TencentOS Server

工具指南

腾讯云

文档目录

工具指南
AI 使用
TencentOS Server 3 安装 pytorch 及运行 AI 相关模型

最近更新时间：2024-03-25 14:24:41

环境准备

GPU 机型要求

购买 GPU 机型时，根据需求选择下图所示的 GPU 驱动版本、CUDA 版本、cuDNN 版本。

Python 3 版本要求

必须为 3.8 及以上版本，检查方法如下：

```
[root@VM-0-21-tencentos ~]# python3 -V
Python 3.8.16
[root@VM-0-21-tencentos ~]# pip3 -V
pip 19.3.1 from /usr/lib/python3.8/site-packages/pip (python 3.8)
[root@VM-0-21-tencentos ~]#
```

如果 Python 版本不满足要求，请按照如下步骤操作：

1. 安装 Python 3.8。

```
yum install -y python3.8
```

2. 配置 Python 3.8 为默认的 Python 3 版本。

```
cd /usr/bin/ && rm /usr/bin/python3 && ln -s python3.8 python3
```

3. 配置 pip 3.8 为默认的 pip 3 版本。

```
cd /usr/bin/ && rm /usr/bin/pip3 && ln -s pip3.8 pip3
```

安装 pytorch

您可以执行如下命令安装 pytorch。

```
pip3 install  torch==1.12
```

安装 pytorch 后，您也可以执行如下命令检查 pytorch 是否安装成功。

```
[root@VM-0-21-tencentos ~]# pip3 list | grep torch
torch                         1.12.0
[root@VM-0-21-tencentos ~]#
```

典型模型示例

Stable Diffusion

网站地址：https://huggingface.co/runwayml/stable-diffusion-v1-5

> **说明：**
> 该网站为国外网站，下载速度可能较慢，取决于您的网络性能。

1. 安装依赖的软件包。

```
pip3 install diffusers transformers
```

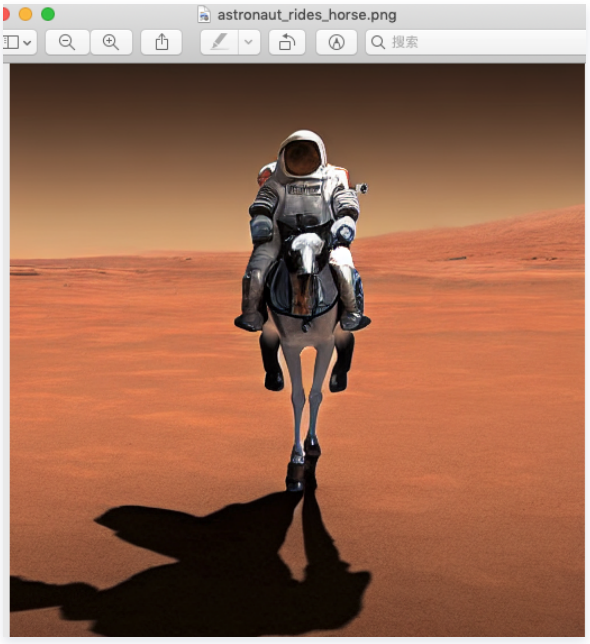
2. 将如下 Python 代码保存为 Python 脚本。假设脚本名称为：stable_diffusion.py。

```
from diffusers import StableDiffusionPipeline
import torch

model_id = "runwayml/stable-diffusion-v1-5"
pipe = StableDiffusionPipeline.from_pretrained(model_id, torch_dtype=torch.float16)
pipe = pipe.to("cuda")

prompt = "a photo of an astronaut riding a horse on mars"
image = pipe(prompt).images[0]

image.save("astronaut_rides_horse.png")
```

3. 运行上一步保存的 Python 脚本。

```
[root@VM-2-9-tencentos stable_diffusion]# python3 stable_diffusion.py
The cache for model files in Transformers v4.22.0 has been updated. Migrating your old cache. This is a one-time only operation. You can i
0it [00:00, ?it/s]
Downloading (…)ain/model_index.json: 100%|
Downloading (…)_checker/config.json: 100%|
Downloading (…)cheduler_config.json: 100%|
Downloading (…)rocessor_config.json: 100%|
Downloading (…)_encoder/config.json: 100%|
Downloading (…)cial_tokens_map.json: 100%|
Downloading (…)okenizer_config.json: 100%|
Downloading (…)7f0/unet/config.json: 100%|
Downloading (…)57f0/vae/config.json: 100%|
Downloading (…)tokenizer/merges.txt: 100%|
Downloading (…)tokenizer/vocab.json: 100%|
Downloading (…)ch_model.safetensors: 100%|
Downloading model.safetensors:  17%|
Downloading model.safetensors:  47%|
```

