响应:", json.dumps(generate_response, ensure_ascii=False, indent=2))
else:
    print("生成请求失败:", response_generate.status_code, response_generate.text)
```

```
生成响应: {
  "model": "deepseek-r1:32b",
  "created_at": "2025-02-13T10:41:58.256206177Z",
  "response": "<think>\n嗯，用户让我生成一个关于人工智能",
  "done": true,
  "done_reason": "length",
  "context": [
    151644,
    14880,
    43959,
    46944,
    101888,
    104455,
    9370,
    98237,
    99534,
    100157,
    1773,
    151645,
```

```
    151648,
    198,
    106287,
    3837,
    20002,
    104029,
    43959,
    46944,
    101888,
    104455
  ],
  "total_duration": 18876756388,
  "load_duration": 14726802999,
  "prompt_eval_count": 13,
  "prompt_eval_duration": 3829000000,
  "eval_count": 10,
  "eval_duration": 318000000
}
```

4. 流式输出功能

接下来看流式输出输出，其参数和如上代码保持一致，只需要在 `response_generate` 中添加 `stream=True`，最后再通过流式的方式进行响应结果处理即可。代码如下所示：

> 这里有一个使用`DeepSeek-R1`的小技巧，将温度即`temperature`设置在0.5-0.7（建议0.6）的范围内，可以有效防止无尽的重复或不连贯的输出。

```python
import requests # type: ignore
import json

# 设置 API 端点
generate_url = "http://192.168.110.131:11434/api/generate"

# 示例数据
generate_payload = {
    "model": "deepseek-r1:32b",
    "prompt": "请生成一个关于人工智能的简短介绍。",
    "options": {
        "temperature": 0.6,
    }
}

# 调用生成接口
response_generate = requests.post(generate_url, json=generate_payload, stream=True)  # 在这里添加stream=True
if response_generate.status_code == 200:
    # 处理流式响应
    for line in response_generate.iter_lines():
        if line:
            try:
                # 解码并解析每一行的 JSON
                response_json = json.loads(line.decode('utf-8'))
                if 'response' in response_json:
                    print(response_json['response'], end='', flush=True)
```

```
                # 检查 response_json 字典中是否存在键 'done'，并且其值是否为 True。如果这个条件成立，表示生成的响应已经完成。
                if response_json.get('done', False):
                    print('\n\n完整响应:', json.dumps(response_json, ensure_ascii=False, indent=2))
            except json.JSONDecodeError as e:
                print(f"JSON 解析错误: {e}")
else:
    print("生成请求失败:", response_generate.status_code, response_generate.text)
```

<think>
好的，用户让我生成一个关于人工智能的简短介绍。首先，我需要明确什么是人工智能，通常称为AI。它指的是模拟人类智能的技术，包括学习、推理和问题解决等能力。

接下来，我要考虑用户的使用场景可能是什么。可能是学生写作业，或者是在准备演讲时需要参考资料。用户的身份可能不是专业人士，所以介绍要简明易懂，避免过于技术化的术语。

然后，我得想一下用户的真实需求。他们可能只是想快速了解AI的基本概念和应用领域，而不是深入的技术细节。因此，我应该涵盖主要的应用范围，比如机器学习、自然语言处理等，并提到一些实际例子，如语音助手或自动驾驶。

另外，用户可能还希望知道AI的影响，包括积极的一面，比如提高效率和医疗诊断，以及潜在的挑战，如隐私问题和伦理考量。这样可以让介绍更全面，帮助用户理解AI的重要性及其带来的影响。

最后，我需要确保整个介绍结构清晰，逻辑连贯，用词简洁明了，让用户能够轻松理解人工智能的基本概念、应用和发展前景。
</think>

人工智能（Artificial Intelligence, AI）是指通过模拟人类智能的系统或机器，使计算机能够执行复杂的任务，如学习、推理、问题解决和自然语言处理等。AI技术广泛应用于语音助手、图像识别、自动驾驶和医疗诊断等领域，正在深刻改变我们的生活方式和社会发展。

完整响应: {


```
  "model": "deepseek-r1:32b",
  "created_at": "2025-02-13T10:53:40.624562752Z",
  "response": "",
  "done": true,
  "done_reason": "stop",
  "context": [
    151644,
    14880,
    43959,
    46944,
    101888,
    104455,
    9370,
    98237,
    99534,
    100157,
    1773,
    151645,
    151648,
    198,
    99692,
    3837,
    20002,
    104029,
    43959,
```

