Deepseek企业级Agent项目开发实战

Part 1. Ollama 本地部署 Deepseek R1 模型

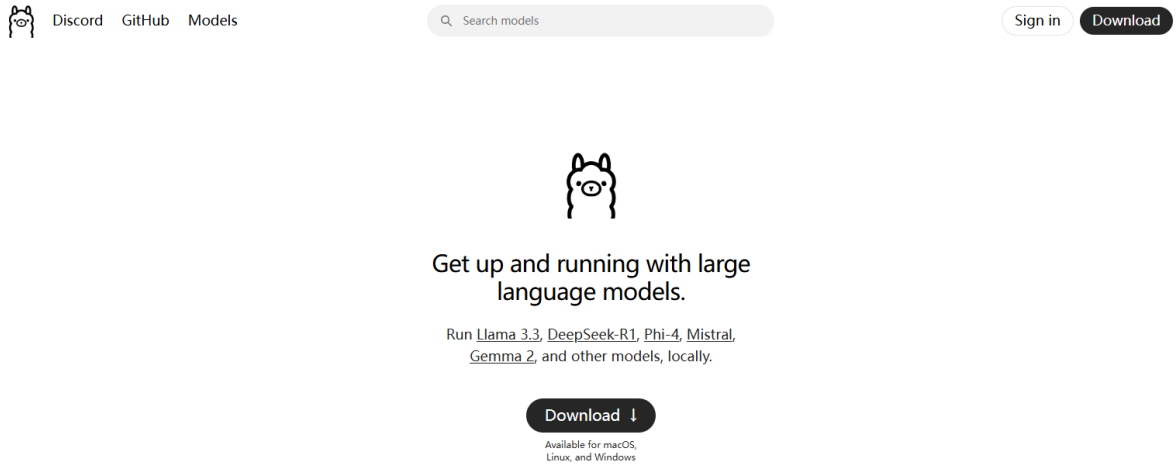
1. Ollama项目介绍

`Ollama`是在`Github`上的一个开源项目，其项目定位是：**一个本地运行大模型的集成框架**，目前主要针对主流的`LLaMA`架构的开源大模型设计，通过将模型权重、配置文件和必要数据封装进由`Modelfile`定义的包中，从而实现大模型的下载、启动和本地运行的自动化部署及推理流程。此外，`Ollama`内置了一系列针对大模型运行和推理的优化策略，目前作为一个非常热门的大模型托管平台，基本主流的大模型应用开发框架如`LangChain`、`AutoGen`、`Microsoft GraphRAG`及热门项目`AnythingLLM`、`OpenWebUI`等高度集成。

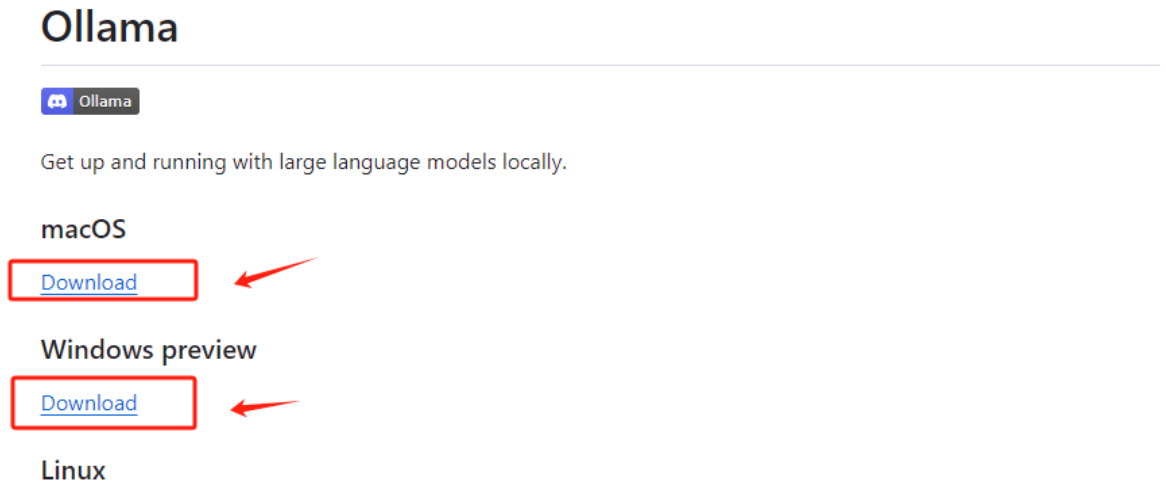
> `Ollama`通过将大模型运行的所有必要组件（如权重文件、配置设置和相关数据）封装在一个单一的文件或包中，`Modelfile 允许用户更容易地下载、安装、配置和启动模型。这种方法类似于其他软件或应用程序的安装包，它们将所有必要的文件打包在一起，以便用户可以通过简单的安装过程将软件添加到他们的系统中。

> Ollama官方地址：https://ollama.com/

> Ollama Github开源地址：https://github.com/ollama/ollama



`Ollama`项目支持跨平台部署，目前已兼容**Mac、Linux和Windows**操作系统。特别地对`Mac`和`Windows`用户提供了非常直观的预览版，包括了内置的`GPU`加速功能、访问完整模型库的能力，以及对`OpenAI`的兼容性在内的`Ollama REST API`，对用户使用尤为友好。



我们重点介绍在Ubuntu 22.04系统下安装部署Ollama项目的详细步骤。具体来说，Ollama在Ubunut系统上的安装方式有两种，分别是：**一键安装和手动安装**，但**不论使用哪种方法进行安装，都需要安装Ollama项目的服务器上具备网络连通环境**，因为不仅涉及Ollama安装包的更新，还会涉及后续大模型的下载。

