GPU 云服务器

实践教程

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2025 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

腾讯云

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

版权所有：腾讯云计算（北京）有限责任公司

实践教程
基于 Linux GPU 云服务器安装 NVIDIA Container Toolkit

最近更新时间：2025-05-22 17:50:02

操作场景

在 Docker 中，容器默认无法直接访问宿主机的 GPU 资源。为了解决这一限制，NVIDIA 官方提供了 NVIDIA Docker 容器支持方案，用于将宿主机的 GPU 运行时环境映射到容器中。该方案经历了多个阶段的演进，从最初的 nvidia-docker、nvidia-docker2，发展到现在的 NVIDIA Container Toolkit。
NVIDIA Container Toolkit 是一套用于容器运行时环境的工具包，能够使容器具备使用 NVIDIA GPU 的能力。本文将介绍如何在 Linux GPU 云服务器使用腾讯云源安装 NVIDIA Container Toolkit。

操作步骤

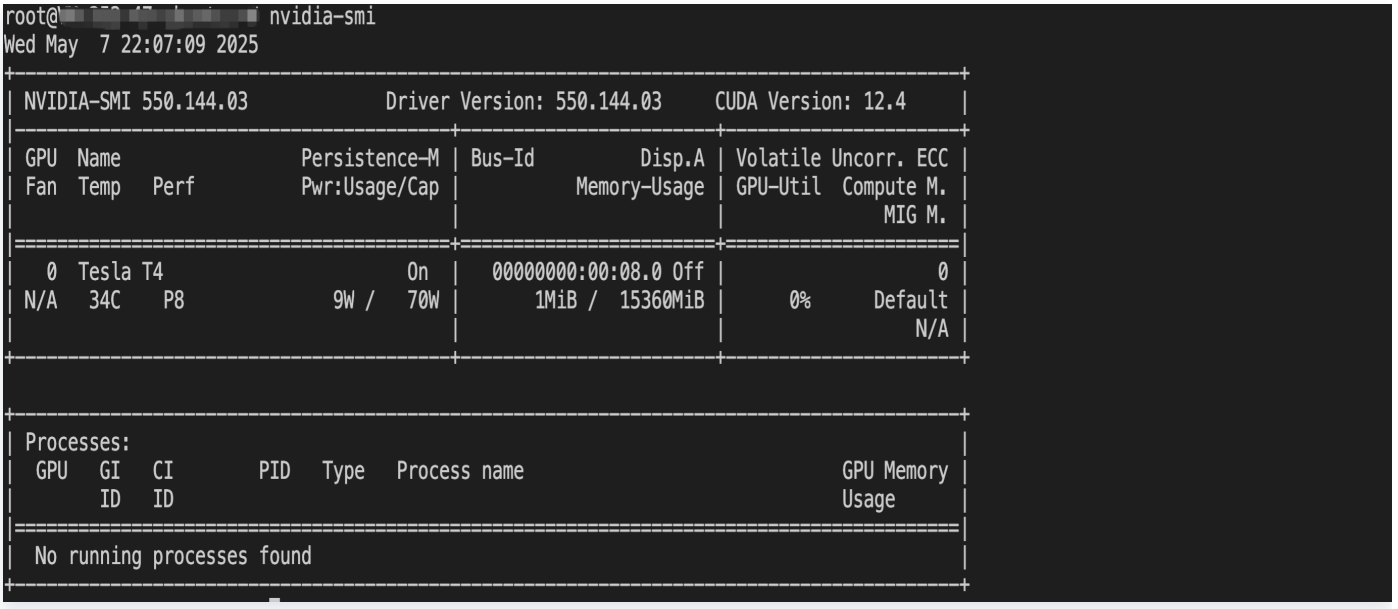
实例环境准备

1. 检查 GPU 驱动是否已安装。

   登录实例，执行如下命令确认 GPU 驱动是否正常安装：

   ```
   nvidia-smi
   ```

   若如下图所示，正常返回 GPU 设备说明已安装驱动；若返回提示 `nvidia-smi: command not found` 说明没有安装驱动，需参见 NVIDIA 驱动安装指引 安装 NVIDIA GPU 驱动。



2. 确保实例已安装并启动 docker 。

   执行如下命令可以确认是否安装 docker 及启动 docker 服务。

```
docker ps
```

   若返回如下图所示，说明已安装 docker 及正常启动。若未正常返回可参见 搭建 docker 安装 docker。

3. 检查是否未安装过 NVIDIA Container Toolkit。

   通过如下命令尝试启动 GPU 容器：

```
docker run --gpus all [镜像名称]
```

   若返回报错信息

```
docker: Error response from daemon: could not select device driver "" with capabilities: [[gpu]]
```

   ，说明当前系统没有安装 NVIDIA Container Toolkit，缺少必要的 GPU 驱动配置支持，无法识别并分配 GPU 资源给容器。

安装 NVIDIA Container Toolkit

1. 本步骤以腾讯云的 TencentOS Server、CentOS、RHEL、Rocky Linux 、Ubuntu、Debian 操作系统 GPU 服务器为例，若涉及其他操作系统请参见 Installing the NVIDIA Container Toolkit 。

TencentOS Server/CentOS/RHEL/Rocky Linux

```
curl -s -L https://mirrors.tencentyun.com/libnvidia-container/stable/rpm/nvidia-container-toolkit.repo | sudo tee /etc/yum.repos.d/nvidia-container-toolkit.repo #配置nvidia-container-toolkit源
sed -i 's/nvidia.github.io/mirrors.tencentyun.com/g' /etc/yum.repos.d/nvidia-container-toolkit.repo #修改nvidia-container-toolkit源为腾讯云源
sudo yum install -y nvidia-container-toolkit #TencentOS 2.4/CentOS 7/RHEL 7 使用yum安装nvidia-container-toolkit
sudo dnf install -y nvidia-container-toolkit #TencentOS 3.1/TencentOS 4/CentOS 8/RHEL 8/RHEL 9/Rocky Linux 8/Rocky Linux 9 使用dnf安装nvidia-container-toolkit
```

```
 sudo rpm -qa | grep nvidia-container-toolkit #查看NVIDIA Container
Toolkit是否安装成功，若有nvidia-container-toolkit 相关软件信息说明安装成功
```

Ubuntu/Debian

```
curl -fsSL https://mirrors.tencentyun.com/libnvidia-container/gpgkey |
sudo gpg --dearmor -o /usr/share/keyrings/nvidia-container-toolkit-
keyring.gpg \
  && curl -s -L https://mirrors.tencentyun.com/libnvidia-
container/stable/deb/nvidia-container-toolkit.list | \
    sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-
container-toolkit-keyring.gpg] https://#g' | \
    sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list #配置
nvidia-container-toolkit源
sed -i -e '/experimental/ s/^#//g' /etc/apt/sources.list.d/nvidia-
container-toolkit.list #启用experimental仓库
sed -i 's/nvidia.github.io/mirrors.tencentyun.com/g'
/etc/apt/sources.list.d/nvidia-container-toolkit.list #修改nvidia-
container-toolkit源为腾讯云源
sudo apt-get update #更新软件包列表
sudo apt-get install -y nvidia-container-toolkit #安装 NVIDIA Container
Toolkit 软件包
sudo dpkg -l | grep nvidia-container-toolkit #查看NVIDIA Container
Toolkit是否安装成功，若有nvidia-container-toolkit相关软件信息说明安装成功
```

2. 执行如下命令重启 docker 服务使 NVIDIA Container Toolkit 生效。

```
sudo systemctl restart docker
```

使用 GPU 云服务器单机部署满血版 DeepSeek 模型

最近更新时间：2025-03-18 20:51:42

操作场景

DeepSeek-R1/V3 满血版模型参数规模为 671B（6710 亿参数），其模型权重已开源。本文将介绍如何使用 SGLang 在单台 PNV6 上进行推理服务部署。

操作步骤

实例环境准备

1. 参见 创建 NVIDIA GPU 实例 创建一台实例。其中：
   - **实例**：计算型 PNV6，类型选择 PNV6.32XLARGE1280 或 PNV6.96XLARGE2304，两规格均搭载 8 块 GPU 卡。
   - **镜像**：推荐使用 TencentOS Server 3.1 (TK4)。
   - **驱动**：推荐您使用 535 以上 GPU 驱动 + 12.4 CUDA。

POC测试-按量付费

1. 进入 CVM 购买页，**计费模式**选择**按量计费**、选择地域可用区，**架构**选择**异构计算**，**实例族**选择 GPU 机型，**类型**选择 8卡 PNV6 规格。
2. **镜像**选择 TencentOS Server 3.1 (TK4) 镜像，勾选**后台自动安装 GPU 驱动**。如下图所示：

3. 云盘规格按需选择，建议准备大于 800G 的存储空间。完成 VPC 网络、公网资源、安全组、实例名称、登录方式等配置后提交订单。

长期业务部署-包销/包月

正式购买阶段，需通过 高性能计算平台-工作空间 购买，计费模式为包年包月或包销计费，请联系腾讯云架构师开通使用。

2. 配置容器环境。

登录实例，执行以下命令配置容器环境：

```
curl -s -L http://mirrors.tencent.com/install/GPU/taco/get-nvidia-docker2.sh | sudo bash
```

3. 准备存储空间。
   - 方式一：使用 CFS Turbo 文件系统 挂载到实例内 /cfs 目录。
   - 方式二：如果使用云硬盘存储模型，建议准备大于 800G 的存储空间，需要 初始化云硬盘，将该磁盘挂载到实例内 /data 挂载点。

下载模型权重

1. 进入准备好的存储目录 /data 或 /cfs，根据需要选择命令下载模型权重：

```
# 上海地域 DeepSeek R1
wget https://haihub-model-1251001002.cos.ap-shanghai.myqcloud.com/DeepSeek-R1_1739186633750.zip
# 南京地域 DeepSeek R1
wget https://haihub-model-nj-1251001002.cos.ap-nanjing.myqcloud.com/DeepSeek-R1_1739186633750.zip

# 上海地域 DeepSeek V3
wget https://haihub-model-1251001002.cos.ap-shanghai.myqcloud.com/DeepSeek-V3_1739186651905.zip
# 南京地域 DeepSeek V3
wget https://haihub-model-nj-1251001002.cos.ap-nanjing.myqcloud.com/DeepSeek-V3_1739186651905.zip

# 可选，附加下载，上面四个必选其一下载
# 注意：当前 DeepSeek R1 权重为旧版本，可能无法触发思维链，如有强制思维链的需求，使用最新版 tokenizer.json 替换即可
```

```
wget https://haihub-model-1251001002.cos.ap-
shanghai.myqcloud.com/tokenizer_config.json
```

2. 解压下载好的模型权重：

```
UNZIP_DISABLE_ZIPBOMB_DETECTION=TRUE nohup unzip DeepSeek-
R1_xxxxxxxxxxxx.zip -d ./DeepSeek-R1 &
```

3. 解压完毕后，检查模型权重文件解压文件序号是否连续完整。

部署推理服务

1. 拉取推理容器镜像。

```
# -v 映射到容器里面的目录根据实际情况调整
# 将宿主机内下载地址 /data 或 /cfs 映射到 docker 目录

docker run \
    -itd \
    --gpus all \
    --privileged --cap-add=IPC_LOCK \
    --ulimit memlock=-1 --ulimit stack=67108864 \
    -v /data0:/data \
    -v /cfs:/cfs \
    --net=host \
    --ipc=host \
    --name=sglang_tencent
aicompute.tencentcloudcr.com/aibench/sglang:0.4.3post4_tencent
```

2. 进入容器镜像。

   运行以下命令，进入上一步拉起的容器镜像环境，如有基础环境问题请按照 镜像文档 排查。

```
docker exec -it sglang_tencent bash
```

3. 启动推理服务。

```
# --model-path 修改为模型权重所在地址
# --tp 8 单机部署为8
# 如果测试性能指标，可以使用 --disable-radix-cache 参数以禁用 prefill
cache，避免因为缓存命中率影响
```

```
nohup  python3 -m sglang.launch_server --model-path  /data/DeepSeek-R1
--tp 8 --disable-radix-cache --trust-remote-code --mem-fraction-static
0.9 --host 0.0.0.0 --port  30030  2>&1 > ds_infer_$(date
+'%Y%m%d_%H%M%S').log &
```

在当前路径查看日志文件 `ds_infer_$(date +'%Y%m%d_%H%M%S').log`，检查节点启动日志查看推理服务是否正常启动以及监听端口是否正常，若日志中有以下提示，则表明推理服务已正常部署：

```
[2025-02-09 17:33:14] INFO:     127.0.0.1:45712 - "POST /generate HTTP/1.1" 200 OK
[2025-02-09 17:33:14] The server is fired up and ready to roll!
```