上图展示的是等待下载模型所需要的资源，最后会在当前目录生成一张图片：astronaut_rides_horse.png。
训练结果如下：



百川 13B 对话模型

网站地址：https://modelscope.cn/models/baichuan-inc/Baichuan-13B-Chat/summary

> **说明：**
> 该网站为国内模型网站，下载速度较快。

1. 安装依赖的软件包

```
pip3 install modelscope
pip3 install pip --upgrade       #运行过程中会失败，需要升级
pip3 install sentencepiece
```

2. 运行如下 Python 脚本。

```
import torch
from modelscope import snapshot_download, Model
model_dir = snapshot_download("baichuan-inc/Baichuan-13B-Chat", revision='v1.0.3')
model = Model.from_pretrained(model_dir, device_map="balanced", trust_remote_code=True,
torch_dtype=torch.float16)
messages = []
messages.append({"role": "user", "content": "世界上第二高的山峰是哪一座？"})
response = model(messages)
print(response)
```

模型运行结果如下：

```
[root@VM-0-21-tencentos Baichuan-13B-Chat]# python3 bai.py
2023-07-12 15:20:53,287 - modelscope - INFO - PyTorch version 1.12.0 Found.
2023-07-12 15:20:53,288 - modelscope - INFO - Loading ast index from /root/.cache/modelscope/ast_indexer
2023-07-12 15:20:53,316 - modelscope - INFO - Loading done! Current index file version is 1.7.1, with md5 42c0395d32aad94a0fb1c9345805da71 and a total nu
2023-07-12 15:20:53,530 - modelscope - INFO - Use user-specified model revision: v1.0.3
2023-07-12 15:20:53,721 - modelscope - INFO - initialize model from /root/.cache/modelscope/hub/baichuan-inc/Baichuan-13B-Chat
Looking in indexes: http://mirrors.tencentyun.com/pypi/simple
Looking in links: https://modelscope.oss-cn-beijing.aliyuncs.com/releases/repo.html, https://modelscope.oss-cn-beijing.aliyuncs.com/releases/repo.html
Requirement already satisfied: cpm_kernels in /usr/local/lib/python3.8/site-packages (1.0.11)
Requirement already satisfied: transformers_stream_generator in /usr/local/lib/python3.8/site-packages (0.0.4)
Requirement already satisfied: transformers>=4.26.1 in /usr/local/lib/python3.8/site-packages (from transformers_stream_generator) (4.30.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.8/site-packages (from transformers>=4.26.1->transformers_stream_generator) (3.12.2)
Requirement already satisfied: huggingface-hub<1.0,>=0.14.1 in /usr/local/lib/python3.8/site-packages (from transformers>=4.26.1->transformers_stream_gen
Requirement already satisfied: numpy>=1.17 in /usr/local/lib64/python3.8/site-packages (from transformers>=4.26.1->transformers_stream_generator) (1.22.0
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.8/site-packages (from transformers>=4.26.1->transformers_stream_generator) (23.1
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib64/python3.8/site-packages (from transformers>=4.26.1->transformers_stream_generator) (6.0)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib64/python3.8/site-packages (from transformers>=4.26.1->transformers_stream_generator) (
Requirement already satisfied: requests in /usr/local/lib/python3.8/site-packages (from transformers>=4.26.1->transformers_stream_generator) (2.31.0)
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /usr/local/lib64/python3.8/site-packages (from transformers>=4.26.1->transformers_str
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib64/python3.8/site-packages (from transformers>=4.26.1->transformers_stream_generator)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.8/site-packages (from transformers>=4.26.1->transformers_stream_generator) (4.65.0)
Requirement already satisfied: fsspec in /usr/local/lib/python3.8/site-packages (from huggingface-hub<1.0,>=0.14.1->transformers>=4.26.1->transformers_st
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.8/site-packages (from huggingface-hub<1.0,>=0.14.1->transformers>=4.2
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib64/python3.8/site-packages (from requests->transformers>=4.26.1->transformers_st
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.8/site-packages (from requests->transformers>=4.26.1->transformers_stream_generator
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.8/site-packages (from requests->transformers>=4.26.1->transformers_stream_gen
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/site-packages (from requests->transformers>=4.26.1->transformers_stream_gen
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to
2023-07-12 15:20:56,136 - modelscope - WARNING - ('MODELS', 'text-generation', 'Baichuan-13B-Chat') not found in ast index file
WARNING: ('MODELS', 'text-generation', 'Baichuan-13B-Chat') not found in ast index file
WARNING: The model weights are not tied. Please use the `tie_weights` method before using the `infer_auto_device` function.
Loading checkpoint shards: 100%|
{'response': '乔戈里峰。世界第二高峰——乔戈里峰西方登山者称其为k2峰，海拔高度是8611米，位于喀喇昆仑山脉的中巴边境上。', 'history': ''}
[root@VM-0-21-tencentos Baichuan-13B-Chat]# ls
```