2. Ollama项目本地安装

`Ollama`项目本地安装的方法极为简单，这里我们以`Linux`系统为例，先进入命令行终端，执行如下一条命令行即可自动化完成：

```
curl -fsSL https://ollama.com/install.sh | sh
```

```
(base) root@4U:~# curl -fsSL https://ollama.com/install.sh | sh
>>> Installing ollama to /usr/local
>>> Downloading Linux amd64 bundle
######################################################################## 100.0%
>>> Adding ollama user to render group...
>>> Adding ollama user to video group...
>>> Adding current user to ollama group...
>>> Creating ollama systemd service...
>>> Enabling and starting ollama service...
>>> NVIDIA GPU installed.
```

这行命令的目的是从`https://ollama.com/`网站读取`install.sh`脚本，并立即通过`sh`执行该脚本，在安装过程中会包含以下几个主要的操作：

1. 检查当前服务器的基础环境，如系统版本等；
2. 下载Ollama的二进制文件；
3. 配置系统服务，包括创建用户和用户组，添加Ollama的配置信息；
4. 启动Ollama服务；

这个过程会比较慢，拉取的文件约2G左右，如果安装过程中未出现任何错误信息，通常情况下能够表明安装已成功。可以通过执行以下命令来检查Ollama服务的运行状态：

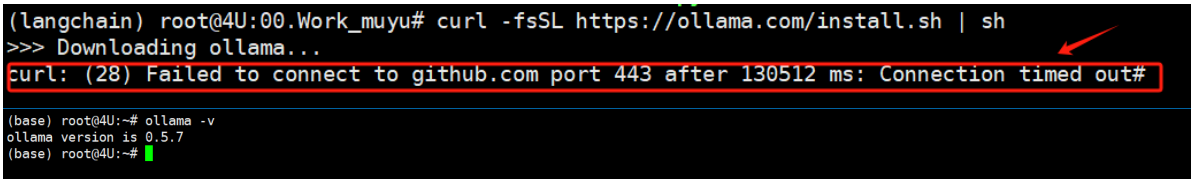
```
systemctl status ollama
```

```
(base) root@4U:~# systemctl status ollama
● ollama.service - Ollama Service
     Loaded: loaded (/etc/systemd/system/ollama.service; enabled; vendor preset: enabled)
     Active: active (running) since Tue 2025-02-11 18:54:26 CST; 1min 4s ago
   Main PID: 2779171 (ollama)
      Tasks: 16 (limit: 231378)
     Memory: 35.6M
        CPU: 782ms
     CGroup: /system.slice/ollama.service
             └─2779171 /usr/local/bin/ollama serve

2月 11 18:54:27 4U ollama[2779171]: [GIN-debug] HEAD   /                         --> github.com/ollama/ollama/server.(*Server).GenerateRoutes.func1 (5 h>
2月 11 18:54:27 4U ollama[2779171]: [GIN-debug] HEAD   /api/tags                 --> github.com/ollama/ollama/server.(*Server).ListHandler-fm (5 handler>
2月 11 18:54:27 4U ollama[2779171]: [GIN-debug] HEAD   /api/version              --> github.com/ollama/ollama/server.(*Server).GenerateRoutes.func2 (5 h>
2月 11 18:54:27 4U ollama[2779171]: time=2025-02-11T18:54:27.027+08:00 level=INFO source=routes.go:1238 msg="Listening on 127.0.0.1:11434 (version 0.5.7>
2月 11 18:54:27 4U ollama[2779171]: time=2025-02-11T18:54:27.028+08:00 level=INFO source=routes.go:1267 msg="Dynamic LLM libraries" runners="[cpu_avx cp>
2月 11 18:54:27 4U ollama[2779171]: time=2025-02-11T18:54:27.029+08:00 level=INFO source=gpu.go:226 msg="looking for compatible GPUs"
2月 11 18:54:27 4U ollama[2779171]: time=2025-02-11T18:54:27.743+08:00 level=INFO source=types.go:131 msg="inference compute" id=GPU-84297d89-08e4-e21e->
2月 11 18:54:27 4U ollama[2779171]: time=2025-02-11T18:54:27.743+08:00 level=INFO source=types.go:131 msg="inference compute" id=GPU-3363d295-16ba-4a31->
2月 11 18:54:27 4U ollama[2779171]: time=2025-02-11T18:54:27.743+08:00 level=INFO source=types.go:131 msg="inference compute" id=GPU-2f30cfd0-af19-64b3->
2月 11 18:54:27 4U ollama[2779171]: time=2025-02-11T18:54:27.743+08:00 level=INFO source=types.go:131 msg="inference compute" id=GPU-4a3c9301-84de-7334->
```

如果`Active`状态显示为`active`，则说明Ollama服务目前处于正常运行状态。同时还可以通过以下命令查询当前安装的Ollama版本：

```
sudo ollama -v
```