模型互动体验

打开一个新的终端页面，向刚部署的推理服务发送请求：

```
# 若在其他服务器访问推理服务，127.0.0.1可以改成部署服务节点 eth0 对应 IP，访问端口
号需要与启动推理服务的端口号保持一致，推理服务使用的是30030
# model 需要指定为 server 的 model 名 /data0/DeepSeek-R1

# stream 参数开启决定是否流式输出，以下为流式输出交互命令
curl -X POST \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer ''" \
  -d '{
    "model": "/data0/DeepSeek-R1",
    "messages": [
      {"role": "user", "content": "你好！请帮我制定一个去大理游玩的计划！"}
    ],
    "temperature": 0.7,
    "stream": true
}' \
  "http://127.0.0.1:30030/v1/chat/completions"

# stream 参数开启决定是否流式输出，以下为非流式输出交互命令
curl -X POST \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer ''" \
  -d '{
    "model": "/data0/DeepSeek-R1",
    "messages": [
      {"role": "user", "content": "你好！请帮我制定一个去大理游玩的计划！"}
    ],
    "temperature": 0.7
```

版权所有：腾讯云计算（北京）有限责任公司

```
}' \
  "http://127.0.0.1:30030/v1/chat/completions"
```

若选择非流式输出，可以看到如下返回：

```
[root@VM-0-15-tencentos ~]# curl -X POST \
>   -H "Content-Type: application/json" \
>   -H "Authorization: Bearer ''" \
>   -d '{
>     "model": "/data0/DeepSeek-R1",
>     "messages": [
>       {"role": "user", "content": "你好！请帮我制定一个去大理游玩的计划！"}
>     ],
>     "temperature": 0.7
> }' \
>   "http://127.0.0.1:30030/v1/chat/completions"

{"id":"1213ba4eac69427899e84844a8484075","object":"chat.completion","created":1742194194,"model":"/data0/DeepSeek-R1","choices":[{"index":0,"message":{"role":"assistant","content":"<think>\n好的，用户需要一个大理游玩的计划。首先，我得考虑他们可能的旅行时间和天数，但用户没具体说明，可能需要默认建议，比如3天左右的行程。接下来，用户可能感兴趣的地方可能包括自然风光、文化体验和美食。大理的经典景点有洱海、苍山、大理古城、双廊、喜洲古镇等，这些都应该包含进去。\n\n还要考虑交通方式，大理的景点分布较广，建议租车或者电动车。住宿的话，大理古城和双廊是比较热门的区域，可以推荐不同价位的住宿选择。美食方面，乳扇、饵块、酸辣鱼这些当地特色要提到，同时推荐一些受欢迎的餐厅。\n\n用户可能还关心天气和穿着，大理昼夜温差大，需要提醒带外套和防晒用品。另外，是否有购物或者小众景点的需求？比如周城扎染或者沙溪古镇，如果时间允许可以加入。可能还需要考虑到游客的体力安排，比如环洱海骑行或徒步苍山，根据时间调整行程紧凑度。\n\n另外，用户可能有拍照打卡的需求，推荐一些拍照好看的地方，如碙溪村S弯、小普陀等。最后，注意事项比如防晒、高原反应、尊重当地文化这些也要提醒。可能需要分天数详细安排，并给出小贴士让计划更周全。\n</think>\n\n以下是一份详细的大理3-4天游玩计划，涵盖自然风光、人文体验和特色美食，供你参考：\n\n--- \n\n### 📅 **3天精华版行程**\n\n**Day 1：环洱海东线（建议包车/租车）**\n🔹上午：\n- 龙龛码头：看日出（夏季约6:30-7:00）\n- 碙溪村S弯：骑行洱海生态廊道（可租彩色自行车）\n- 喜洲古镇：转角楼拍照+稻田（季节限定）+喜洲破酥粑粑\n🔹下午：\n- 双廊古镇：杨丽萍太阳宫（需预🔹下午茶进入）/南诏风情岛\n- 挖色镇：打卡水上小树林（枯水期才有）\n- 小普陀：冬季喂海鸥最佳地\n- 文笔村：网红彩虹公路+海边咖啡馆\n🌙夜宿：双廊海景房（推荐明月松间/六阅）\n\n**Day 2：环洱海西线+大理古城**\n🔹上午：\n- 苍山感通索道：徒步玉带路（体力一般可选洗马潭索道）\n- 寂照庵：11:30前到吃斋饭（20元/人）\n🔹下午：\n- 大理古城：人民路逛文艺店铺+洋人街集市（周末开放）\n- 三塔倒影公园：远观崇圣寺三塔\n- 床单厂艺术区：文艺青年聚集地\n🌙夜晚：\n-
```

使用 Windows GPU 云服务器搭建深度学习环境

最近更新时间：2025-06-03 16:06:42

> **说明**
> 本文来自 GPU 云服务器用户实践征文，仅供学习和参考。

操作场景

本文介绍如何使用 Windows GPU 云服务器，通过云服务器控制台搭建深度学习环境。

实例环境

- **实例类型**：GN8.LARGE56
- **操作系统**：Windows Server 2019 数据中心版 64位 中文版
- **CPU**：Intel(R) Xeon(R) CPU E5-2680 v4 @2.40GHz 2.40GHz * 6vCPUs
- **RAM**：56GB
- **GPU**：Tesla P40 * 1
- **驱动及相关库、软件版本**：CUDA 10.2、Python 3.7、Pytorch 1.8.1、Tensorflow_gpu_2.2.0

选择驱动及相关库、软件版本

在安装驱动前，您需大致了解 CUDA、cuDNN、Pytorch、TensorFlow 及 Python 版本对应关系，以便根据实际配置选择适配版本，免除后续出现版本不匹配等问题。

选择 CUDA 驱动版本

CUDA（Compute Unified Device Architecture），是显卡厂商 NVIDIA 推出的运算平台。
CUDA™ 是一种由 NVIDIA 推出的通用并行计算架构，该架构使 GPU 能够解决复杂的计算问题。其包含了 CUDA 指令集架构（ISA）以及 GPU 内部的并行计算引擎。
1. 查看显卡算力

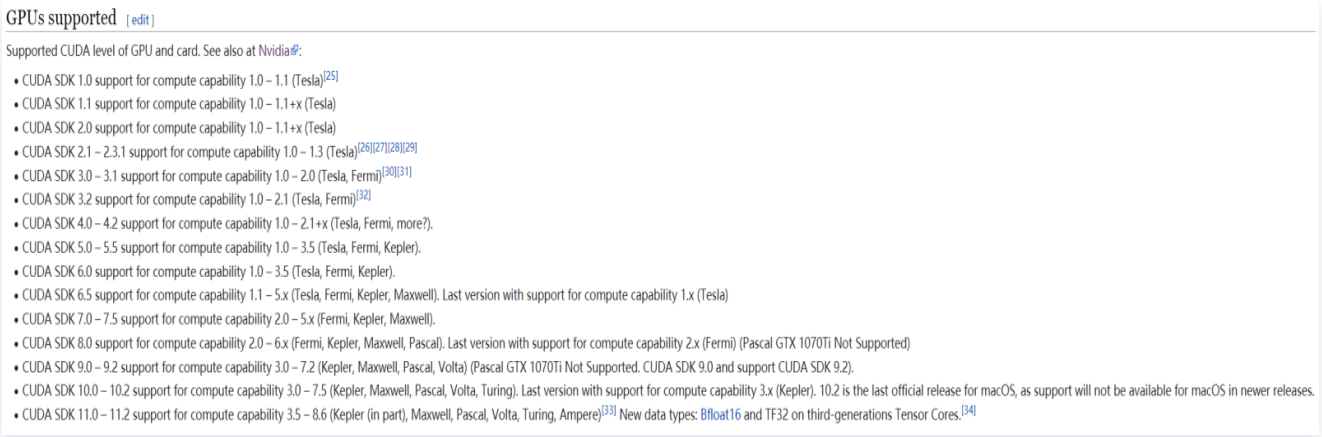
在选择 CUDA 驱动版本时，需先了解本文使用（Tesla P40）显卡的算力。可通过 NVIDIA 官网 查询 Tesla P40 显卡算力为6.1。如下图所示：

NVIDIA 数据中心产品

GPU	计算能力
NVIDIA A100	8.0
NVIDIA T4	7.5
NVIDIA V100	7.0
Tesla P100	6.0
Tesla P40	6.1
Tesla P4	6.1
Tesla M60	5.2
Tesla M40	5.2
Tesla K80	3.7
Tesla K40	3.5
Tesla K20	3.5
Tesla K10	3.0

2. 选择 CUDA 版本

如下图所示 CUDA 版本与显卡算力的关系，Tesla P40 显卡应选择8.0以上的 CUDA 版本。如需了解更多算力与 CUDA 版本信息，请参见 Application Compatibility on the NVIDIA Ampere GPU Architecture 。



GPUs supported [edit]

Supported CUDA level of GPU and card. See also at Nvidia:

- CUDA SDK 1.0 support for compute capability 1.0 – 1.1 (Tesla)[25]
- CUDA SDK 1.1 support for compute capability 1.0 – 1.1+x (Tesla)
- CUDA SDK 2.0 support for compute capability 1.0 – 1.1+x (Tesla)
- CUDA SDK 2.1 – 2.3.1 support for compute capability 1.0 – 1.3 (Tesla)[26][27][28][29]
- CUDA SDK 3.0 – 3.1 support for compute capability 1.0 – 2.0 (Tesla, Fermi)[30][31]
- CUDA SDK 3.2 support for compute capability 1.0 – 2.1 (Tesla, Fermi)[32]
- CUDA SDK 4.0 – 4.2 support for compute capability 1.0 – 2.1+x (Tesla, Fermi, more?).
- CUDA SDK 5.0 – 5.5 support for compute capability 1.0 – 3.5 (Tesla, Fermi, Kepler).
- CUDA SDK 6.0 support for compute capability 1.0 – 3.5 (Tesla, Fermi, Kepler).
- CUDA SDK 6.5 support for compute capability 1.1 – 5.x (Tesla, Fermi, Kepler, Maxwell). Last version with support for compute capability 1.x (Tesla)
- CUDA SDK 7.0 – 7.5 support for compute capability 2.0 – 5.x (Fermi, Kepler, Maxwell).
- CUDA SDK 8.0 support for compute capability 2.0 – 6.x (Fermi, Kepler, Maxwell, Pascal). Last version with support for compute capability 2.x (Fermi) (Pascal GTX 1070Ti Not Supported)
- CUDA SDK 9.0 – 9.2 support for compute capability 3.0 – 7.2 (Kepler, Maxwell, Pascal, Volta) (Pascal GTX 1070Ti Not Supported. CUDA SDK 9.0 and support CUDA SDK 9.2).
- CUDA SDK 10.0 – 10.2 support for compute capability 3.0 – 7.5 (Kepler, Maxwell, Pascal, Volta, Turing). Last version with support for compute capability 3.x (Kepler). 10.2 is the last official release for macOS, as support will not be available for macOS in newer releases.
- CUDA SDK 11.0 – 11.2 support for compute capability 3.5 – 8.6 (Kepler (in part), Maxwell, Pascal, Volta, Turing, Ampere)[33] New data types: Bfloat16 and TF32 on third-generations Tensor Cores.[34]

选择显卡驱动版本

确定 CUDA 版本后，再选择显卡驱动版本。您可参考如下图所示 CUDA 与驱动对应关系图进行选择。如需了解更多信息，请参见 cuda-toolkit-driver-versions 。

选择 cuDNN 版本

NVIDIA cuDNN 是用于深度神经网络的 GPU 加速库。其强调性能、易用性和低内存开销。NVIDIA cuDNN 可以集成到更高级别的机器学习框架中，例如谷歌的 Tensorflow、加州大学伯克利分校的流行 caffe 软件。简单的插入式设计可以让开发人员专注于设计和实现神经网络模型，而不是简单调整性能，同时还可以在 GPU 上实现高性能现代并行计算。
cuDNN 是基于 CUDA 的深度学习 GPU 加速库，有它才能在 GPU 上完成深度学习的计算。如需在 CUDA 上运行深度神经网络，需安装 cuDNN，才能使 GPU 进行深度神经网络的工作，工作速度相较 CPU 快很多。cuDNN 版本与 CUDA 版本的对应关系请参见 cuDNN Archive 。