```
    46944,
    101888,
    104455,
    9370,
    98237,
    99534,
    100157,
    1773,
    101140,
    3837,
    35946,
    85106,
    100692,
    106582,
    104455,
    3837,
    102119,
    102424,
    15469,
    1773,
    99652,
    111198,
    105717,
    103971,
    100168,
    105535,
    3837,
    100630,
    100134,
    5373,
    113272,
    33108,
    86119,
    100638,
    49567,
    99788,
    1773,
    198,
    198,
    104326,
    3837,
    104515,
    101118,
    107494,
    37029,
    102122,
    87267,
    102021,
    1773,
    104560,
    99720,
    61443,
    104147,
    3837,
    100631,
    101219,
    101077,
    105465,
```

```
    13343,
    85106,
    112872,
    1773,
    20002,
    106613,
    87267,
    99520,
    114516,
    3837,
    99999,
    100157,
    30534,
    98237,
    30858,
    86744,
    100272,
    3837,
    101153,
    102767,
    99361,
    105302,
    116925,
    1773,
    198,
    198,
    101889,
    3837,
    35946,
    49828,
    99172,
    100158,
    20002,
    103277,
    100354,
    1773,
    99650,
    87267,
    100009,
    99172,
    101098,
    99794,
    15469,
    105166,
    101290,
    33108,
    99892,
    100650,
    3837,
    104610,
    100403,
    105535,
    104449,
    1773,
    101886,
    3837,
    35946,
    99730,
```

```
    102994,
    99558,
    106736,
    101121,
    3837,
    101912,
    102182,
    100134,
    5373,
    99795,
    102064,
    54542,
    49567,
    3837,
    62926,
    104496,
    101883,
    99912,
    103358,
    3837,
    29524,
    105761,
    110498,
    57191,
    109044,
    1773,
    198,
    198,
    101948,
    3837,
    20002,
    87267,
    97706,
    99880,
    99392,
    15469,
    104126,
    3837,
    100630,
    99666,
    111866,
    3837,
    101912,
    100627,
    101991,
    33108,
    100182,
    105262,
    3837,
    101034,
    106362,
    9370,
    104036,
    3837,
    29524,
    107120,
    86119,
    33108,
```

```
    112811,
    110260,
    1773,
    99654,
    107366,
    100157,
    33126,
    100011,
    3837,
    100364,
    20002,
    101128,
    15469,
    106508,
    104204,
    102220,
    99564,
    1773,
    198,
    198,
    100161,
    3837,
    35946,
    85106,
    103944,
    101908,
    100157,
    100166,
    104542,
    3837,
    104913,
    54926,
    100116,
    3837,
    11622,
    99689,
    110485,
    30858,
    34187,
    3837,
    115987,
    100006,
    104261,
    101128,
    104455,
    105166,
    101290,
    5373,
    99892,
    106344,
    102653,
    1773,
    198,
    151649,
    198,
    198,
    104455,
    9909,
```

```
    9286,
    16488,
    21392,
    11,
    15235,
    7552,
    104442,
    67338,
    105717,
    103971,
    100168,
    9370,
    72448,
    57191,
    102182,
    3837,
    32555,
    104564,
    100006,
    75117,
    106888,
    88802,
    3837,
    29524,
    100134,
    5373,
    113272,
    5373,
    86119,
    100638,
    33108,
    99795,
    102064,
    54542,
    49567,
    1773,
    15469,
    99361,
    100789,
    110645,
    105761,
    110498,
    5373,
    107553,
    102450,
    5373,
    109044,
    33108,
    100182,
    105262,
    106483,
    3837,
    96555,
    101295,
    101933,
    103952,
    107142,
    106640,
```

```
    99185,
    1773
  ],
  "total_duration": 23138032274,
  "load_duration": 10248659864,
  "prompt_eval_count": 13,
  "prompt_eval_duration": 3370000000,
  "eval_count": 289,
  "eval_duration": 9517000000
}
```

5. Ollama 模型生命周期管理

默认情况下，**通过 `Ollama run` 启动一个模型后，会将其在VRAM(显存)中保存5分钟**。主要作用是为了做性能优化，通过保持模型在显存中，可以避免频繁的加载和卸载操作，从而提高响应速度，特别是在连续请求的情况下。

- **keep_alive**

我们可以通过 `ollama stop` 命令立即卸载某个模型。而在生成请求中，一种高效的方式是通过 `keep_alive` 参数来控制模型在请求完成后保持加载在内存中的时间。其可传入的参数规则如下：

keep_alive 参数类型

参数类型	示例	描述
持续时间字符串	"10m" 或 "24h"	表示保持模型在内存中的时间，单位可以是分钟（m）或小时（h）。
以秒为单位的数字	3600	表示保持模型在内存中的时间，单位为秒。
任何负数	-1 或 "-1m"	表示保持模型在内存中，负数值将使模型持续加载。
'0'	0	表示在生成响应后立即卸载模型。