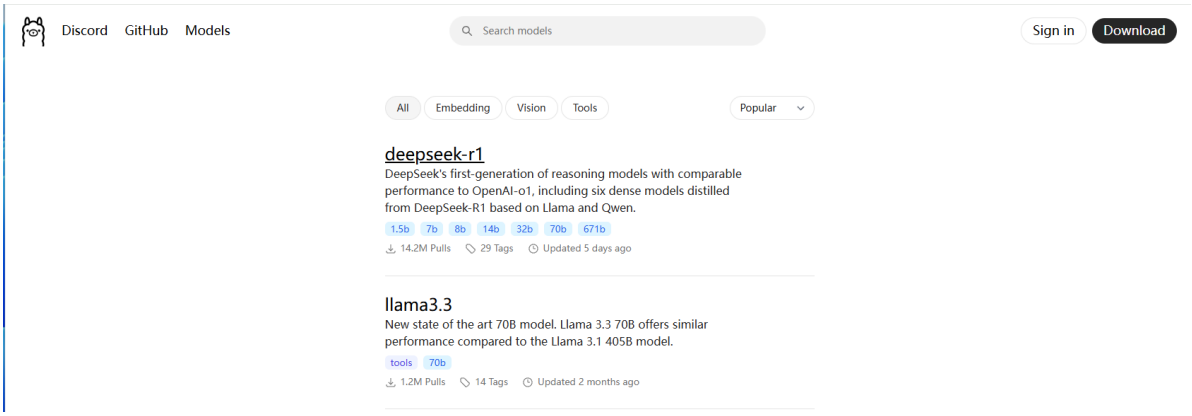
请注意：这种安装方式需要服务器保持联网状态以自动下载`ollama`的二进制文件。如果出现下述报错，则说明网络环境不通，需要根据实际情况处理网络连接。



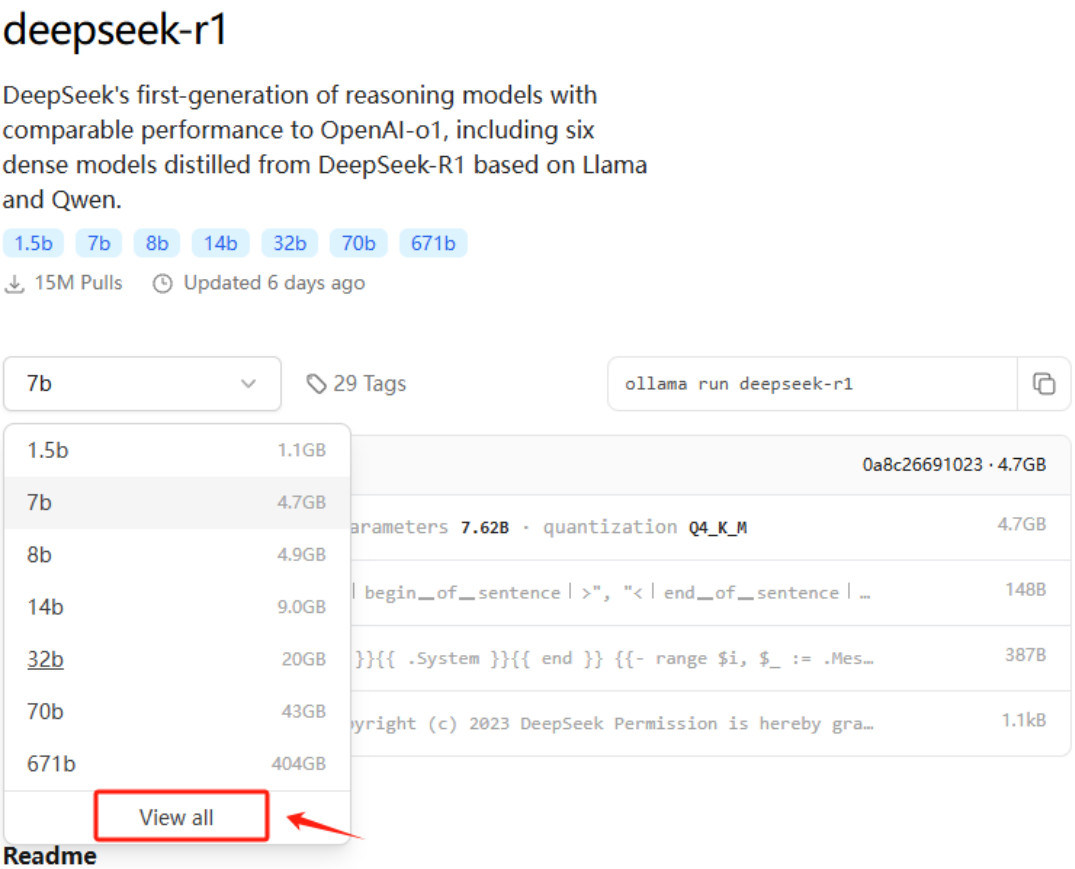
至此，我们已成功完成`ollama`项目的本地部署，并顺利启动了`ollama`服务。下面，我们将介绍如何开始使用该服务。

3. Ollama下载 DeepSeek R1 及启动

需要说明的一点是：`ollama`项目虽然提供了本地化大模型的能力，但这并不意味着所有大模型都可以通过它下载和使用，其支持的大模型的详细列表可在`ollama`的官方模型库页面查看：https://ollama.com/library。



在`ollama`的模型库中主要支持的还是基于`LLaMA`架构的一些主流大模型，并且现在已经全面接入了`DeepSeek R1`满血版模型及其蒸馏的小模型，可以进入如下页面查看所有可使用的`DeepSeek`模型。注意：`ollama`暂时没有接入`DeepSeek v3`模型。