选择 Pytorch 版本

您需根据 CUDA 版本，选择对应的 Pytorch 版本，匹配版本信息请参见 previous-versions 。

> **注意**
> CUDA 及 Pytorch 最新版本不一定是最佳选择，可能出现适配问题。建议在查阅版本适配信息后，选择合适的版本后再安装对应驱动。

选择 TesorFlow 版本

Tensorflow 较 Pytorch 稍复杂，它还需要 Python、编译器的版本支持。CPU、GPU 版本与 Python、CUDA、cuDNN 的版本对应关系如下：

- 基于 CPU 版本的 TensorFlow 版本
- 基于 GPU 版本的 TensorFlow 版本

操作步骤

创建实例

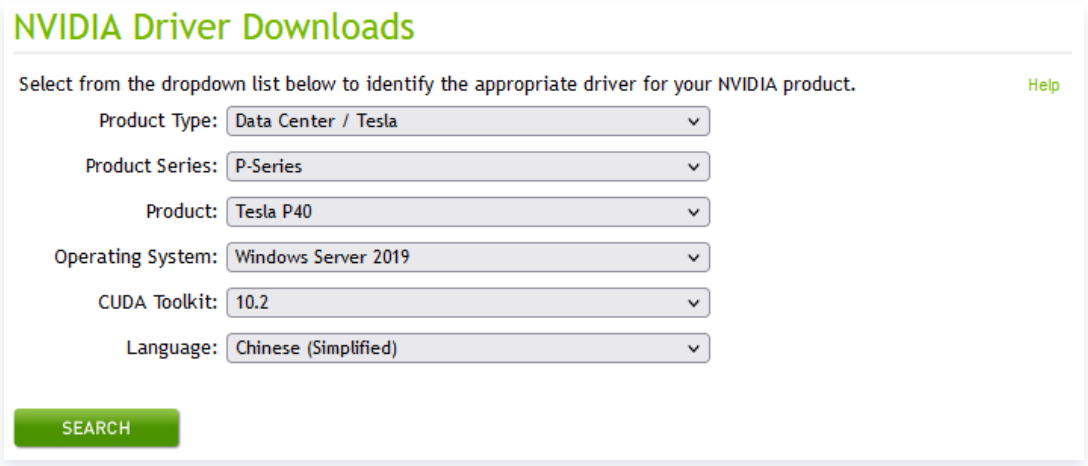
参见 购买 NVIDIA GPU 实例，创建 GPU 云服务器实例。
若您已具备 GPU 云服务器实例，则可参见 重装系统，重置已有实例的操作系统。

安装驱动、CUDA 及 cuDNN

安装显卡驱动

1. 参见 使用标准方式登录 Windows 实例（推荐），登录已创建的 GPU 云服务器。

2. 使用浏览器访问 NVIDIA 官网，并选择显卡的驱动版本。本文选择配置如下图所示：

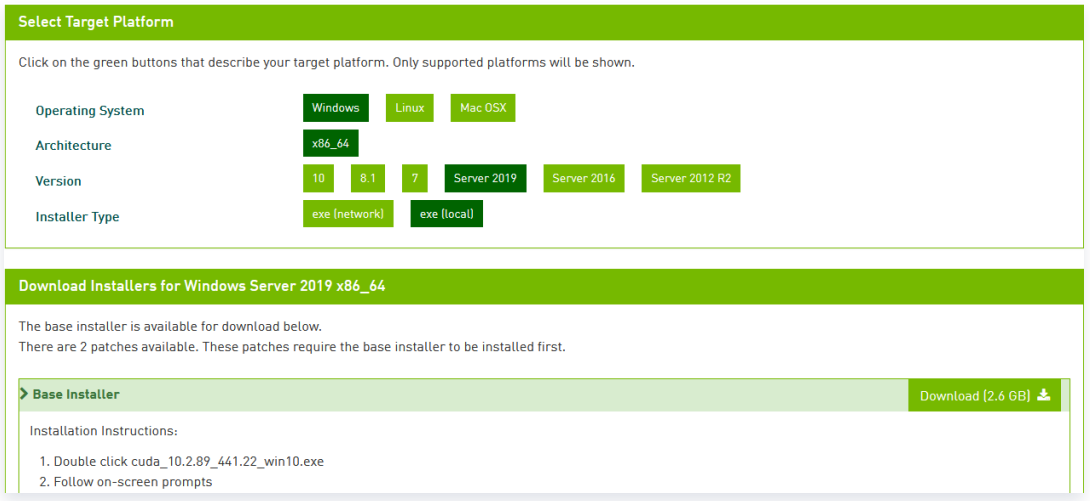


3. 选择 SEARCH 进入下载页面，单击**下载**即可。
   若您想通过下载至本地，再通过 FTP 上传至 GPU 云服务器，可参见 如何将本地文件拷贝到云服务器 。
4. 下载完成后，请双击安装包，根据页面提示完成安装。

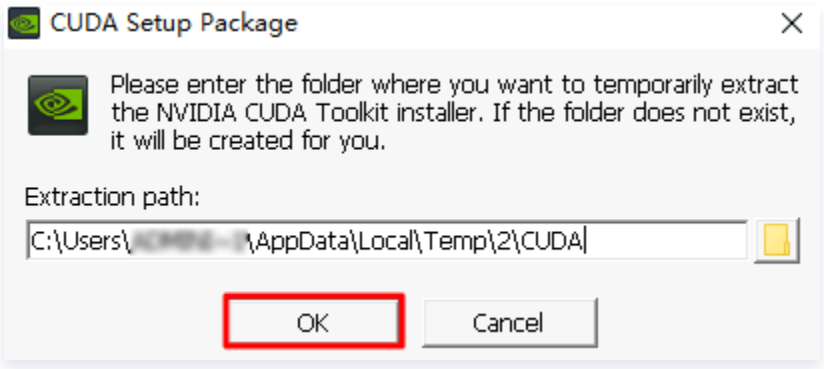
安装 CUDA

1. 进入 CUDA Toolkit Archive，选择对应版本。本文以下载10.2版本为例，如下图所示：

2. 进入 CUDA Toolkit 10.2 Download 页面，选择对应系统配置。本文选择配置如下图所示：



3. 单击 **Download**，开始下载。
4. 下载完成后，请双击安装包，并根据页面提示进行安装。其中，请注意以下步骤：
   - 在弹出的 **CUDA Setup Package** 窗口中，Extraction path 为暂时存放地址，无需修改，保持默认并单击 **OK**。如下图所示：



   - 在**许可协议**步骤中，选择**自定义**并单击**下一步**。如下图所示：

   根据实际需求选择安装组件，并单击**下一步**。如下图所示：

其余选项请根据页面提示，及实际需求进行选择，直至安装完毕。

配置环境变量

1. 在操作系统界面，右键单击左下角的  ，在弹出菜单中选择**运行**。
2. 在**运行**窗口中输入 `sysdm.cpl` ，并单击**确定**。

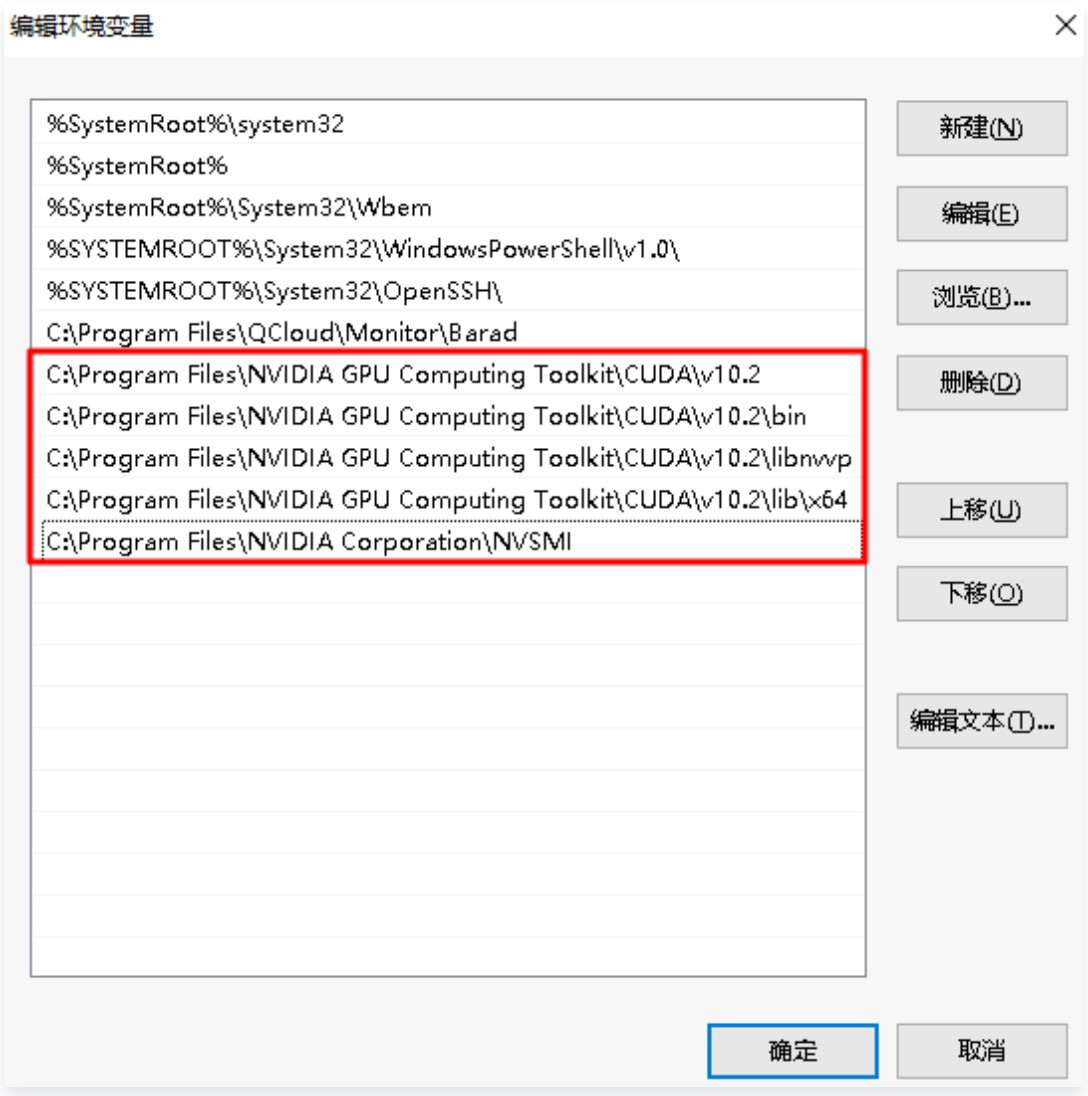
3. 在打开的**系统属性**窗口中，选择**高级**页签，并单击**环境变量**。如下图所示：

4. 选择系统变量**中的 Path**，单击**编辑**。
5. 在弹出的**编辑环境变量**窗口中，新建并输入如下环境变量配置。

```
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.2
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.2\bin
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.2\libnvvp
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.2\lib\x64
C:\Program Files\NVIDIA Corporation\NVSMI
```

版权所有：腾讯云计算（北京）有限责任公司

编辑完成后如下图所示：



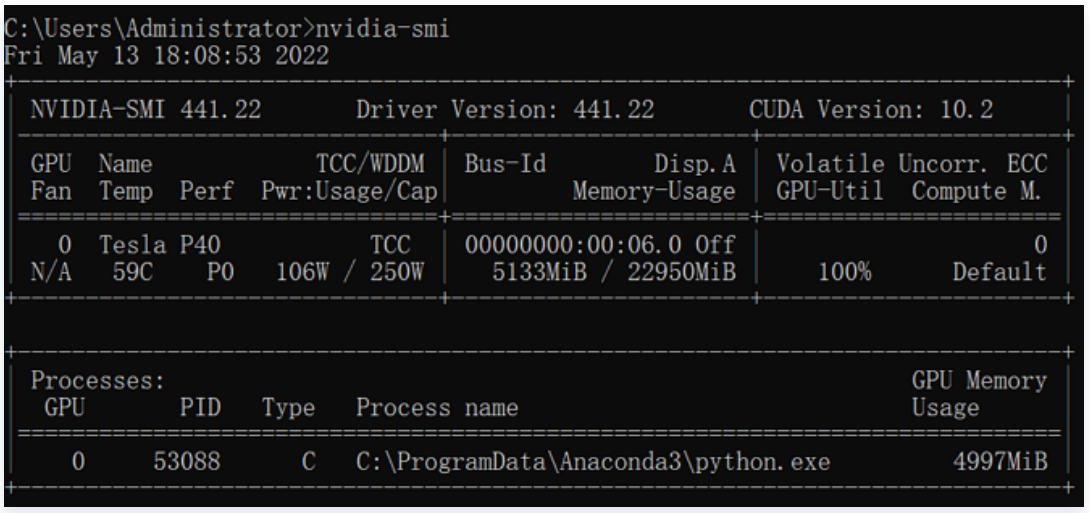
6. 连续单击3次**确定**，保存设置。

检查显卡驱动及 CUDA

1. 在操作系统界面，右键单击左下角的 ，在弹出菜单中选择**运行**。
2. 在**运行**窗口中输入 `cmd` ，并单击**确定**。
3. 在 cmd 窗口中：
   - 执行以下命令，检查显卡驱动是否安装成功。

```
nvidia-smi
```