您也可以替换脚本中的提问，例如：北京申奥成功是哪一年？返回结果如下：

```
[root@VM-0-21-tencentos Baichuan-13B-Chat]# python3 bai.py
2023-07-12 15:53:06,072 - modelscope - INFO - PyTorch version 1.12.0 Found.
2023-07-12 15:53:06,072 - modelscope - INFO - Loading ast index from /root/.cache/modelscope/ast_indexer
2023-07-12 15:53:06,102 - modelscope - INFO - Loading done! Current index file version is 1.7.1, with md5 42c0395d32aad94a0fb1c9345805da71 and a total number of 861 components indexed
2023-07-12 15:53:06,365 - modelscope - INFO - Use user-specified model revision: v1.0.3
2023-07-12 15:53:06,681 - modelscope - INFO - initialize model from /root/.cache/modelscope/hub/baichuan-inc/Baichuan-13B-Chat
Looking in indexes: http://mirrors.tencentyun.com/pypi/simple
Looking in links: https://modelscope.oss-cn-beijing.aliyuncs.com/releases/repo.html, https://modelscope.oss-cn-beijing.aliyuncs.com/releases/repo.html
Requirement already satisfied: cpm_kernels in /usr/local/lib/python3.8/site-packages (1.0.11)
Requirement already satisfied: transformers_stream_generator in /usr/local/lib/python3.8/site-packages (0.0.4)
Requirement already satisfied: transformers>=4.26.1 in /usr/local/lib/python3.8/site-packages (from transformers_stream_generator) (4.30.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.8/site-packages (from transformers>=4.26.1->transformers_stream_generator) (3.12.2)
Requirement already satisfied: huggingface-hub<1.0,>=0.14.1 in /usr/local/lib/python3.8/site-packages (from transformers>=4.26.1->transformers_stream_generator) (0.16.4)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib64/python3.8/site-packages (from transformers>=4.26.1->transformers_stream_generator) (1.22.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.8/site-packages (from transformers>=4.26.1->transformers_stream_generator) (23.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib64/python3.8/site-packages (from transformers>=4.26.1->transformers_stream_generator) (6.0)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib64/python3.8/site-packages (from transformers>=4.26.1->transformers_stream_generator) (2023.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.8/site-packages (from transformers>=4.26.1->transformers_stream_generator) (2.31.0)
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /usr/local/lib64/python3.8/site-packages (from transformers>=4.26.1->transformers_stream_generator) (0.13.3)
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib64/python3.8/site-packages (from transformers>=4.26.1->transformers_stream_generator) (0.3.1)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.8/site-packages (from transformers>=4.26.1->transformers_stream_generator) (4.65.0)
Requirement already satisfied: fsspec in /usr/local/lib/python3.8/site-packages (from huggingface-hub<1.0,>=0.14.1->transformers>=4.26.1->transformers_stream_generator) (2023.6.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.8/site-packages (from huggingface-hub<1.0,>=0.14.1->transformers>=4.26.1->transformers_stream_generator) (4.7.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib64/python3.8/site-packages (from requests->transformers>=4.26.1->transformers_stream_generator) (3.2.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.8/site-packages (from requests->transformers>=4.26.1->transformers_stream_generator) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.8/site-packages (from requests->transformers>=4.26.1->transformers_stream_generator) (2.0.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/site-packages (from requests->transformers>=4.26.1->transformers_stream_generator) (2023.5.7)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
2023-07-12 15:53:09,117 - modelscope - WARNING - ('MODELS', 'text-generation', 'Baichuan-13B-Chat') not found in ast index file
WARNING: ('MODELS', 'text-generation', 'Baichuan-13B-Chat') not found in ast index file
WARNING: The model weights are not tied. Please use the `tie_weights` method before using the `infer_auto_device` function.
Loading checkpoint shards: 100%|
{'response': '2001年，北京时间2001年7月13日晚上，在莫斯科的卢日尼基体育馆，当国际奥委会主席萨马兰奇宣布北京获得2008年奥运会主办权时，整个中国都沸腾了！\n从1992年的错失良机到2001年的众望所归，中国人等了整整9个春秋。', 'history': ''}
[root@VM-0-21-tencentos Baichuan-13B-Chat]#
```

openjourney

网站地址：prompthero/openjourney · Hugging Face

该例子使用源码的方式训练

1. 安装软件：

```
yum install git-lfs -y
```

2. 下载 openjourney 对应的代码。

```
# Make sure you have git-lfs installed (https://git-lfs.com)
git lfs install
git clone https://huggingface.co/prompthero/openjourney

# if you want to clone without large files - just their pointers
# prepend your git clone with the following env var:
GIT_LFS_SKIP_SMUDGE=1
```