在进入到大模型的详细信息页面后，可以通过下拉菜单选择不同参数量的大模型版本。然后需要复制页面右侧提供的模型标识符以进行下一步的模型下载操作。

接下来回到服务器的命令行终端，直接复制并运行此命令即可执行`Deepseek R1`模型文件的自动化下载，执行的具体命令如下：

```
    ollama run deepseek-r1:32b
```

```
(base) root@4U:~# ollama run deepseek-r1:32b
pulling manifest
pulling 6150cb382311... 100% ▕▕▕
```

```
(base) root@4U:models# cd /usr/share/ollama/.ollama/models/
(base) root@4U:models# ll
total 36
drwxr-xr-x 4 ollama ollama  4  9月  18 18:13 ./
drwxr-xr-x 3 ollama ollama  5  9月  18 18:13 ../
drwxr-xr-x 2 ollama ollama 18  2月  13 14:19 blobs/
drwxr-xr-x 3 ollama ollama  3  2月  12 15:21 manifests/
(base) root@4U:models# cd manifests/
(base) root@4U:manifests# ll
total 3
drwxr-xr-x 3 ollama ollama 3  2月  12 15:21 ./
drwxr-xr-x 4 ollama ollama 4  9月  18 18:13 ../
drwxr-xr-x 3 ollama ollama 3 11月   6 11:13 registry.ollama.ai/
(base) root@4U:manifests# cd registry.ollama.ai/
(base) root@4U:registry.ollama.ai# ll
total 3
drwxr-xr-x 3 ollama ollama 3 11月   6 11:13 ./
drwxr-xr-x 3 ollama ollama 3  2月  12 15:21 ../
drwxr-xr-x 4 ollama ollama 4  2月  12 15:22 library/
(base) root@4U:registry.ollama.ai# cd library/
(base) root@4U:library# ll
total 4
drwxr-xr-x 4 ollama ollama 4  2月  12 15:22 ./
drwxr-xr-x 3 ollama ollama 3 11月   6 11:13 ../
drwxr-xr-x 2 ollama ollama 6  2月  13 14:21 deepseek-r1/
drwxr-xr-x 2 ollama ollama 3  2月  13 14:13 qwen2.5/
(base) root@4U:library#
```

`Ollama` 下载的模型是 `GGUF` 格式。`GGUF`（Generalized Graph Universal Format）是一种用于存储和表示模型的格式。它与原版开源模型的关系是：

- 首先下载原版的开源模型（例如这里的 `DeepSeek-R1-Distill-Qwen-32B`）。
- 通过转化脚本将原版开源模型被转换为 `GGUF` 格式
- 将 `GGUF` 格式的模型文件量化为较低的精度

在 `Ollama` 中，最常用的量化类型是 `Q4_K_M`，表示 `4-bit` 量化，旨在在保持较高性能的同时减少模型的存储需求。

此外，还可以使用命令 `ollama list` 来直接查看通过 `Ollama` 下载的大模型文件列表，这些模型都支持在线启动和调用。

```
(base) root@4U:library# ollama list
NAME                ID              SIZE      MODIFIED
deepseek-r1:32b     38056bbcbb2d    19 GB     14 minutes ago
qwen2.5:1.5b        65ec06548149    986 MB    5 hours ago
deepseek-r1:70b     0c1615a8ca32    42 GB     23 hours ago
deepseek-r1:7b      0a8c26691023    4.7 GB    26 hours ago
deepseek-r1:1.5b    a42b25d8c10a    1.1 GB    5 days ago
```

4. Ollama启动和使用方法

在 `Ollama` 的机制中，使用 `run` 命令时，系统会首先检查本地是否已经存在指定的模型，如果本地没有找到该模型，`Ollama` 会自动执行 `ollama pull <model_name>` 命令，从远程仓库下载该模型，下载完成后将模型存储为 `GGUF` 格式，供后续使用。最后，当成功下载后，`Ollama` 会继续执行 `run` 命令，启动模型并进行推理或生成任务。

因此是可以直接通过在命令行终端对启动的大模型进行调用的，如下所示：

```
(base) root@4U:~# ollama run deepseek-r1:32b
>>> 你好，请你介绍一下什么是大模型
<think>
嗯，我现在要弄清楚什么是“大模型”。根据之前的回答，它主要是指参数量特别多的深度学习模型，比如
从头开始分析一下。

首先，大模型的特点包括高参数量、复杂结构和强大的任务处理能力。我需要理解每个特点的具体含义。
多的神经元和连接，这可能让它能够捕捉更复杂的模式。复杂结构可能指的是像Transformer这样的架构
者卷积层。任务处理能力强则说明它能做很多不同的事情，比如翻译、问答、文本生成等等。

接下来，技术基础方面提到了深度学习、Transformer架构和分布式训练与并行计算。我需要理解这些技
深度学习是机器学习的一个分支，主要用多层神经网络来模拟数据的特征。Transformer是一种基于注意
理序列数据，比如文本。而分布式训练和并行计算则是为了处理大量的数据和参数，提高训练效率。

应用领域方面，自然语言处理、计算机视觉和其他领域都有涉及。NLP里的机器翻译、问答系统和文本生
视觉中的图像识别和视频分析也有用到大模型；其他领域比如语音处理和推荐系统也可能在使用这些模型

挑战与未来方向部分提到了计算资源需求高、训练时间长、过拟合风险以及伦理问题。这些都是需要解决
更高效的方法来优化模型，减少资源消耗，并提高模型的可解释性和安全性。
```

这里要重点说明两点：其一是`DeepSeek R1`作为推理模型，其返回结果是包含的，里面包含的是思考推理的内容；其二也会存在中为空，这其实是因为`DeepSeek-R1`系列模型倾向于绕过思维模式（即输出”\n\n”）,因此一个使用的技巧是：每个输出的开头强制模型以 "\n" 开头。（此问题我们在代码环节在给大家讲解实现的方式）

5. Ollama 多GPU部署及serve启动

使用最简单的命令，即`ollama run xxxx`时，`Ollama`的内部机制会根据启动模型的参数量去运行该模型所需的`VRAM`(显存)。如果该模型可以使用单个`GPU`加载，则`Ollama`将在该`GPU`上加载该模型。这种做法一般可以提供出最佳的性能，因为它可以减少推理过程中`PCI`总线的数据传输量。而如果该模型没办法仅在一个`GPU`上加载，则将分布在所有可用的`GPU`中。比如：