返回如下图所示界面表示显卡驱动安装成功。下图为正在运行中的 GPU，在 GPU 运行时，该命令可查看 GPU 的使用情况。



- 执行以下命令，检查 CUDA 是否安装成功。

```
nvcc -V
```

返回如下图所示界面表示 CUDA 安装成功。

安装 cuDNN

1. 前往 cuDNN Download 页面，单击 **Archived cuDNN Releases** 查看更多版本。
2. 找到所需 cuDNN 版本，并下载。
3. 解压 cuDNN 压缩包，并将 `bin`、`include` 及 `lib` 文件夹拷贝至 `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.2` 目录下。
4. 至此已完成 cuDNN 安装。

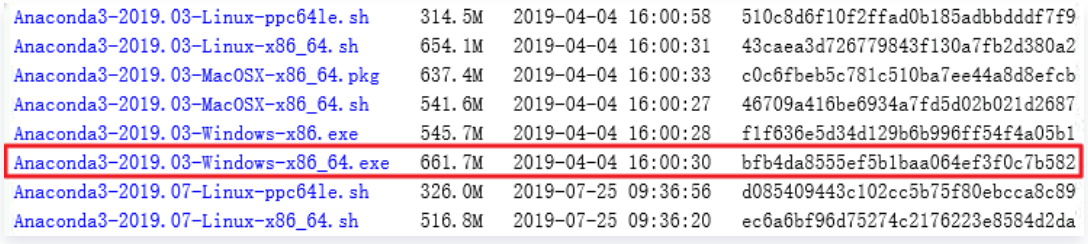
安装深度学习库

安装 Anaconda

建议通过 Anaconda 创建的虚拟环境安装 Pytorch 和 Tensorflow。通过 Anaconda，可便捷获取包并对包进行管理，同时可统一管理环境。Anaconda 包含了 conda、Python 在内的超过180个科学包及其依赖项，安装过程简单，能高性能使用 Python 和 R 语言，且有免费的社区支持。

1. 前往 Anaconda 官网，拉至页面底部，选择 **archive** 查看更多版本。

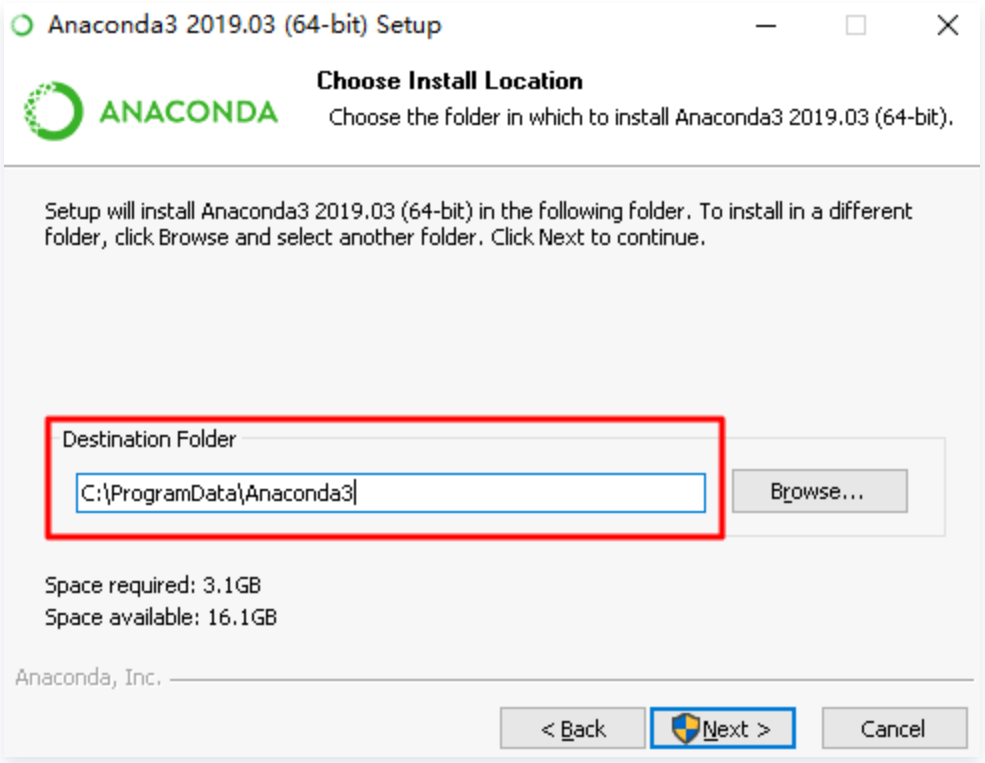
2. 在页面中下载所需版本，本文以下载 `Anaconda3-2019.03-Windows-x86_64` 为例。如下图所示：



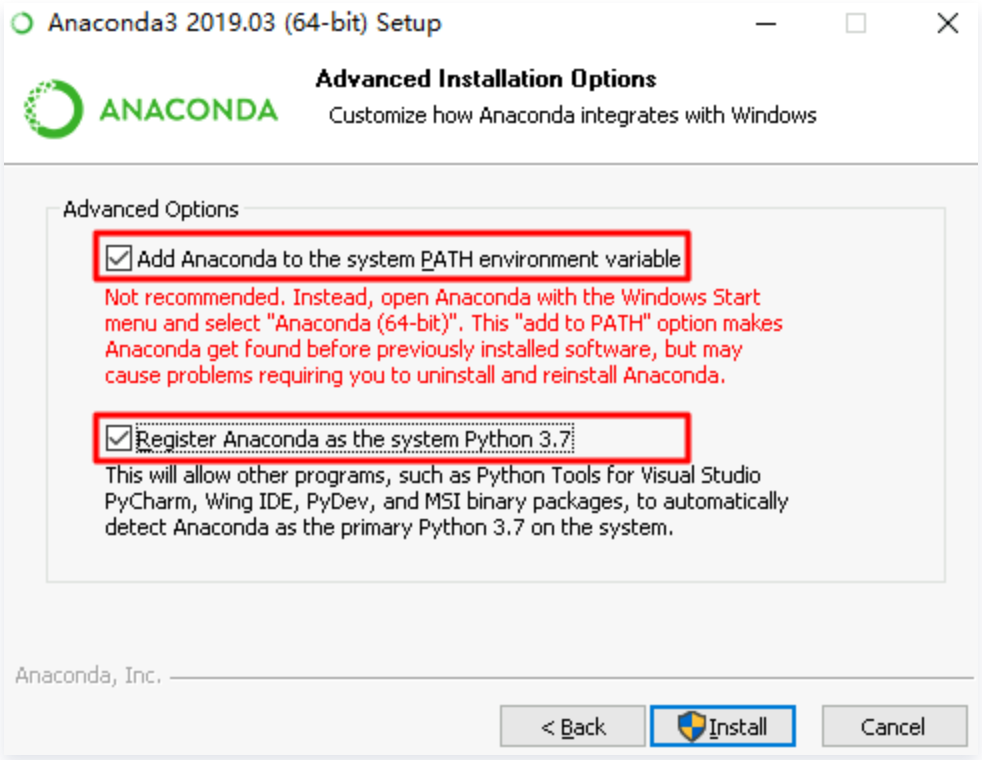
Anaconda3-2019.03-Linux-ppc64le.sh	314.5M	2019-04-04 16:00:58	510c8d6f10f2ffad0b185adbbdddf7f9
Anaconda3-2019.03-Linux-x86_64.sh	654.1M	2019-04-04 16:00:31	43caea3d726779843f130a7fb2d380a2
Anaconda3-2019.03-MacOSX-x86_64.pkg	637.4M	2019-04-04 16:00:33	c0c6fbeb5c781c510ba7ee44a8d8efcb
Anaconda3-2019.03-MacOSX-x86_64.sh	541.6M	2019-04-04 16:00:27	46709a416be6934a7fd5d02b021d2687
Anaconda3-2019.03-Windows-x86.exe	545.7M	2019-04-04 16:00:28	f1f636e5d34d129b6b996ff54f4a05b1
Anaconda3-2019.03-Windows-x86_64.exe	661.7M	2019-04-04 16:00:30	bfb4da8555ef5b1baa064ef3f0c7b582
Anaconda3-2019.07-Linux-ppc64le.sh	326.0M	2019-07-25 09:36:56	d085409443c102cc5b75f80ebcca8c89
Anaconda3-2019.07-Linux-x86_64.sh	516.8M	2019-07-25 09:36:20	ec6a6bf96d75274c2176223e8584d2da

3. 请双击安装包，并根据页面提示进行安装。其中，请注意以下步骤：

- 在 **Choose Install Location** 步骤中，更改默认安装路径。因默认安装路径 C 盘中的 ProgramData 文件夹为隐藏文件夹，为了方便管理，建议安装在其他文件夹。下图所示为默认安装路径：



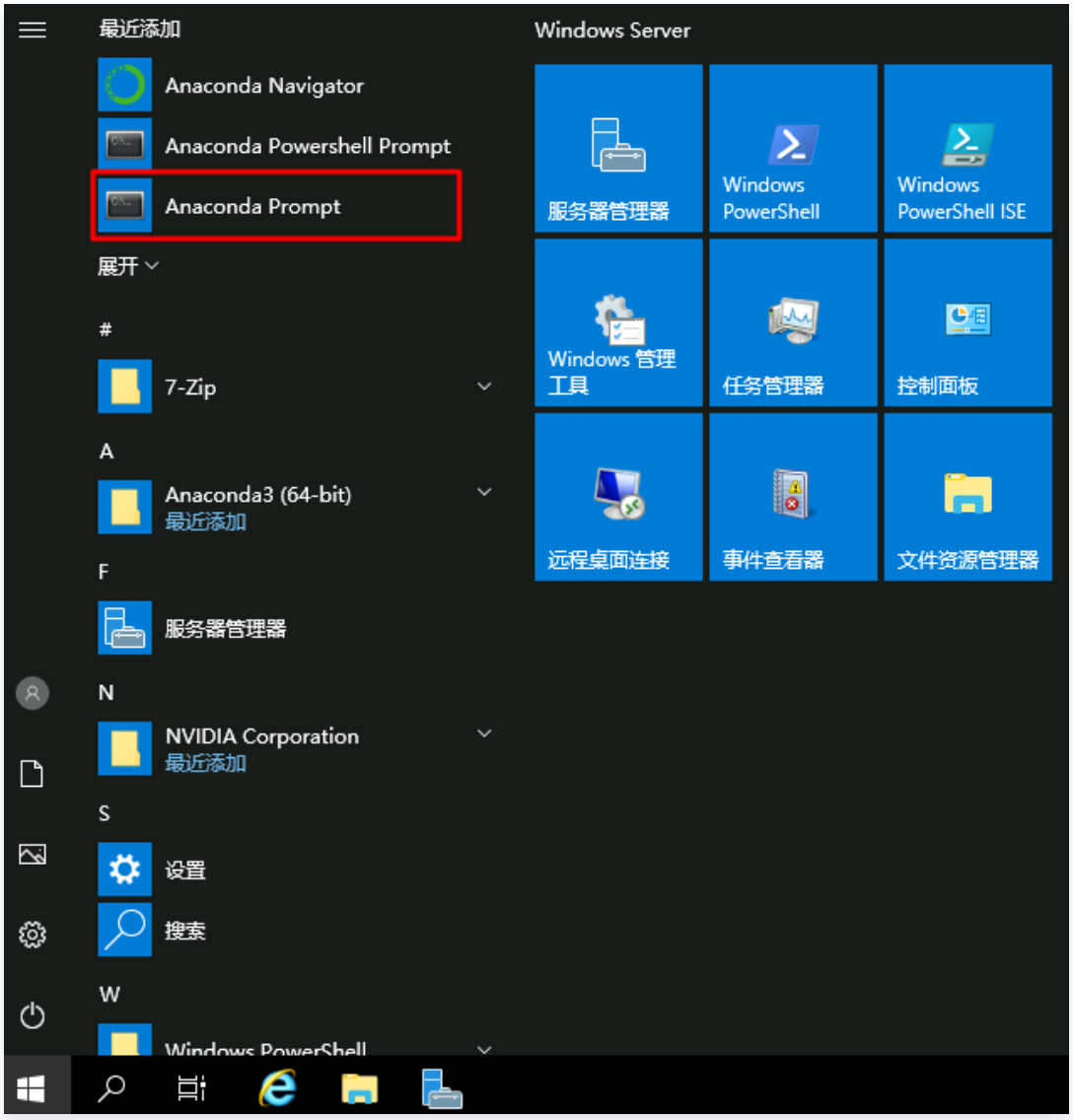
- 在 **Advanced Installation Options** 步骤中，勾选全部选项，表示将 Anaconda 安装路径添加至环境变量，并将 Python 3.7 作为解释器。如下图所示：