```python
import requests # type: ignore
import json

# 设置 API 端点
generate_url = "http://192.168.110.131:11434/api/generate"

# 示例数据
generate_payload = {
    "model": "deepseek-r1:32b",
    "prompt": "请生成一个关于人工智能的简短介绍。",
    "stream": False,
    "keep_alive": "10m",    # 设置模型在请求后保持加载的时间
    "options": {
        "temperature": 0.6,
    }
}
```

```python
# 调用生成接口
response_generate = requests.post(generate_url, json=generate_payload)
if response_generate.status_code == 200:
    generate_response = response_generate.json()
    print("生成响应:", json.dumps(generate_response, ensure_ascii=False, indent=2))
else:
    print("生成请求失败:", response_generate.status_code, response_generate.text)



if generate_response["eval_duration"] != 0:
    tokens_per_second = generate_response["eval_count"] /
generate_response["eval_duration"] * 10**9
    print(f"Tokens per second: {tokens_per_second}")
else:
    print("eval_duration is zero, cannot calculate tokens per second.")
```


生成响应: {
  "model": "deepseek-r1:32b",
  "created_at": "2025-02-13T11:06:33.5111786Z",
  "response": "<think>\n嗯，用户让我生成一个关于人工智能的简短介绍。首先，我得理解用户的需求是什么。可能他正在做研究，或者想快速了解AI的基本概念。简短说明要涵盖主要方面，但又不能太长。\n\n那我应该从定义开始，解释什么是人工智能，以及它模仿的是什么人类能力。然后，提到一些核心技术，比如机器学习和深度学习，这样可以展示AI的发展基础。接着，列举几个应用领域，如语音识别、图像处理和自然语言处理，这样用户能明白AI的实际用途。\n\n还要强调AI带来的便利，比如提高效率和推动社会进步。同时，不能忽略潜在的挑战，比如隐私和伦理问题，这显示全面性。最后，可以展望一下未来的发展趋势，让介绍更有深度。\n\n总的来说，结构要清晰，每个部分简明扼要，用词准确但不过于技术化，确保用户容易理解。这样生成的介绍既全面又简洁，符合用户的需求。\n</think>\n\n人工智能（Artificial Intelligence, AI）是指模拟人类智能的系统或机器，能够执行如学习、推理、问题解决和语言理解等任务。AI的核心技术包括机器学习和深度学习，通过大量数据训练模型，使其具备自主决策能力。如今，AI广泛应用于语音识别、图像处理、自然语言处理等领域，为社会带来便利的同时，也引发对隐私、伦理等问题的思考。未来，随着技术进步，人工智能将继续推动社会发展与变革。",
  "done": true,
  "done_reason": "stop",
  "context": [
    151644,
    14880,
    43959,
    46944,
    101888,
    104455,
    9370,
    98237,
    99534,
    100157,
    1773,
    151645,
    151648,
    198,
    106287,
    3837,
    20002,
    104029,
    43959,
    46944,
    101888,
    104455,
    9370,

```
    98237,
    99534,
    100157,
    1773,
    101140,
    3837,
    35946,
    49828,
    101128,
    20002,
    104378,
    102021,
    1773,
    87267,
    42411,
    96555,
    99190,
    99556,
    3837,
    100631,
    99172,
    101098,
    99794,
    15469,
    105166,
    101290,
    1773,
    98237,
    99534,
    66394,
    30534,
    102994,
    99558,
    99522,
    3837,
    77288,
    99518,
    53153,
    99222,
    45861,
    1773,
    198,
    198,
    99212,
    35946,
    99730,
    45181,
    91282,
    55286,
    3837,
    104136,
    106582,
    104455,
    3837,
    101034,
    99652,
    108391,
    100146,
```

```
    99245,
    103971,
    99788,
    1773,
    101889,
    3837,
    104496,
    101883,
    110025,
    3837,
    101912,
    102182,
    100134,
    33108,
    102217,
    100134,
    3837,
    99654,
    73670,
    101987,
    15469,
    103949,
    99896,
    1773,
    102524,
    3837,
    118569,
    100204,
    99892,
    100650,
    3837,
    29524,
    105761,
    102450,
    5373,
    107553,
    54542,
    33108,
    99795,
    102064,
    54542,
    3837,
    99654,
    20002,
    26232,
    101265,
    15469,
    108081,
    105795,
    1773,
    198,
    198,
    104019,
    104046,
    15469,
    102220,
    102421,
    3837,
```