根据官网的介绍，`DeepSeek-r1:32b`模型需要占用`20GB`显存。

实际也确实运行在了单张`3090 GPU`上，占用约`21GB`显存，如下：

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 530.30.02              Driver Version: 530.30.02    CUDA Version: 12.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf           Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce RTX 3090         On | 00000000:18:00.0 Off |                  N/A |
| 30%   38C    P8              28W / 350W|     19MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA GeForce RTX 3090         On | 00000000:3B:00.0 Off |                  N/A |
| 32%   44C    P2             107W / 350W|  21606MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA GeForce RTX 3090         On | 00000000:86:00.0 Off |                  N/A |
| 30%   39C    P8              22W / 350W|      8MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   3  NVIDIA GeForce RTX 3090         On | 00000000:AF:00.0 Off |                  N/A |
| 35%   38C    P8              20W / 350W|      8MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

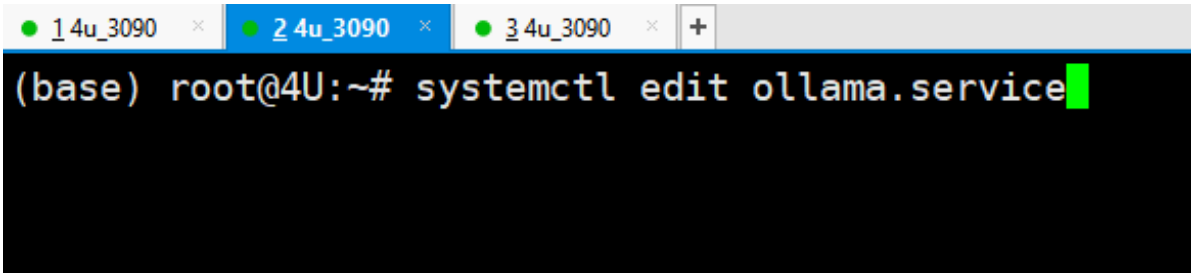
如果想加载多张显卡且做到负载均衡，可以去修改 `ollama` 的 `SystemD` 配置服务，首先找到当前服务器上 `GPU` 的 `ID`，执行命令如下：

```
nvidia-smi
```

```
(base) root@4U:~# nvidia-smi
Thu Feb 13 17:09:06 2025
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 530.30.02              Driver Version: 530.30.02    CUDA Version: 12.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf           Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce RTX 3090         On | 00000000:18:00.0 Off |                  N/A |
| 30%   37C    P8              21W / 350W|     19MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA GeForce RTX 3090         On | 00000000:3B:00.0 Off |                  N/A |
| 32%   37C    P8              25W / 350W|      8MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA GeForce RTX 3090         On | 00000000:86:00.0 Off |                  N/A |
| 30%   38C    P8              22W / 350W|      8MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   3  NVIDIA GeForce RTX 3090         On | 00000000:AF:00.0 Off |                  N/A |
| 34%   38C    P8              21W / 350W|      8MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A      5779      G   /usr/lib/xorg/Xorg                 14MiB |
|    1   N/A  N/A      5779      G   /usr/lib/xorg/Xorg                  4MiB |
|    2   N/A  N/A      5779      G   /usr/lib/xorg/Xorg                  4MiB |
|    3   N/A  N/A      5779      G   /usr/lib/xorg/Xorg                  4MiB |
+-----------------------------------------------------------------------------+
```

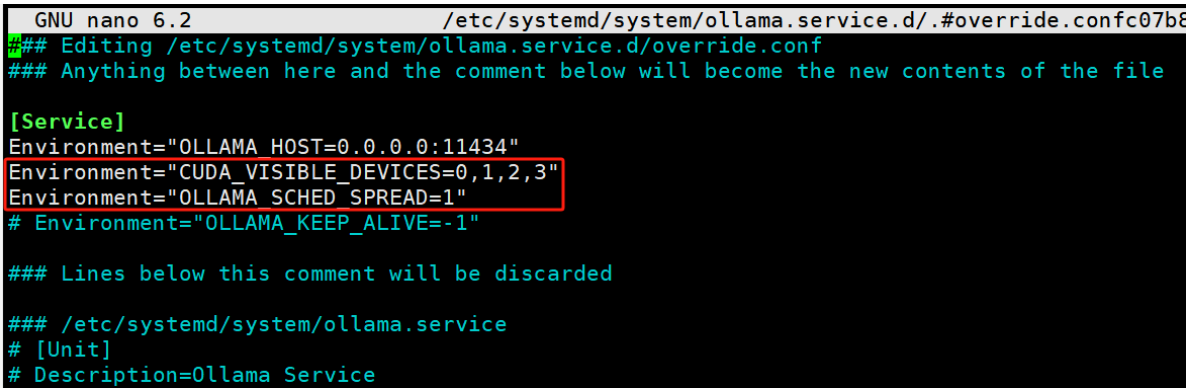
如果想加载多张显卡且做到负载均衡，可以去修改 `ollama` 的 `SystemD` 配置服务，执行如下代码：

```
systemctl edit ollama.service
```