4. 单击 Install 等待完成安装。

配置 Anaconda

1. 在操作系统界面，单击左下角的 ，在弹出菜单中选择 **Anaconda Prompt**。如下图所示：

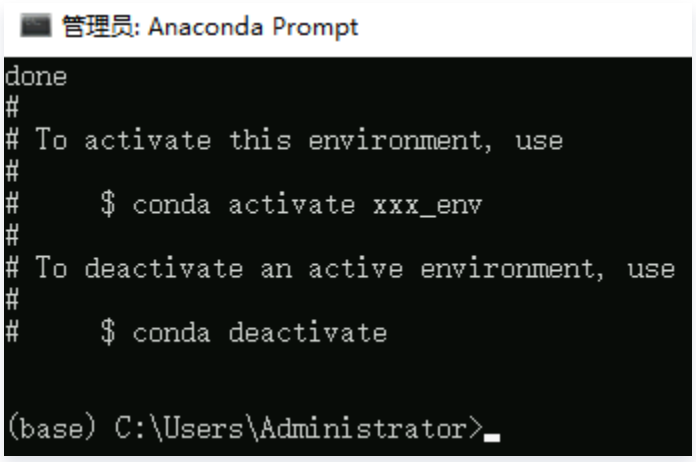


2. 在打开的 **Anaconda Prompt** 命令行窗口中，执行以下命令，创建虚拟环境。

```
conda create -n xxx_env python=3.7
```

> **说明**
>
> `xxx_env` 为环境名，`python=3.7` 为 Python 版本，您可根据实际需求进行修改。

创建成功即如下图所示：



您可使用以下命令进入或退出已创建的虚拟环境。进入虚拟环境后，即可按照实际需求安装包。

```
#激活命令
conda activate xxx_env
#退出命令
conda deactivate
```

安装 Pytorch

前往 Pytorch 官网，使用官网推荐的安装代码。
本文已安装 CUDA 版本为10.2，并选择 pip 安装方式，则在已创建的 `xxx_env` 虚拟环境中执行如下命令进行安装：

```
# CUDA 10.2
pip install torch==1.8.1+cu102 torchvision==0.9.1+cu102
torchaudio==0.8.1 -f https://download.pytorch.org/whl/torch_stable.html
```

可通过替换源，加快安装速度，替换为清华源后则执行如下命令：

```
# CUDA 10.2
pip install torch==1.8.1+cu102 torchvision==0.9.1+cu102
torchaudio==0.8.1 -f https://download.pytorch.org/whl/torch_stable.html
-i https://pypi.tuna.tsinghua.edu.cn/simple
```

安装 Tensorflow

执行以下命令，安装 Tensorflow_gpu_2.2.0。

```
pip install tensorflow-gpu==2.2.0 -i
https://pypi.tuna.tsinghua.edu.cn/simple
```

执行以下命令，安装 keras。

```
pip install keras -i https://pypi.tuna.tsinghua.edu.cn/simple
```

至此，已完成了基本深度学习库的安装。您可参考本文方法安装更多所需要的包，并利用 Anaconda 自带的 jupyter notebook、Spyder 工具或者安装 PyCharm 等工具开始代码学习！

使用视频增型实例 GN7vi 实现视频画质增强

最近更新时间：2025-06-09 18:06:32

操作场景

本文介绍如何在视频增强型实例 GN7vi 服务器上进行视频编解码和 AI 画质增强。视频增强型实例 GN7vi 提供了视频编解码功能和 AI 画质增强功能，使用方式和开源 FFmpeg 完全兼容，您可以参考本文完成视频画质处理。

操作步骤

实例环境准备

参见 创建 NVIDIA GPU 实例 创建一台实例。其中：

- **实例**：根据对 GPU 和内核数量的需求，参见 视频增强型 GN7vi 选择。
- **镜像**：可参考表格中的 可用镜像 进行选择。
- **驱动**：CUDA 及 cuDNN 的自动安装并非本次部署的必选项，您可根据实际情况选择。可选择如下安装方式：

创建实例时自动安装

部分镜像支持自动安装驱动，通过勾选**后台自动安装 GPU 驱动**进行安装。如下图所示：

若您所选的镜像版本不支持自动安装驱动，可参考手动安装步骤登录实例进行安装。

创建实例后手动安装

1. 参见 使用标准方式登录 Windows 实例，登录已创建的 GPU 云服务器。
2. 执行以下命令，实现 GPU 驱动、CUDA、cuDNN 的自动安装。

> **说明：**
> 环境版本配置为 GPU 驱动460.106.00、CUDA11.2.2和 cuDNN8.2.1。若您有其他版本需要，可通过 联系我们 咨询。

```
wget https://gpu-related-scripts-1251783334.cos.ap-guangzhou.myqcloud.com/gpu-auto-install/gpu_auto_install_220823.sh && wget https://gpu-related-scripts-1251783334.cos.ap-guangzhou.myqcloud.com/gpu-auto-install/driver460_cuda11.2.2.txt && sudo bash ./gpu_auto_install_220823.sh install --config_file=./driver460_cuda11.2.2.txt && source /etc/bash.bashrc && source ${HOME}/.bashrc
```

文件总览

依次执行以下命令，进入 tscsdk-center 查看当前目录下的所有文件。

```
cd /usr/local/qcloud/tscsdk-center
```

```
ls -l
```

返回结果如下图所示：

```
drwxr-xr-x 2 root root     4096 Aug 26 10:42 fflib_gpu
-rwxr-xr-x 1 root root 20532592 Aug 26 10:01 ffmpeg
drwxr-xr-x 3 root root     4096 Aug 26 17:55 tenmodel
drwxr-xr-x 2 root root     4096 Aug 29 17:28 videos
```

说明如下：

文件名	说明
fflib_gpu	画质处理程序的运行依赖库。
ffmpeg	已经嵌入画质处理功能的 ffmpeg 程序。
tenmodel	画质处理使用的 AI 模型。

videos	内置的样例视频。

开始使用

1. 依次执行以下命令，设置环境变量。

```
cd /usr/local/qcloud/tscsdk-center
```

```
export LD_LIBRARY_PATH=./fflib_gpu:$LD_LIBRARY_PATH
```

2. 在 tscsdk-center 目录下，依次执行以下命令，生成经画质处理之后的样例输出视频。

- 低清视频处理：低清视频一般指分辨率小于或等于720p的视频，该命令使用了 tenfilter 中均衡模式下的标准超分辨率模型以及 unsharp 锐化处理。

```
./ffmpeg -i ./videos/input1.mp4 -vf
tenfilter=mag_filter=1:mag_sr=2:mag_sr_stre=balance,unsharp -c:v
libten264 -ten264opts crf=26:vbv-maxrate=2000 -y output1.mp4
```

- 高清视频处理：高清视频一般指分辨率大于720p的视频。该命令使用了 tenfilter 中的高质量超分辨率模型。

```
./ffmpeg -i ./videos/input2.mp4 -vf tenfilter=mag_srgan=1 -c:v
libten264 -ten264opts crf=26:vbv-maxrate=2000 -y output2.mp4
```

- 快速处理模型：以下命令使用了 tenfilter 中的去压缩伪影、正常模式下的标准超分辨率模型以及 unsharp 锐化处理。

```
./ffmpeg -i ./videos/input1.mp4 -vf
tenfilter=af=auto,tenfilter=mag_filter=1:mag_sr=2:mag_sr_stre=normal,u
nsharp -c:v libten264 -ten264-params crf=26:vbv-maxrate=2000 -y
fast_output1.mp4
```

```
./ffmpeg -i ./videos/input2.mp4 -vf
tenfilter=af=auto,tenfilter=mag_filter=1:mag_sr=2:mag_sr_stre=normal,u
nsharp -c:v libten264 -ten264-params crf=26:vbv-maxrate=2000 -y
fast_output2.mp4
```

> **说明：**

> 同一模型的运行速度会受到输入分辨率的影响，分辨率越大运行速度越慢。初次运行特定 AI 模型时，会耗费较长的时间初始化模型，后续重复执行相同命令时速度会有明显提升。如需评估运行速度，请参考重复执行时的结果。

ffmpeg 命令中的部分参数含义如下表：

参数内容	参数含义
`-i videos/input1.mp4`	指定输入视频文件。
`-vf tenfilter=mag_srgan=1`	指定视频处理滤镜（filter）图，各参数含义请参见 视频处理 AI 模型功能清单 。
`-c:v libten264`	指定编码器为腾讯自研的 Ten264 或 Ten265 视频编码器。
`-ten264opts crf=26:vbv-maxrate=2000`	设置视频编码器的相关参数，各参数含义请参见 视频编码器功能清单 。
`-y output.mp4`	指定输出视频文件，自动覆盖已有文件。

3. 等待程序运行结束后，可将输出视频文件下载到本地进行查看，建议使用 xshell、MobaXterm 等工具。以下为执行上述命令输出的4个视频文件截图，以用作对比验证。

output1.mp4

截图时间：01分15秒

fast_output1.mp4

截图时间：01分15秒

output2.mp4

截图时间：00分10秒

fast_output2.mp4

截图时间：00分10秒

功能清单

tscsdk-center 包含 视频处理 AI 模型 和 腾讯自研视频编码器 两部分视频能力的支持。
视频处理 AI 模型利用 FFmpeg 的滤镜（filter）机制进行集成，使相关 filter 能够将 AI 推理能力嵌入到视频编解码和处理流程中，提升硬件设备的利用效率和吞吐能力，结合腾讯自研的视频编码器，可以在视频画质增强的基础上带来更高的视频编码压缩率。

视频处理 AI 模型功能清单

视频处理 AI 模型全部集成在一个名为 tenfilter 的滤镜中。通过 `"-vf tenfilter=name1=value1:name2=value2"` 的方式实现调用和配置。在单个 tenfilter 中可以开启一个 AI 模型，多个 tenfilter 之间可以任意组合。
所有 AI 模型的详细描述如下表：

模型或功能名称	参数	使用示例
通用参数	• mdir：模型配置文件路径。默认值为 "./tenmodel/tve-conf.json"。 • gpu：tenfilter 所用的 GPU 编号。	tenfilter=mdir=./tenmodel/tve-conf.json:gpu=1

去压缩伪影	af：去压缩伪影强度。当前仅支持 auto。	tenfilter=af=auto
人脸保护	• face_protect_enable：为1时开启人脸保护逻辑。 • face_af_ratio：人脸区域去噪弱化系数。 • face_sp_ratio：人脸区域锐化强化系数。	tenfilter=face_protect_enable=1:face_af_ratio=0.5:face_sp_ratio=0.5
视频插帧	• mag_fps：为1时开启视频插帧。 • fps：目标帧率。	tenfilter=mag_fps=1:fps=60
色彩增强	• mag_filter：需要设置为1。 • cebb：为1时开启色彩增强。	tenfilter=mag_filter=1:cebb=1
标准超分辨率	• mag_filter：需要设置为1。 • mag_sr：超分辨率比率，当前仅支持2倍超分辨率。 • mag_sr_stre：超分辨率模式。可设置为正常模式（normal）或均衡模式（balance）。	tenfilter=mag_filter=1:mag_sr=2:mag_sr_stre=normal
高质量超分辨率	mag_srgan：为1时开启高质量超分辨率模型。	tenfilter=mag_srgan=1
视频去噪声	• mag_filter：需要设置为1。 • dn：去噪声强度，当前仅支持3。	tenfilter=mag_filter=1:dn=3
视频画质增强人脸增强字体增强（支持多模型）	• mag_filter：需要设置为1。 • eh：为1时开启画质增强。 • faceeh：为1时开启人脸增强。 • fonteh：为1时开启字体增强。 • prior：AI 模型执行优先级，例如 "faceeh-eh-fonteh"，需同时开启对应模型。添加 "-parally" 时开启并行优化。	• 单模型：tenfilter=mag_filter=1:eh=1 • 多模型：tenfilter=mag_filter=1:eh=1:faceeh=1:prior=faceeh-eh-parally