下载过程可能较慢，其中有三个大文件，需要耐心等待。

```
[root@VM-2-9-tencentos openjourney]# ls -lh
total 4.5G
drwxr-xr-x 2 root root 4.0K Jul 12 15:57 feature_extractor
-rw-r--r-- 1 root root 2.0G Jul 12 16:16 mdjrny-v4.ckpt
-rw-r--r-- 1 root root 2.0G Jul 12 16:16 mdjrny-v4.safetensors
-rw-r--r-- 1 root root  541 Jul 12 15:57 model_index.json
-rw-r--r-- 1 root root 470M Jul 12 16:03 model.safetensors
-rw-r--r-- 1 root root 2.7K Jul 12 15:57 README.md
drwxr-xr-x 2 root root 4.0K Jul 12 16:11 safety_checker
drwxr-xr-x 2 root root 4.0K Jul 12 15:57 scheduler
drwxr-xr-x 2 root root 4.0K Jul 12 16:08 text_encoder
drwxr-xr-x 2 root root 4.0K Jul 12 15:57 tokenizer
drwxr-xr-x 2 root root 4.0K Jul 12 16:19 unet
drwxr-xr-x 2 root root 4.0K Jul 12 16:10 vae
[root@VM-2-9-tencentos openjourney]#
```

3. 运行如下脚本：

```
from diffusers import StableDiffusionPipeline
import torch
model_id = "/root/openjourney_code/openjourney"     #修改为你本地的目录即可
pipe = StableDiffusionPipeline.from_pretrained(model_id, torch_dtype=torch.float16)
pipe = pipe.to("cuda")
prompt = "retro serie of different cars with different colors and shapes, mdjrny-v4 style"
image = pipe(prompt).images[0]
image.save("./retro_cars.png")
```

运行结果如下：

```
[root@VM-2-9-tencentos openjourney_code]# python3 open.py
Cannot initialize model with low cpu memory usage because `accelerate` was not found in the environment. Defaulting to `low_cpu
_mem_usage=False`. It is strongly recommended to install `accelerate` for faster and less memory-intense model loading. You can
 do so with:
```
pip install accelerate
```
.
`text_config_dict` is provided which will be used to initialize `CLIPTextConfig`. The value `text_config["id2label"]` will be o
verriden.
100%|| 50/50 [00:03<00:00, 14.03it/s]
[root@VM-2-9-tencentos openjourney_code]#
[root@VM-2-9-tencentos openjourney_code]# ls
openjourney  open.py  retro_cars.png
[root@VM-2-9-tencentos openjourney_code]#
```

运行完毕会生成一张图片，如下图所示：

⚠ 注意：

如果出现类似如下报错，说明 GPU 显存不够，需要更高配置的 GPU 机型。

```
RuntimeError: CUDA out of memory. Tried to allocate 1.25 GiB (GPU 0; 14.76 GiB total capacity;
12.96 GiB already allocated; 993.75 MiB free; 12.96 GiB reserved in total by
PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid
fragmentation.  See documentation for Memory Management
```

TencentOS Server 3 安装主要深度学习框架及示例

最近更新时间：2024-03-25 14:24:41

环境准备

GPU 机型要求

购买 GPU 机型时，根据需求选择下图所示的 GPU 驱动版本、CUDA 版本、cuDNN 版本。

Python 3 版本要求

必须为 3.8 及以上版本，检查方法如下：

```
[root@VM-0-21-tencentos ~]# python3 -V
Python 3.8.16
[root@VM-0-21-tencentos ~]# pip3 -V
pip 19.3.1 from /usr/lib/python3.8/site-packages/pip (python 3.8)
[root@VM-0-21-tencentos ~]#
```

如果 Python 版本不满足要求，请按照如下步骤操作：

1. 安装 Python 3.8。

```
yum install -y python38 python38-devel
```

2. 配置 Python 3.8 为默认的 Python 3 版本。

```
cd /usr/bin/ && rm /usr/bin/python3 && ln -s python3.8 python3
```

3. 配置 pip 3.8 为默认的 pip 3 版本。

```
cd /usr/bin/ && rm /usr/bin/pip3 && ln -s pip3.8 pip3
```

升级 pip 及 setuptools

```
pip3 install --upgrade pip
pip3 install --upgrade setuptools
```

配置 python3-config

```
ln -s /usr/bin/python3.8-config /usr/bin/python3-config
```

典型模型示例

Tensorflow 训练示例

1. 安装 TensorFlow。

   1.1 您可以执行以下命令安装 TensorFlow。

```
pip3 install tensorflow==2.6
```

   1.2 安装 TensorFlow 后，您也可以执行以下命令检查 TensorFlow 是否安装成功。

```
[root@VM-16-12-tencentos ~]# pip3 list |  grep tensorflow
tensorflow              2.6.0
tensorflow-estimator    2.13.0
```

2. 安装依赖。

```
pip3 install keras==2.6
pip3 uninstall protobuf      #卸载默认的 4.23.4高版本
pip3 install protobuf==3.20.0
```