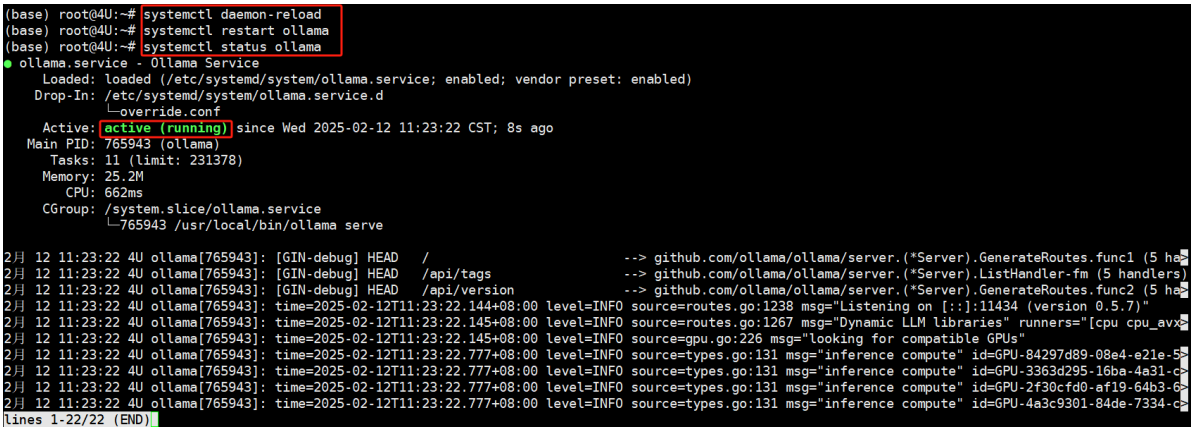
编辑并填写如下内容：

```
    Environment="CUDA_VISIBLE_DEVICES=0,1,2,3"     # 这里根据你自己实际的 GPU标号来进行修改
    Environment="OLLAMA_SCHED_SPREAD=1"            # 这个参数是做负载均衡
```

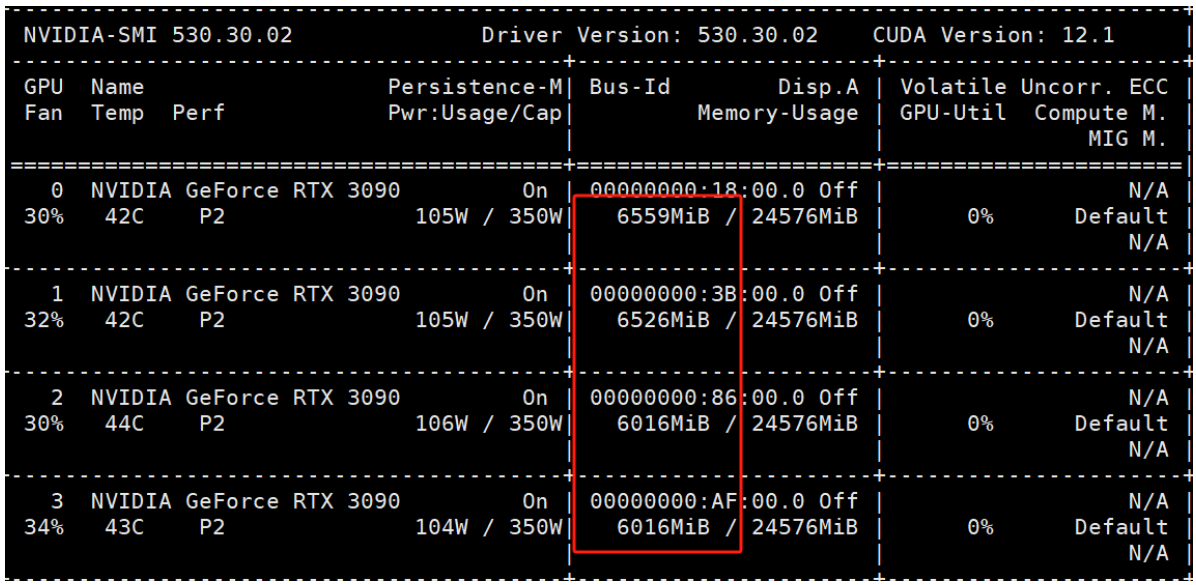


保存退出后，重新加载`systemd`并重新启动`ollama`服务使其配置生效，执行如下命令：

```
    systemctl daemon-reload
    systemctl restart ollama
```



此时再次通过`ollama run xxx`即可分布式的加载到多张`GPU`显卡上，如下所示：



6. Ollama REST API 服务启动及调用

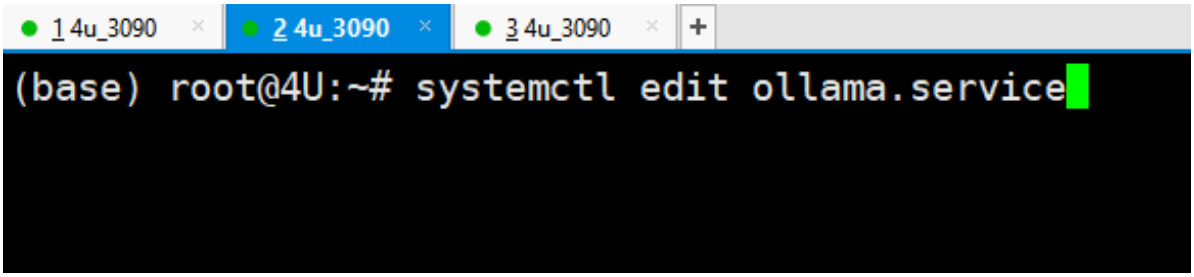
`Ollama run xxx` 命令启动模型后，不仅仅是可以在命令行终端与启动的大模型进行对话，更重要的是它还会同步启动 `Ollama REST API`，这个 `REST API` 服务简单理解：我们可以通过某种方式在代码环境中调用到使用 `Ollama` 模型启动的大模型，从而和大模型进行对话。默认绑定的 `IP + Port` 是：`http://localhost:11434`，所以，如果启动 `Ollama` 的服务和当前的代码环境是同一台机器的话，可以使用如下代码进行快速的调用测试：

```
from openai import OpenAI

client = OpenAI(
    base_url='http://localhost:11434/v1/',
    api_key='ollama',  # 这里随便写，但是api_key字段一定要有
)

chat_completion = client.chat.completions.create(
    model='deepseek-r1:32b',        # 这里要修改成 你 ollama 启动模型的名称
    messages=[
        {
            'role': 'user',
            'content': '你好，请你介绍一下你自己',
        }
    ],
)

print(chat_completion)
```