腾讯自研视频编码器

tscsdk-center 包含 Ten264 和 Ten265 两个腾讯自研的视频编码器。编码器类型以及编码器参数可以在视频处理时通过命令参数进行设置。
每种编码器的指定和设置方式如下：

编码器名称	指定方式	设置方式
Ten264	-vcodec libten264-c:v libten264	-ten264opts name1=value1:name2=value2
Ten265	-vcodec libten265-c:v libten265	-ten265-params name1=value1:name2=value2

每种编码器的具体参数及其详细说明如下：

Ten264 编码器

参数名称	参数说明
preset	指定编码器编码参数集合的配置。0："ultrafast"，1："superfast"，2："veryfast"，3："faster"，4："fast"，5："medium"，6："slow"，7："slower"，8："veryslow"，9："placebo"。
bitrate	ABR 模式下面的输出视频的码率。
crf	CRF 模式下面的 CRF 数值。
aq-mode	0：关闭 aqmode，1：开启 aqmode，2：基于方差的aqmode，3：基于方差的 aqmode 并且偏向于暗场景。默认2会产生更好的 ssim 结果。
vbv-maxrate	vbv 的最大码率。默认情况下该数值和配置的码率相同。
vbv-bufsize	vbv 缓冲 buffer 的大小。默认情况下该数值是配置的码率的四倍。
rc-lookahead	lookahead 长度。
scenecut	是否开启场景切换。默认开启，一般不建议关闭。
keyint	关键帧最长间隔。默认是256，可以根据实际业务情况配置，一般配置为2 – 5s的时间间隔的帧数。
threads	使用的线程池的线程数。
lookahead-	预分析，lookahead 使用线程数。

threads	
profile	"baseline"，"main"，"high"，"high422"，"high444"。

Ten265 编码器

参数名称	参数说明
preset	指定编码器编码参数集合的配置。-1：ripping，0：placebo，1：veryslow，2：slower，3：slow，4：universal，5：medium，6：fast，7：faster，8：veryfast，9：superfast。
rc	码率控制方式。0：CQP，1：ABR_VBV，2：ABR，3：CRF_VBV，4：CRF。
bitrate	ABR 模式下面的输出视频的码率。
crf	CRF 模式下面的 CRF 数值，取值范围：[1,51]。
aq-mode	0：关闭 aqmode，1：开启 aqmode，2：基于方差的 aqmode，3：基于方差的 aqmode 并且偏向于暗场景。默认2会产生更好的 ssim 结果。
vbv-maxrate	vbv 的最大码率。默认情况下该数值和配置的码率相同。
vbv-bufsize	vbv 缓冲 buffer 的大小。默认情况下该数值是配置的码率的四倍。
rc-lookahead	lookahead 长度。
scenecut	场景切换阈值，取值范围[0,100]。0表示关闭。默认开启，一般不建议关闭。
open-gop	是否开启 open gop。0：关闭，1：开启。默认开启，在直播场景中为了支持随机接入，建议关闭这个功能。
keyint	关键帧最长间隔。默认是256，可以根据实际业务情况配置，需要大于50且是8的倍数。
ltr	是否要支持长期参考帧 。0：关闭，1:：开启。默认开启，如果播放 HEVC 视频的硬件设备比较差，建议关闭这个功能。
pool-	WPP 使用的线程池的线程数。默认和 CPU 的核数相同，如果想减少 CPU 占用，可以

threads	降低这个数目。

使用建议

- tscsdk-center 支持每个 AI 模型的灵活控制，如果有特殊的需求和应用场景，可以通过设置每项功能的开关和组合顺序，以达到更好的适配和视频处理效果。
- tscsdk-center 提供了两种超分辨率模型。其中，标准超分辨率模型相对适合于分辨率较低的老片源，高质量超分辨率模型相对适合于高清片源，建议结合片源类型对这两种超分辨率模型的实际效果进行评估。
- tscsdk-center 提供 AI 模型需要运行在 GPU 上，而视频编码器只运行在 CPU 上。大部分情况下，在 GPU 算力跑满后，CPU 还会存在一定的空闲。此时可以合理安排一些只需运行在 CPU 上的视频处理任务（例如视频转码）以充分利用各类硬件资源。

版权所有：腾讯云计算（北京）有限责任公司

使用 Docker 安装 TensorFlow 并设置 GPU/CPU 支持

最近更新时间：2025-06-03 16:06:42

> **说明**
> 本文来自 GPU 云服务器用户实践征文，仅供学习和参考。

操作场景

您可通过 Docker 快速在 GPU 实例上运行 TensorFlow，且该方式仅需实例已安装 NVIDIA® 驱动程序，无需安装 NVIDIA® CUDA® 工具包。
本文介绍如何在 GPU 云服务器上，使用 Docker 安装 TensorFlow 并设置 GPU/CPU 支持。

说明事项

- 本文操作步骤以 Ubuntu 20.04 操作系统的 GPU 云服务器为例。
- 您的 GPU 云服务器实例需已安装 GPU 驱动。

> **说明**
> 建议使用公共镜像创建 GPU 云服务器。若选择公共镜像，则勾选**后台自动安装 GPU 驱动**即可预装相应版本驱动。该方式仅支持部分 Linux 公共镜像，详情请参见 各实例支持的 GPU 驱动版本及安装方式 。

操作步骤

安装 Docker

1. 登录实例，依次执行以下命令，安装所需系统工具。

```
sudo apt-get update
```

```
sudo apt-get install \
ca-certificates \
curl \
gnupg \
lsb-release
```

2. 执行以下命令，安装 GPG 证书，写入软件源信息。

```
sudo mkdir -p /etc/apt/keyrings
 curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
dearmor -o /etc/apt/keyrings/docker.gpg
 echo \
  "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null
```

3. 依次执行以下命令，更新并安装 Docker-CE。

```
sudo apt-get update
```

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-
compose-plugin
```

安装 TensorFlow

设置 NVIDIA 容器工具包

1. 执行以下命令，设置包存储库和 GPG 密钥。详细信息请参见 Setting up NVIDIA Container Toolkit 。

```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
   && curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey |
sudo gpg --dearmor -o /usr/share/keyrings/nvidia-container-toolkit-
keyring.gpg \
   && curl -s -L https://nvidia.github.io/libnvidia-
container/$distribution/libnvidia-container.list | \
         sed 's#deb https://#deb [signed-
by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg]
https://#g' | \
         sudo tee /etc/apt/sources.list.d/nvidia-container-
toolkit.list
```

2. 执行以下命令，安装 nvidia-docker2 包及依赖项。

```
sudo apt-get update
```

```
sudo apt-get install -y nvidia-docker2
```

3. 执行以下命令，设置默认运行时重启 Docker 守护进程完成安装。

```
sudo systemctl restart docker
```

4. 此时可执行以下命令，通过运行基本 CUDA 容器来测试工作设置。

```
sudo docker run --rm --gpus all nvidia/cuda:11.0.3-base-ubuntu20.04
nvidia-smi
```

返回结果如下所示：

```
+-----------------------------------------------------------------------
--------+
| NVIDIA-SMI 450.51.06    Driver Version: 450.51.06    CUDA Version:
11.0     |
|-------------------------------+----------------------+--------------
--------+
| 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  Tesla T4            On   | 00000000:00:1E.0 Off |
0 |
| N/A   34C    P8     9W /  70W |      0MiB / 15109MiB |      0%
Default |
|                               |                      |
N/A |
+-------------------------------+----------------------+--------------
--------+

+-----------------------------------------------------------------------
--------+
| Processes:
|
```

```
|  GPU   GI   CI        PID   Type   Process name                  GPU
Memory |
|        ID   ID
Usage      |
|=============================================================================
========|
|  No running processes found
|
+-----------------------------------------------------------------------------
--------+
```

下载 TensorFlow Docker 镜像

官方 TensorFlow Docker 镜像位于 tensorflow/tensorflow Docker Hub 代码库中。镜像版本按照以下格式进行标记：

标记	说明
`latest`	TensorFlow CPU 二进制镜像的最新版本。（默认版本）
`nightly`	TensorFlow 镜像的每夜版。（不稳定）
`version`	指定 TensorFlow 二进制镜像的版本，例如 2.1.0。
`devel`	TensorFlow master 开发环境的每夜版。包含 TensorFlow 源代码。
`custom-op`	用于开发 TensorFlow 自定义操作的特殊实验性镜像，详情请参见 tensorflow/custom-op 。

每个基本标记都有会添加或更改功能的变体：

标记变体	说明
`tag-gpu`	支持 GPU 的指定标记版本。
`tag-jupyter`	针对 Jupyter 的指定标记版本（包含 TensorFlow 教程笔记本）。

您可以一次使用多个变体。例如，以下命令会将 TensorFlow 版本镜像下载到计算机上：

```
docker pull tensorflow/tensorflow                   # latest stable
release
docker pull tensorflow/tensorflow:devel-gpu         # nightly dev
release w/ GPU support
```

```
docker pull tensorflow/tensorflow:latest-gpu-jupyter  # latest release
w/ GPU support and Jupyter
```

启动 TensorFlow Docker 容器

启动配置 TensorFlow 的容器，请使用以下命令格式。

```
docker run [-it] [--rm] [-p hostPort:containerPort]
tensorflow/tensorflow[:tag] [command]
```

示例

使用仅支持 CPU 的镜像的示例

如下所示，使用带 `latest` 标记的镜像验证 TensorFlow 安装效果。Docker 会在首次运行时下载新的 TensorFlow 镜像：

```
docker run -it --rm tensorflow/tensorflow \
   python -c "import tensorflow as tf;
print(tf.reduce_sum(tf.random.normal([1000, 1000])))"
```

其他 TensorFlow Docker 方案示例如下：

- 在配置 TensorFlow 的容器中启动 `bash` shell 会话：

  ```
  docker run -it tensorflow/tensorflow bash
  ```

- 如需在容器内运行在主机上开发的 TensorFlow 程序，请通过 `-v hostDir:containerDir -w workDir` 参数，装载主机目录并更改容器的工作目录。示例如下：

  ```
  docker run -it --rm -v $PWD:/tmp -w /tmp tensorflow/tensorflow python
  ./script.py
  ```

  > **说明**
  > 向主机公开在容器中创建的文件时，可能会出现权限问题。通常情况下，最好修改主机系统上的文件。

- 使用 nightly 版 TensorFlow 启动 Jupyter 笔记本服务器：

  ```
  docker run -it -p 8888:8888 tensorflow/tensorflow:nightly-jupyter
  ```

请参考 Jupyter 官网 相关说明，使用浏览器访问 `http://127.0.0.1:8888/?token=...` 。

使用支持 GPU 的镜像的示例

执行以下命令，下载并运行支持 GPU 的 TensorFlow 镜像。

```
docker run --gpus all -it --rm tensorflow/tensorflow:latest-gpu \
   python -c "import tensorflow as tf;
print(tf.reduce_sum(tf.random.normal([1000, 1000])))"
```

设置支持 GPU 镜像可能需要一段时间。如果重复运行基于 GPU 的脚本，您可以使用 `docker exec` 重复使用容器。
执行以下命令，使用最新的 TensorFlow GPU 镜像在容器中启动 `bash` shell 会话：