3. 将如下 Python 代码保存为 Python 脚本，例如：demo.py。

```
import tensorflow as tf


mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])

predictions = model(x_train[:1]).numpy()
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
loss_fn(y_train[:1], predictions).numpy()

model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test,  y_test, verbose=2)
```

4. 运行保存好的 Python 脚本。

```
[root@VM-0-10-tencentos tf]# python3 demo.py
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 53s 5us/step
11501568/11490434 [==============================] - 53s 5us/step
2023-07-19 20:01:40.952882: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neu
ral Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-07-19 20:01:43.950482: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device /job:localhost/replica:0/task:0/devic
e:GPU:0 with 20812 MB memory:  -> device: 0, name: NVIDIA A10, pci bus id: 0000:0b:01.0, compute capability: 8.6
2023-07-19 20:01:43.952451: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device /job:localhost/replica:0/task:0/devic
e:GPU:1 with 20812 MB memory:  -> device: 1, name: NVIDIA A10, pci bus id: 0000:0b:02.0, compute capability: 8.6
2023-07-19 20:01:43.954224: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device /job:localhost/replica:0/task:0/devic
e:GPU:2 with 20812 MB memory:  -> device: 2, name: NVIDIA A10, pci bus id: 0000:0b:03.0, compute capability: 8.6
2023-07-19 20:01:43.955987: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device /job:localhost/replica:0/task:0/devic
e:GPU:3 with 20812 MB memory:  -> device: 3, name: NVIDIA A10, pci bus id: 0000:0b:04.0, compute capability: 8.6
2023-07-19 20:01:43.957774: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device /job:localhost/replica:0/task:0/devic
e:GPU:4 with 22338 MB memory:  -> device: 4, name: NVIDIA A10, pci bus id: 0000:41:01.0, compute capability: 8.6
2023-07-19 20:01:43.959711: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device /job:localhost/replica:0/task:0/devic
e:GPU:5 with 22338 MB memory:  -> device: 5, name: NVIDIA A10, pci bus id: 0000:41:02.0, compute capability: 8.6
2023-07-19 20:01:43.961455: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device /job:localhost/replica:0/task:0/devic
e:GPU:6 with 20812 MB memory:  -> device: 6, name: NVIDIA A10, pci bus id: 0000:41:03.0, compute capability: 8.6
2023-07-19 20:01:43.963299: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device /job:localhost/replica:0/task:0/devic
e:GPU:7 with 20812 MB memory:  -> device: 7, name: NVIDIA A10, pci bus id: 0000:41:04.0, compute capability: 8.6
2023-07-19 20:01:44.638762: I tensorflow/stream_executor/cuda/cuda_blas.cc:1760] TensorFloat-32 will be used for the matrix multiplicatio
n. This will only be logged once.
2023-07-19 20:01:44.772794: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enab
led (registered 2)
Epoch 1/5
1875/1875 [==============================] - 2s 827us/step - loss: 0.2927 - accuracy: 0.9144
Epoch 2/5
1875/1875 [==============================] - 2s 820us/step - loss: 0.1415 - accuracy: 0.9577
Epoch 3/5
1875/1875 [==============================] - 2s 816us/step - loss: 0.1054 - accuracy: 0.9681
Epoch 4/5
1875/1875 [==============================] - 2s 816us/step - loss: 0.0865 - accuracy: 0.9733
Epoch 5/5
1875/1875 [==============================] - 2s 815us/step - loss: 0.0742 - accuracy: 0.9765
313/313 - 0s - loss: 0.0709 - accuracy: 0.9783
```

Pytorch 训练示例

1. 安装 Pytorch。

   1.1 您可以执行以下命令安装 Pytorch。

```
pip3 install torch==1.12.1+cu113 torchvision==0.13.1+cu113 torchaudio==0.12.1 --extra-index-url
https://download.pytorch.org/whl/cu113
```

   1.2 安装 Pytorch 后，您也可以执行以下命令检查 Pytorch 是否安装成功。

```
[root@VM-16-12-tencentos ~]# pip3 list | grep torch
torch              1.12.1+cu113
torchaudio         0.12.1+cu113
torchvision        0.13.1+cu113
```

2. 将如下 Python 代码保存为 Python 脚本，例如：demo.py。

```
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor


# Download training data from open datasets.
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor(),
)

# Download test data from open datasets.
```

```
test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor(),
)

batch_size = 64

# Create data loaders.
train_dataloader = DataLoader(training_data, batch_size=batch_size)
test_dataloader = DataLoader(test_data, batch_size=batch_size)

# Get cpu, gpu or mps device for training.
device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)
print(f"Using {device} device")

# Define model
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10)
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = NeuralNetwork().to(device)

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)


def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
```

```
        if batch % 100 == 0:
            loss, current = loss.item(), (batch + 1) * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")


def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")


epochs = 5
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train(train_dataloader, model, loss_fn, optimizer)
    test(test_dataloader, model, loss_fn)
print("Done!")
```