这里需要注意的一点是：如果 `Ollama` 启动和执行调用的代码是同一台机器，上述代码是可以的跑通的。比如 `Ollama` 服务在云服务器、局域网的服务器上等情况，则无法通过 `http://localhost:11434/v1/` 来进行访问，因为**网络不通**。 正如上述的报错，我的 `Ollama` 模型服务是在局域网的服务器上，因此我需要修改 `Ollama REST API` 的请求地址，操作方法如下：

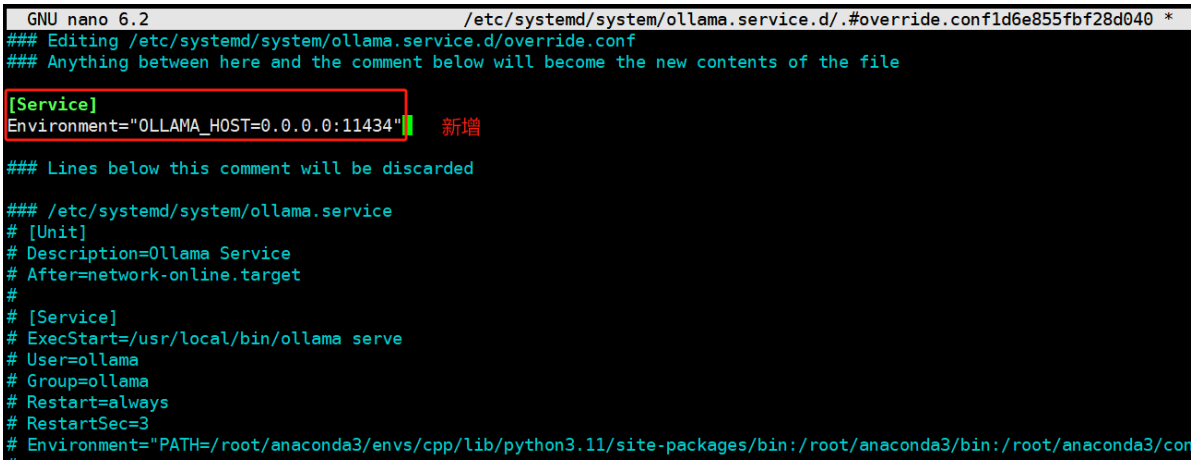
修改 `ollama` 的 `SystemD` 配置服务，执行如下代码：

```
    systemctl edit ollama.service
```



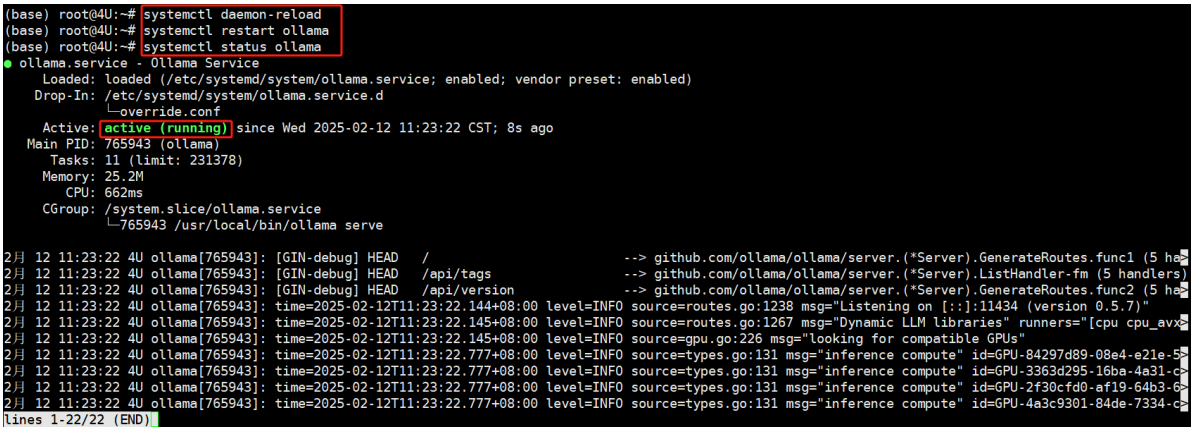
编辑并填写如下内容：

```
Environment="OLLAMA_HOST=0.0.0.0:11434"
```



保存退出后，重新加载 `systemd` 并重新启动 `ollama` 服务使其配置生效，执行如下命令：

```
systemctl daemon-reload
systemctl restart ollama
```