```
    101912,
    100627,
    101991,
    33108,
    101890,
    99328,
    101300,
    1773,
    91572,
    3837,
    53153,
    103325,
    106362,
    9370,
    104036,
    3837,
    101912,
    107120,
    33108,
    112811,
    86119,
    3837,
    43288,
    54021,
    100011,
    33071,
    1773,
    100161,
    3837,
    73670,
    109789,
    100158,
    100353,
    103949,
    101226,
    3837,
    99258,
    100157,
    104885,
    102217,
    1773,
    198,
    198,
    116880,
    3837,
    100166,
    30534,
    104542,
    3837,
    103991,
    99659,
    98237,
    30858,
    117225,
    30534,
    3837,
    11622,
    99689,
```

```
    102188,
    77288,
    100632,
    34204,
    99361,
    32108,
    3837,
    103944,
    20002,
    100047,
    101128,
    1773,
    99654,
    43959,
    9370,
    100157,
    99929,
    100011,
    99518,
    110485,
    3837,
    101137,
    20002,
    104378,
    1773,
    198,
    151649,
    198,
    198,
    104455,
    9909,
    9286,
    16488,
    21392,
    11,
    15235,
    7552,
    104442,
    105717,
    103971,
    100168,
    9370,
    72448,
    57191,
    102182,
    3837,
    100006,
    75117,
    29524,
    100134,
    5373,
    113272,
    5373,
    86119,
    100638,
    33108,
    102064,
    101128,
```

```
    49567,
    88802,
    1773,
    15469,
    104867,
    99361,
    100630,
    102182,
    100134,
    33108,
    102217,
    100134,
    3837,
    67338,
    100722,
    20074,
    104034,
    104949,
    3837,
    102989,
    102094,
    100842,
    102041,
    99788,
    1773,
    101936,
    3837,
    15469,
    100789,
    110645,
    105761,
    102450,
    5373,
    107553,
    54542,
    5373,
    99795,
    102064,
    54542,
    106483,
    3837,
    17714,
    99328,
    100393,
    102421,
    104146,
    3837,
    74763,
    102361,
    32664,
    107120,
    5373,
    112811,
    105108,
    9370,
    104107,
    1773,
    100353,
```

```
    3837,
    101067,
    99361,
    101300,
    3837,
    104455,
    106326,
    101890,
    108256,
    57218,
    105316,
    1773
  ],
  "total_duration": 23224924189,
  "load_duration": 9684027991,
  "prompt_eval_count": 13,
  "prompt_eval_duration": 3400000000,
  "eval_count": 306,
  "eval_duration": 10138000000
}
Tokens per second: 30.18346813967252
```

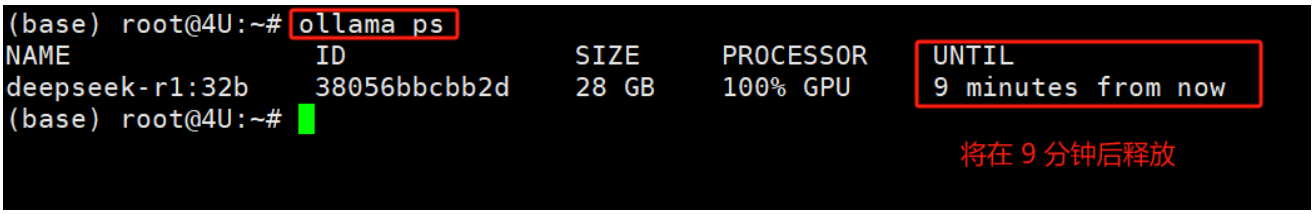
在当前单元格或上一个单元格中执行代码时 Kernel 崩溃。

请查看单元格中的代码，以确定故障的可能原因。

单击此处了解详细信息。

有关更多详细信息，请查看 Jupyter log。

此时就可以在服务器控制台查看到，`deepseek-r1:32b`模型将可以在显存中保持10分钟。



`keep_alive`在工程化的项目中，往往需要根据请求的频率来设置，如果请求不频繁，可以使用默认值或较短的时间，以便在不使用时释放内存。而如果应用程序需要频繁调用模型，可以设置较长的`keep_alive`时间，以减少加载时间。很关键，非常影响服务器的性能和应用程序的用户体验。大家一定要注意。