3. 运行保存好的 Python 脚本。

```
[root@VM-0-10-tencentos py]# python3 demo.py
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz to data/FashionMNIST/raw/train-images-i
dx3-ubyte.gz
100.0%
Extracting data/FashionMNIST/raw/train-images-idx3-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz to data/FashionMNIST/raw/train-labels-i
dx1-ubyte.gz
100.0%
Extracting data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to data/FashionMNIST/raw/t10k-images-idx
3-ubyte.gz
100.0%
Extracting data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to data/FashionMNIST/raw/t10k-labels-idx
1-ubyte.gz
100.0%
Extracting data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to data/FashionMNIST/raw

Using cuda device
Epoch 1
-------------------------------
loss: 2.305151  [   64/60000]
loss: 2.294250  [ 6464/60000]
loss: 2.282794  [12864/60000]
loss: 2.274063  [19264/60000]
loss: 2.257649  [25664/60000]
loss: 2.238950  [32064/60000]
loss: 2.237539  [38464/60000]
loss: 2.214913  [44864/60000]
loss: 2.212914  [51264/60000]
loss: 2.172773  [57664/60000]
Test Error:
 Accuracy: 42.0%, Avg loss: 2.171635
```

DeepSpeed 训练示例

1. 安装 DeepSpeed。

1.1 您可以执行以下命令安装 DeepSpeed。

```
pip3 install deepspeed
```

1.2 安装 DeepSpeed 后，您也可以执行以下命令检查 DeepSpeed 是否安装成功。

```
[root@VM-16-12-tencentos ~]# pip3 list | grep deepspeed
deepspeed          0.9.5
```

2. 安装依赖。

```
# NCCL
sudo yum-config-manager --add-repo
https://developer.download.nvidia.com/compute/cuda/repos/rhel8/x86_64/cuda-rhel8.repo
sudo yum install libnccl libnccl-devel libnccl-static -y

pip3 install datasets evaluate accelerate sentencepiece transformers==4.28.1 pydantic==1.10.7

pip3 uninstall numpy    #卸载numpy的高版本
pip3 install numpy==1.22.0
```

3. 下载示例代码，并做相应修改。

```
wget https://taco-1251783334.cos.ap-shanghai.myqcloud.com/demo/LLM/llama.tar.gz
tar xzf llama.tar.gz && cd llama
```