使用 `ollama run xxx` 启动模型。然后找到服务器可访问的有效 `IP`。在 `Linux` 系统中，可以通过多种方式查看有效的访问 `IP` 地址（即当前与系统建立连接或尝试访问系统的远程 `IP` 地址）。这里使用如下命令：

```
sudo netstat -tn | grep ESTABLISHED
```

```
(base) root@4U:~# sudo netstat -tn | grep ESTABLISHED
tcp        0      0 192.168.110.131:8002    192.168.110.190:58535   ESTABLISHED
tcp        0      0 127.0.0.1:40871         127.0.0.1:34144         ESTABLISHED
tcp        0      0 127.0.0.1:50354         127.0.0.1:58389         ESTABLISHED
tcp        0      0 127.0.0.1:41576         127.0.0.1:40149         ESTABLISHED
tcp        0      0 127.0.0.1:41534         127.0.0.1:40149         ESTABLISHED
tcp        0      0 127.0.0.1:40149         127.0.0.1:41534         ESTABLISHED
tcp        0      0 192.168.110.131:22      192.168.110.190:58042   ESTABLISHED
tcp        0      0 127.0.0.1:47939         127.0.0.1:53866         ESTABLISHED
tcp        0    140 192.168.110.131:22      192.168.110.190:54378   ESTABLISHED
tcp        0      0 127.0.0.1:53844         127.0.0.1:47939         ESTABLISHED
tcp        0      0 192.168.110.131:8002    192.168.110.190:58560   ESTABLISHED
tcp        0      0 127.0.0.1:40936         127.0.0.1:11434         ESTABLISHED
tcp        0      0 192.168.110.131:34760   192.168.110.134:445     ESTABLISHED
tcp        0      0 127.0.0.1:34144         127.0.0.1:40871         ESTABLISHED
tcp        0      0 127.0.0.1:58389         127.0.0.1:50320         ESTABLISHED
tcp        0      0 127.0.0.1:58389         127.0.0.1:50354         ESTABLISHED
tcp        0      0 127.0.0.1:40807         127.0.0.1:52360         ESTABLISHED
tcp        0      0 127.0.0.1:40149         127.0.0.1:41576         ESTABLISHED
tcp        0      0 127.0.0.1:50320         127.0.0.1:58389         ESTABLISHED
tcp        0      0 192.168.110.131:8002    192.168.110.190:58410   ESTABLISHED
tcp        0      0 127.0.0.1:53866         127.0.0.1:47939         ESTABLISHED
tcp        0      0 192.168.110.131:22      192.168.110.133:65206   ESTABLISHED
tcp        0      0 127.0.0.1:52360         127.0.0.1:40807         ESTABLISHED
tcp        0      0 127.0.0.1:47939         127.0.0.1:53844         ESTABLISHED
tcp6       0      0 127.0.0.1:11434         127.0.0.1:40936         ESTABLISHED
```

因此，修改访问`Ollama`的`REST API`地址，如下所示：

```python
from openai import OpenAI

client = OpenAI(
    base_url='http://192.168.110.131:11434/v1/',      # 这里修改成可访问的 IP
    api_key='ollama',    # 这里随便写，但是api_key字段一定要有
)

chat_completion = client.chat.completions.create(
    model='deepseek-r1:32b',
    messages=[
        {
            'role': 'user',
            'content': '你好，请你介绍一下你自己',
        }
    ],
)

print(chat_completion)
```

```
ChatCompletion(id='chatcmpl-309', choices=[Choice(finish_reason='stop', index=0,
logprobs=None, message=ChatCompletionMessage(content='<think>\n我是DeepSeek-R1，一
个由深度求索公司开发的智能助手，我会尽我所能为您提供帮助。\n</think>\n\n我是DeepSeek-R1，一
个由深度求索公司开发的智能助手，我会尽我所能为您提供帮助。', refusal=None,
role='assistant', audio=None, function_call=None, tool_calls=None))],
created=1739439431, model='deepseek-r1:32b', object='chat.completion',
service_tier=None, system_fingerprint='fp_ollama',
usage=CompletionUsage(completion_tokens=53, prompt_tokens=8, total_tokens=61,
completion_tokens_details=None, prompt_tokens_details=None))
```

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

```
<think>
我是DeepSeek-R1，一个由深度求索公司开发的智能助手，我会尽我所能为您提供帮助。
</think>

我是DeepSeek-R1，一个由深度求索公司开发的智能助手，我会尽我所能为您提供帮助。
```

至此，我们就可以像访问大模型`在线API`一样调用本地通过`Ollama`启动的`DeepSeek`模型了。而关于数据隐私问题，因为`Ollama`在本地服务器运行，因此所有的对话数据不会离开机器，大家无需担心隐私数据泄露问题。

同时，`Ollama`还有其他的一些常见操作命令，也都非常直观易懂，如下所示：

```
(base) root@4U:~# ollama
Usage:
  ollama [flags]
  ollama [command]

Available Commands:
  serve       Start ollama
  create      Create a model from a Modelfile
  show        Show information for a model
  run         Run a model
  stop        Stop a running model
  pull        Pull a model from a registry
  push        Push a model to a registry
  list        List models
  ps          List running models
  cp          Copy a model
  rm          Remove a model
  help        Help about any command

Flags:
  -h, --help      help for ollama
  -v, --version   Show version information

Use "ollama [command] --help" for more information about a command.
```

`Ollama`每个命令参数非常容易理解，大家可以自行进行尝试，其参数说明如下所示：

命令	描述
`serve`	启动 Ollama 服务
`create`	从 Modelfile 创建一个模型
`show`	显示模型的信息
`run`	运行一个模型
`stop`	停止正在运行的模型
`pull`	从注册表中拉取一个模型
`push`	将一个模型推送到注册表
`list`	列出所有模型
`ps`	列出正在运行的模型
`cp`	复制一个模型
`rm`	删除一个模型
`help`	显示关于任何命令的帮助信息

通过上述关于`Ollama`的安装、模型下载及启动推理的介绍和实践，我们可以感受到`Ollama`极大地简化了大模型部署的过程，也降低了大模型在使用上的技术门槛。然而，对大部分用户而言，命令行界面并不够友好。正如我们之前提到的，在大模型的应用开发框架下，使用到的往往是其`API`调用形式，为此，`Ollama`也是可以集成多个开源项目，包括`Web`界面、桌面应用和终端工具等方式提升使用体验，并满足满足不同用户的偏好和需求。