```
docker run --gpus all -it tensorflow/tensorflow:latest-gpu bash
```

使用 GPU 云服务器训练 ViT 模型

最近更新时间：2025-06-03 16:06:42

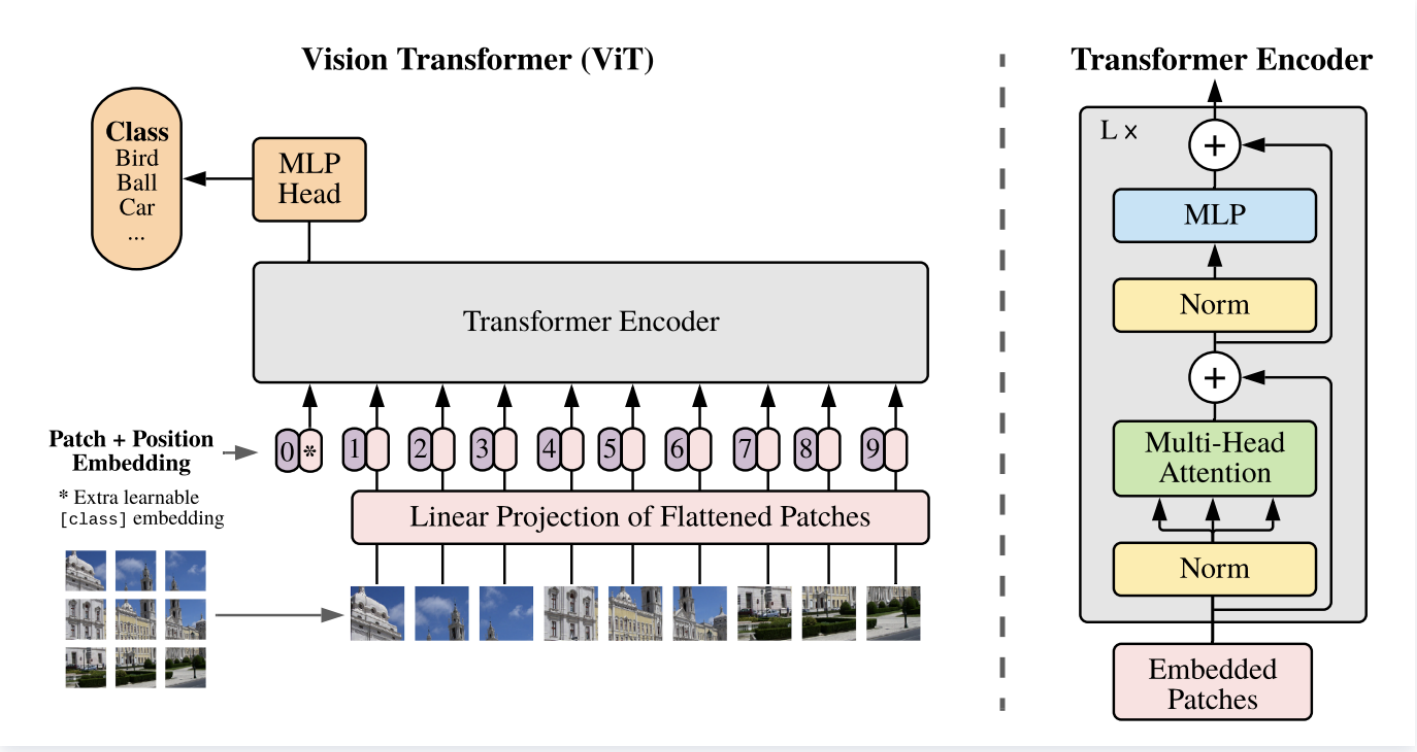
> **说明**
> 本文来自 GPU 云服务器用户实践征文，仅供学习和参考。

操作场景

本文介绍如何使用 GPU 云服务器进行 ViT 模型离线训练，完成简单的图像分类任务。

ViT 模型简介

ViT 全称 Vision Transformer，该模型由 Alexey Dosovitskiy 等人提出，在多个任务上取得 SoTA 结果。示意图如下：



对于一幅输入的图像，ViT 将其划分为多个子图像 patch，每个 patch 拼接 position embedding 后，和类别标签一起作为 Transfomer Encoder 的一组输入。而类别标签位置对应的输出层结果通过一个网络后，即得到 ViT 的输出。在预训练状态下，该结果对应的 ground truth 可以使用掩码的某个 patch 作为替代。

示例环境

- **实例类型**：本文可选实例为 GN7 与 GN8，结合 Technical 提供的 GPU 对比，Turing 架构的 T4 性能优于 Pascal 架构的 P40。本文最终选用 GN7.5XLARGE80。

- **所在地域**：由于可能需上传一些尺寸较大的数据集，需优先选择延迟最低的地域。本文使用 在线 Ping 工具测试，所在位置到提供 GN7 的重庆区域延迟最小，因此选择重庆区域。
- **系统盘**：100GB 高性能云硬盘。
- **操作系统**：Ubuntu 18.04
- **带宽**：5Mbps
- **本地操作系统**：MacOS

操作步骤

设置实例免密登录（可选）

1. （可选）您可在本机 `~/.ssh/config` 中，配置服务器的别名。本文创建别名为 `tcg` 。
2. 通过 `ssh-copy-id` 命令，将本机 SSH 公钥复制至 GPU 云服务器。
3. 在 GPU 云服务器中执行以下命令，关闭密码登录以增强安全性。

```
echo 'PasswordAuthentication no' | sudo tee -a /etc/ssh/ssh\_config
```

4. 执行以下命令，重启 SSH 服务。

```
sudo systemctl restart sshd
```

PyTorch-GPU 开发环境配置

若使用 GPU 版本的 PyTorch 进行开发，则需要进行一些环境配置。步骤如下：

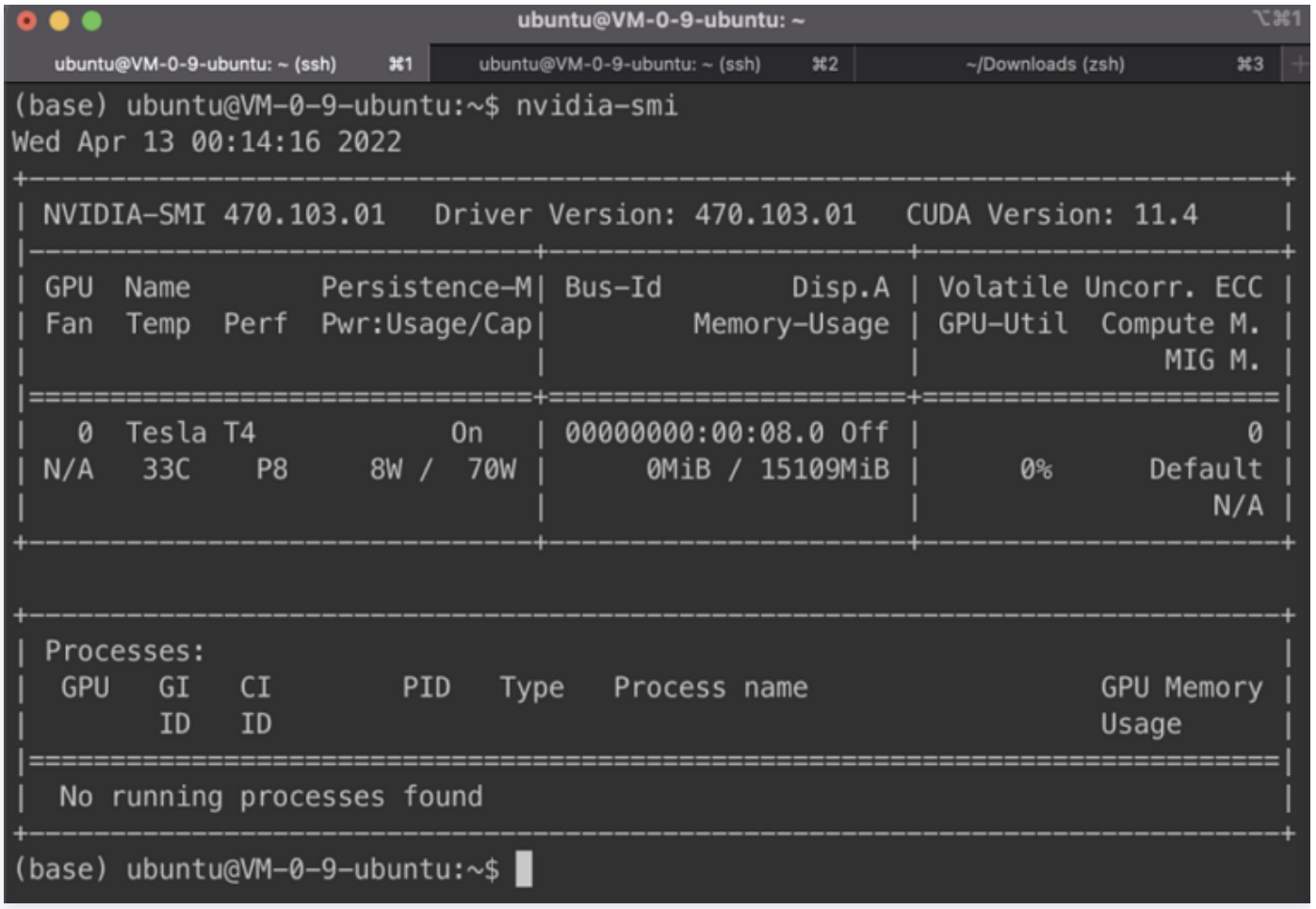
1. 安装 Nvidia 显卡驱动
   执行以下命令，安装 Nvidia 显卡驱动。

```
sudo apt install nvidia-driver-418
```

   安装完成后执行如下命令，查看是否安装成功。

```
nvidia-smi
```

返回结果如下图所示，表示已安装成功。



2. 配置 conda 环境
依次执行以下命令，配置 conda 环境。

```
wget https://repo.anaconda.com/miniconda/Miniconda3-py39_4.11.0-Linux-x86_64.sh
```

```
chmod +x Miniconda3-py39_4.11.0-Linux-x86_64.sh
```

```
./Miniconda3-py39_4.11.0-Linux-x86_64.sh
```

```
rm Miniconda3-py39_4.11.0-Linux-x86_64.sh
```

3. 编辑 `~/.condarc` 文件，加入以下软件源信息，将 conda 的软件源替换为清华源。

```
channels:
  - defaults
show\_channel\_urls: true
default\_channels:
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/r
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/msys2
custom\_channels:
  conda-forge: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  msys2: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  bioconda: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  menpo: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  pytorch: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  pytorch-lts: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  simpleitk: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
```

4. **执行以下命令，设置 pip 源为腾讯云镜像源。**

```
pip config set global.index-url
https://mirrors.cloud.tencent.com/pypi/simple
```

5. **安装 PyTorch**

   执行以下命令，安装 PyTorch。

```
conda install pytorch torchvision cudatoolkit=11.4 -c pytorch --yes
```

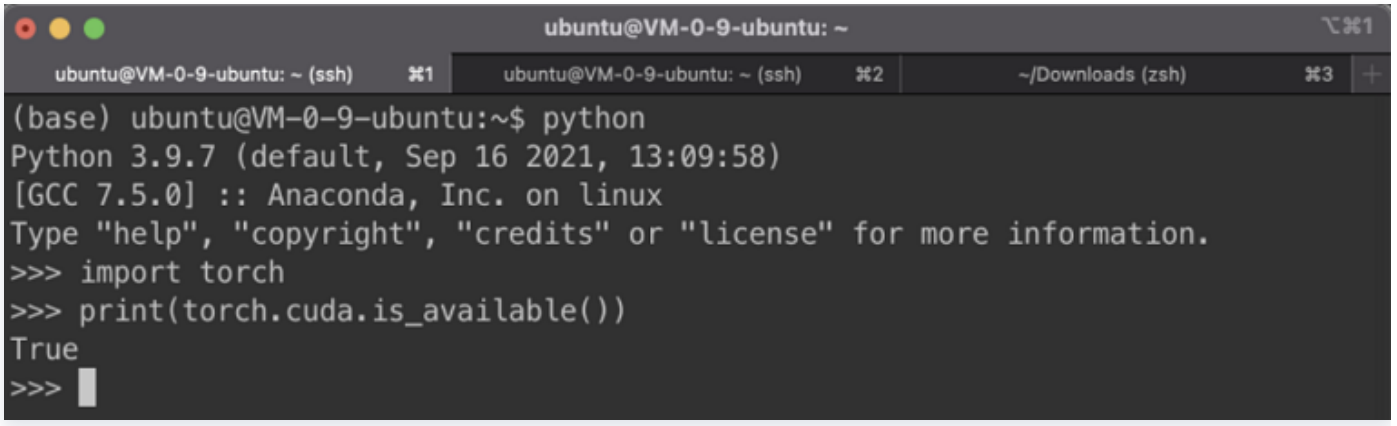
依次执行以下命令，查看 PyTorch 是否安装成功。