3.1 通过以下命令获取本机 eth0 ip，写到 hostfile 中。

```
ifconfig eth0
```

3.2 根据本机 GPU 数量修改 start.sh 文件中的 GPUS_PER_NODE 变量，默认为1。

```
GPUS_PER_NODE=1
```

4. 执行 start.sh 开始训练。

```
[root@VM-16-12-tencentos llama]# bash start.sh
torchrun --nproc_per_node 1 --nnodes 1 --node_rank 0 --master_addr 172.26.16.12 --master_port 6000 run_clm.py --deepspeed ds_zero3_no_of
r_name ./model/ --train_file ./data/pt_sample_data.txt --per_device_train_batch_size 1 --per_device_eval_batch_size 1 --do_train --seed
pe cosine --learning_rate 2e-4 --warmup_ratio 0.05 --weight_decay 0.01 --logging_strategy steps --logging_steps 10 --save_strategy steps
dient_accumulation_steps 1 --preprocessing_num_workers 8 --block_size 1024 --output_dir output --overwrite_output_dir --ddp_timeout 3000
oat16 --gradient_checkpointing --ddp_find_unused_parameters False --report_to none
[2023-07-19 17:38:05,166] [INFO] [real_accelerator.py:110:get_accelerator] Setting ds_accelerator to cuda (auto detect)
[2023-07-19 17:38:08,539] [WARNING] [comm.py:152:init_deepspeed_backend] NCCL backend in DeepSpeed not yet implemented
[2023-07-19 17:38:08,539] [INFO] [comm.py:594:init_distributed] cdb=None
[2023-07-19 17:38:08,539] [INFO] [comm.py:625:init_distributed] Initializing TorchBackend in DeepSpeed with backend nccl
07/19/2023 17:38:08 - WARNING - __main__ - Process rank: 0, device: cuda:0, n_gpu: 1distributed training: True, 16-bits training: True
07/19/2023 17:38:08 - INFO - __main__ - Training/evaluation parameters TrainingArguments(
_n_gpu=1,
adafactor=False,
adam_beta1=0.9,
adam_beta2=0.999,
adam_epsilon=1e-08,
auto_find_batch_size=False,
bf16=False,
bf16_full_eval=False,
data_seed=None,
dataloader_drop_last=False,
dataloader_num_workers=0,
dataloader_pin_memory=True,
ddp_bucket_cap_mb=None,
ddp_find_unused_parameters=False,
ddp_timeout=30000,
```

```
: 277.89 | backward_allreduce: 10.20 | step: 61.14
 95%|▉| 19/20 [00:11<00:00,  2.12it/s][2023-07-19 17:42:02,512] [INFO] [logging.py:96:log_dist] [Rank 0] rank=0 time (ms) | opti
mizer_step: 33.99
[2023-07-19 17:42:02,512] [INFO] [logging.py:96:log_dist] [Rank 0] step=20, skipped=2, lr=[5.418275829936537e-06], mom=[[0.9, 0.999]]
[2023-07-19 17:42:02,513] [INFO] [timer.py:215:stop] epoch=0/micro_step=20/global_step=20, RunningAvgSamplesPerSec=2.1502114540330632, Cu
rrSamplesPerSec=2.151743894343386, MemAllocated=8.85GB, MaxMemAllocated=11.47GB
[2023-07-19 17:42:02,513] [INFO] [logging.py:96:log_dist] [Rank 0] rank=0 time (ms) | forward_microstep: 113.69 | backward_microstep: 288
.21 | backward_inner_microstep: 277.95 | backward_allreduce_microstep: 10.18 | step_microstep: 61.74
[2023-07-19 17:42:02,513] [INFO] [logging.py:96:log_dist] [Rank 0] rank=0 time (ms) | forward: 113.68 | backward: 288.22 | backward_inner
: 277.97 | backward_allreduce: 10.18 | step: 61.74
{'loss': 7.6801, 'learning_rate': 5.418275829936537e-06, 'epoch': 0.0}
100%|▉| 20/20 [00:12<00:00,  2.12it/s][INFO|trainer.py:2039] 2023-07-19 17:42:02,515 >>

Training completed. Do not forget to share your model on huggingface.co/models =)


{'train_runtime': 12.1298, 'train_samples_per_second': 1.649, 'train_steps_per_second': 1.649, 'train_loss': 8.6630859375, 'epoch': 0.0}
100%|▉| 20/20 [00:12<00:00,  1.65it/s]
***** train metrics *****
  epoch                    =        0.0
  train_loss               =     8.6631
  train_runtime            = 0:00:12.12
  train_samples            =       7131
  train_samples_per_second =      1.649
  train_steps_per_second   =      1.649
[INFO|modelcard.py:451] 2023-07-19 17:42:02,517 >> Dropping the following result as it does not have all the necessary fields:
{'task': {'name': 'Causal Language Modeling', 'type': 'text-generation'}}
[root@VM-16-12-tencentos llama]#
```

Megatron-LM 训练示例

1. 安装 Megatron-LM。

   您可以执行以下命令安装 Megatron-LM。

```
yum install git -y
git clone https://github.com/NVIDIA/Megatron-LM.git && cd Megatron-LM && git checkout -b v3.0.2 v3.0.2
```

2. 安装依赖。

```
# NCCL
sudo yum-config-manager --add-repo
https://developer.download.nvidia.com/compute/cuda/repos/rhel8/x86_64/cuda-rhel8.repo
sudo yum install libnccl libnccl-devel libnccl-static



# apex
git clone https://github.com/NVIDIA/apex.git && cd apex
pip3 install -v --disable-pip-version-check --no-cache-dir --no-build-isolation --config-settings "--
build-option=--cpp_ext" --config-settings "--build-option=--cuda_ext" ./
```

3. （可选）上一步如果编译错误，需要注释掉如下代码。

```
diff --git a/setup.py b/setup.py
index b156cfa..150e548 100644
--- a/setup.py
+++ b/setup.py
@@ -174,7 +174,7 @@ if "--distributed_lamb" in sys.argv:
 if "--cuda_ext" in sys.argv:
     sys.argv.remove("--cuda_ext")
     raise_if_cuda_home_none("--cuda_ext")
-    check_cuda_torch_binary_vs_bare_metal(CUDA_HOME)
+#    check_cuda_torch_binary_vs_bare_metal(CUDA_HOME)
```

4. 安装 pybind11。

```
# pybind11
pip3 install pybind11
```

5. 下载示例代码，并做相应修改。

```
wget https://taco-1251783334.cos.ap-shanghai.myqcloud.com/demo/LLM/gpt.tar.gz
tar xzf gpt.tar.gz && cd gpt

cp hostfile download_data.sh start.sh path/to/Megatron-LM && cd path/to/Megatron-LM      #path是Megatorn-LM所在的位置
```

5.1 通过以下命令获取本机 eth0 ip，写到 hostfile 中。

```
ifconfig eth0
```

5.2 根据本机 GPU 数量修改 start.sh 文件中的 GPUS_PER_NODE 变量，默认为1。

```
GPUS_PER_NODE=1
```

6. 执行 download_data.sh 下载数据集。

```
[root@VM-16-12-tencentos Megatron-LM]# bash download_dataset.sh
gpt2-merges.txt                 100%[=============================================================>] 445.62K   547KB/s    in 0.8s
gpt2-vocab.json                 100%[=============================================================>]   1018K  1.20MB/s    in 0.8s
my-gpt2_text_document.bin       100%[=============================================================>] 446.97M  9.81MB/s    in 45s
my-gpt2_text_document.idx       100%[=============================================================>]   1.51M   180KB/s    in 9.2s
[root@VM-16-12-tencentos Megatron-LM]#
```