```
python
```

```
import torch
```

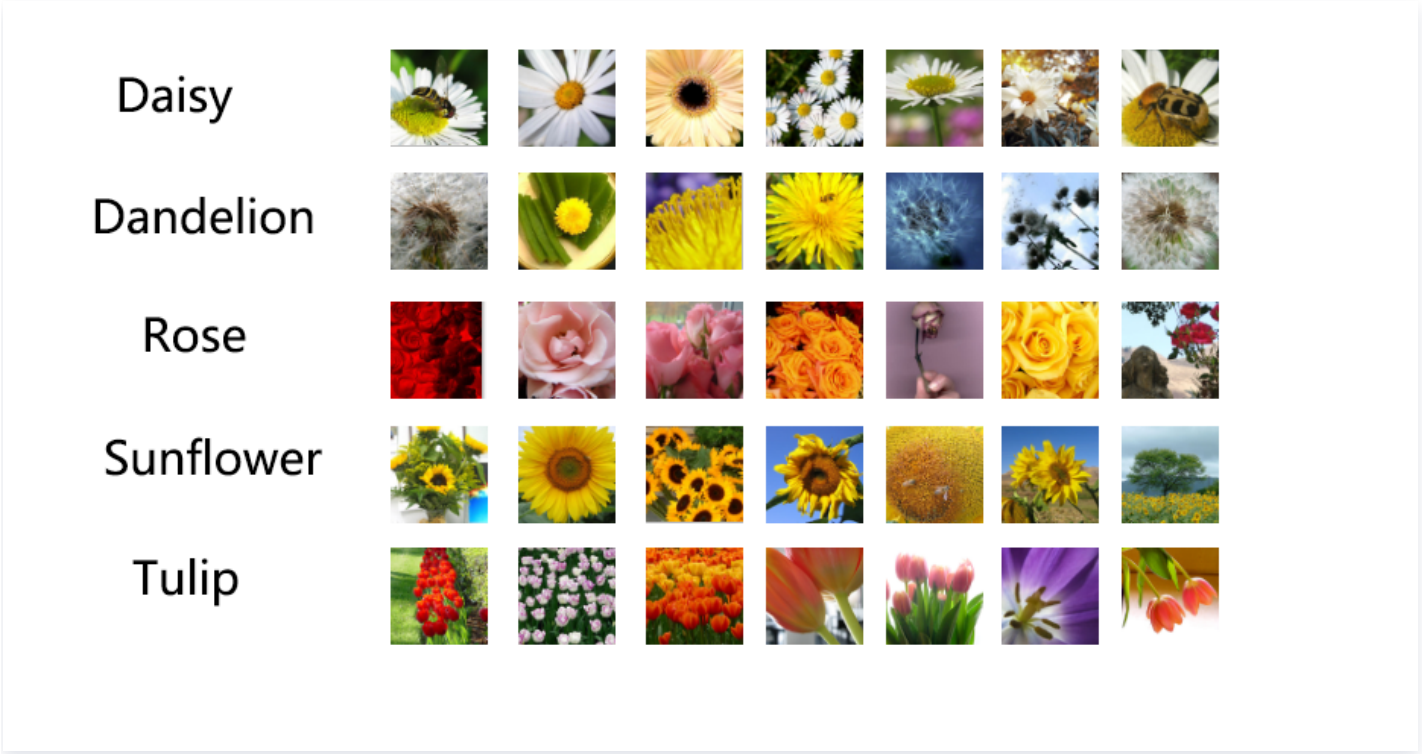
```
print(torch.cuda.is_avaliable())
```

返回结果如下图所示，表示 PyTorch 已安装成功。



准备实验数据

本次训练的测试任务是图像分类任务，使用了花朵图像分类数据集。该数据集包含5类花朵，数据大小为218M。数据集抽样展示如下：（各类别下花朵照片示例）



原始数据集中的各个分类数据分别存放在类名对应的文件夹下。首先需将其转化为 imagenet 对应的标准格式。按 4：1划分训练和验证集，使用以下代码进行格式转换：

```
# split data into train set and validation set, train:val=scale

import shutil

import os

import math

scale = 4

data\_path = '../raw'

data\_dst = '../train\_val'

#create imagenet directory structure

os.mkdir(data\_dst)

os.mkdir(os.path.join(data\_dst, 'train'))

os.mkdir(os.path.join(data\_dst, 'validation'))
```

```
for item in os.listdir(data\_path):
    item\_path = os.path.join(data\_path, item)
 if os.path.isdir(item\_path):
        train\_dst = os.path.join(data\_dst, 'train', item)
        val\_dst = os.path.join(data\_dst, 'validation', item)
        os.mkdir(train\_dst)
        os.mkdir(val\_dst)
        files = os.listdir(item\_path)
 print(f'Class {item}:\n\t Total sample count is {len(files)}')
        split\_idx = math.floor(len(files) \* scale / ( 1 + scale ))
 print(f'\t Train sample count is {split\_idx}')
 print(f'\t Val sample count is {len(files) - split\_idx}\n')
 for idx, file in enumerate(files):
            file\_path = os.path.join(item\_path, file)
 if idx <= split\_idx:
                shutil.copy(file\_path, train\_dst)
 else:
                shutil.copy(file\_path, val\_dst)
print(f'Split Complete. File path: {data\_dst}')
```

数据集概览如下：

```
Class roses:
     Total sample count is 641
```

```
      Train sample count is 512

      Validation sample count is 129

Class sunflowers:

      Total sample count is 699

      Train sample count is 559

      Validation sample count is 140

Class tulips:

      Total sample count is 799

      Train sample count is 639

      Validation sample count is 160

Class daisy:

      Total sample count is 633

      Train sample count is 506

      Validation sample count is 127

Class dandelion:

      Total sample count is 898

      Train sample count is 718

      Validation sample count is 180
```

为了加速训练过程，我们进一步将数据集转换为 Nvidia-DALI 这种 GPU 友好的格式。DALI 全称 Data Loading Library，该库可以通过使用 GPU 替代 CPU 来加速数据预处理过程。在已有 imagenet 格式数据的前提下，使用 DALI 只需运行以下命令即可：

```
git clone https://github.com/ver217/imagenet-tools.git
```

```
cd imagenet-tools && python3 make\_tfrecords.py \

  --raw\_data\_dir="../train\_val" \

  --local\_scratch\_dir="../train\_val\_tfrecord" && \

python3 make\_idx.py --tfrecord\_root="../train\_val\_tfrecord"
```

模型训练结果

为了便于后续训练分布式大规模模型，本文在分布式训练框架 Colossal-AI 的基础上进行模型训练和开发。Colossal-AI 提供了一组便捷的接口，通过这组接口能方便地实现数据并行、模型并行、流水线并行或者混合并行。
参考 Colossal-AI 提供的 demo，本文使用 pytorch-image-models 库所集成的 ViT 实现，选择最小的 `vit\_tiny\_patch16\_224` 模型，该模型的分辨率为224*224, 每个样本被划分为16个 `patch` 。

1. 参见 Colossal- AI官网 安装 Colossal-AI ，通过以下命令pytorch-image-models：

```
pip install timm
```

2. 参见 Colossal-AI 提供的 demo ，编写模型训练代码如下：

```
from pathlib import Path

from colossalai.logging import get\_dist\_logger

import colossalai

import torch

import os

from colossalai.core import global\_context as gpc

from colossalai.utils import get\_dataloader, MultiTimer

from colossalai.trainer import Trainer, hooks

from colossalai.nn.metric import Accuracy

from torchvision import transforms

from colossalai.nn.lr\_scheduler import CosineAnnealingLR
```

```
from tqdm import tqdm

from titans.utils import barrier\_context

from colossalai.nn.lr\_scheduler import LinearWarmupLR

from timm.models import vit\_tiny\_patch16\_224

from titans.dataloader.imagenet import build\_dali\_imagenet

from mixup import MixupAccuracy, MixupLoss

def main():

 parser = colossalai.get\_default\_parser()

 args = parser.parse\_args()

 colossalai.launch\_from\_torch(config='./config.py')

 logger = get\_dist\_logger()

 # build model

 model = vit\_tiny\_patch16\_224(num\_classes=5, drop\_rate=0.1)

 # build dataloader

 root = os.environ.get('DATA', '../train\_val\_tfrecord')

 train\_dataloader, test\_dataloader = build\_dali\_imagenet(

     root, rand\_augment=True)

 # build criterion

 criterion = MixupLoss(loss\_fn\_cls=torch.nn.CrossEntropyLoss)

 # optimizer

 optimizer = torch.optim.SGD(
```

```
    model.parameters(), lr=0.1, momentum=0.9, weight\_decay=5e-4)

# lr\_scheduler

lr\_scheduler = CosineAnnealingLR(

   optimizer, total\_steps=gpc.config.NUM\_EPOCHS)

engine, train\_dataloader, test\_dataloader, \_ =
colossalai.initialize(

    model,

    optimizer,

    criterion,

    train\_dataloader,

    test\_dataloader,

)

# build a timer to measure time

timer = MultiTimer()

# create a trainer object

trainer = Trainer(engine=engine, timer=timer, logger=logger)

# define the hooks to attach to the trainer

hook\_list = [

    hooks.LossHook(),

    hooks.LRSchedulerHook(lr\_scheduler=lr\_scheduler,
by\_epoch=True),

    hooks.AccuracyHook(accuracy\_func=MixupAccuracy()),

    hooks.LogMetricByEpochHook(logger),
```

```
    hooks.LogMemoryByEpochHook(logger),
    hooks.LogTimingByEpochHook(timer, logger),
    hooks.TensorboardHook(log\_dir='./tb\_logs', ranks=[0]),
    hooks.SaveCheckpointHook(checkpoint\_dir='./ckpt')
]
# start training
trainer.fit(train\_dataloader=train\_dataloader,
            epochs=gpc.config.NUM\_EPOCHS,
            test\_dataloader=test\_dataloader,
            test\_interval=1,
            hooks=hook\_list,
            display\_progress=True)
if \_\_name\_\_ == '\_\_main\_\_':
main()
```

模型的具体配置如下所示：

```
from colossalai.amp import AMP\_TYPE
BATCH\_SIZE = 128
DROP\_RATE = 0.1
NUM\_EPOCHS = 200
CONFIG = dict(fp16=dict(mode=AMP\_TYPE.TORCH))
gradient\_accumulation = 16
```

版权所有：腾讯云计算（北京）有限责任公司

```
clip\_grad\_norm = 1.0

dali = dict(

 gpu\_aug=True,

 mixup\_alpha=0.2

)
```

模型运行过程如下图所示， 单个 epoch 的时间在20s以内：

```
[Epoch 3 / Test]: 100%|          | 23/23 [00:01<00:00, 13.11it/s]
[05/30/22 12:03:12] INFO     colossalai - colossalai - INFO: /root/miniconda3/li
                             b/python3.8/site-packages/colossalai/trainer/hooks/
                             _log_hook.py:99 after_test_epoch
                    INFO     colossalai - colossalai - INFO: [Epoch 3 / Test]:
                             Loss = 1.3298 | Accuracy = 0.41724
                    INFO     colossalai - colossalai - INFO: /root/miniconda3/li
                             b/python3.8/site-packages/colossalai/trainer/hooks/
                             _log_hook.py:251 after_test_epoch
                    INFO     colossalai - colossalai - INFO: [Epoch 3 / Test]:
                             Test-epoch: last = 1.754 s, mean = 1.754 s |
                             Test-step: last = 0.065791 s, mean = 0.07539 s
                    INFO     colossalai - colossalai - INFO: /root/miniconda3/li
                             b/python3.8/site-packages/colossalai/utils/memory.p
                             y:91 report_memory_usage
                    INFO     colossalai - colossalai - INFO: [Epoch 3 / Test]:
                             GPU: allocated 260.12 MB, max allocated 2471.5 MB,
                             cached: 5974.0 MB, max cached: 5974.0 MB
[Epoch 4 / Train]: 100%|          | 80/80 [00:17<00:00,  4.47it/s]
```

结果显示模型在验证集上达到的最佳准确率为66.62%。

总结

本次使用过程中遇到的最大的问题是从 GitHub 克隆非常缓慢，为了解决该问题，尝试使用了 tunnel 和 proxychains 工具进行提速。但该行为违反了云服务器使用规则，导致了一段时间的云服务器不可用，最终通过删除代理并提交工单的方式才得以解决。
借此也提醒其他用户，进行外网代理不符合云服务器使用规范，为了保证您服务的稳定运行，切勿违反规定。

参考

[1] Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." arXiv preprint arXiv:2010.11929 (2020).
[2] NVIDIA/DALI

[3] Bian, Zhengda, et al. "Colossal-AI: A Unified Deep Learning System For Large-Scale Parallel Training." arXiv preprint arXiv:2110.14883 (2021).

版权所有：腾讯云计算（北京）有限责任公司