7. 执行 start.sh 开始训练。

```
[root@VM-16-12-tencentos Megatron-LM]# bash start.sh
torchrun --nproc_per_node 1 --nnodes 1 --node_rank 0 --master_addr 172.26.16.12 --master_port 6000 pretrain_gpt.py --tensor-model-paralle
l-size 1 --pipeline-model-parallel-size 1 --sequence-parallel --num-layers 24 --hidden-size 1024 --num-attention-heads 16 --micro-batch-s
ize 1 --global-batch-size 10 --seq-length 2048 --max-position-embeddings 2048 --train-iters 500000 --lr-decay-iters 320000 --save /root/M
egatron-LM/checkpoint --data-path /root/Megatron-LM/data/my-gpt2_text_document --vocab-file /root/Megatron-LM/data/gpt2-vocab.json --merg
e-file /root/Megatron-LM/data/gpt2-merges.txt --data-impl mmap --split 949,50,1 --distributed-backend nccl --lr 0.00015 --lr-decay-style
cosine --min-lr 1.0e-5 --weight-decay 1e-2 --clip-grad 1.0 --lr-warmup-fraction .01 --log-interval 1 --save-interval 10000 --eval-interva
l 10000 --exit-interval 10000 --eval-iters 1000 --bf16 2>&1 | tee gpt_300MB_tp1_pp{1}_dp1_bs{GLOBAL_BATCH_SIZE}.log
using world size: 1, data-parallel-size: 1, tensor-model-parallel size: 1, pipeline-model-parallel size: 1
accumulate and all-reduce gradients in fp32 for bfloat16 data type.
using torch.bfloat16 for parameters ...
------------------------ arguments ------------------------
  accumulate_allreduce_grads_in_fp32 ..... True
  adam_beta1 ..................................... 0.9
  adam_beta2 ..................................... 0.999
  adam_eps ....................................... 1e-08
  adlr_autoresume ................................ False
  adlr_autoresume_interval ....................... 1000
  apply_query_key_layer_scaling .................. True
  apply_residual_connection_post_layernorm ....... False
  async_tensor_model_parallel_allreduce .......... True
  attention_dropout .............................. 0.1
  attention_softmax_in_fp32 ...................... False
  bert_binary_head ............................... True
  bert_load ...................................... None
  bf16 ........................................... True
  bias_dropout_fusion ............................ True
  bias_gelu_fusion ............................... True
  biencoder_projection_dim ....................... 0
  biencoder_shared_query_context_model ........... False
  block_data_path ................................ None
  classes_fraction ............................... 1.0
```

```
tch-generator: 16.56
 iteration       95/     100 | consumed samples:          950 | elapsed time per iteration (ms): 6586.8 | learning rate: 4.453E-06 | glob
al batch size:    10 | lm loss: 9.037986E+00 | loss scale: 1.0 | grad norm: 2.467 | number of skipped iterations:   0 | number of nan ite
rations:   0 |
time (ms) | forward-compute: 2307.76 | backward-compute: 4236.42 | backward-params-all-reduce: 3.59 | backward-embedding-all-reduce: 0.03
 | optimizer-copy-to-main-grad: 0.87 | optimizer-clip-main-grad: 7.95 | optimizer-copy-main-to-model-params: 4.96 | optimizer: 28.36 | ba
tch-generator: 16.94
 iteration       96/     100 | consumed samples:          960 | elapsed time per iteration (ms): 6591.2 | learning rate: 4.500E-06 | glob
al batch size:    10 | lm loss: 9.025996E+00 | loss scale: 1.0 | grad norm: 2.624 | number of skipped iterations:   0 | number of nan ite
rations:   0 |
time (ms) | forward-compute: 2312.53 | backward-compute: 4236.26 | backward-params-all-reduce: 3.53 | backward-embedding-all-reduce: 0.02
 | optimizer-copy-to-main-grad: 0.70 | optimizer-clip-main-grad: 7.61 | optimizer-copy-main-to-model-params: 4.88 | optimizer: 27.74 | ba
tch-generator: 19.96
 iteration       97/     100 | consumed samples:          970 | elapsed time per iteration (ms): 6589.2 | learning rate: 4.547E-06 | glob
al batch size:    10 | lm loss: 8.975855E+00 | loss scale: 1.0 | grad norm: 3.045 | number of skipped iterations:   0 | number of nan ite
rations:   0 |
time (ms) | forward-compute: 2308.23 | backward-compute: 4238.96 | backward-params-all-reduce: 3.54 | backward-embedding-all-reduce: 0.02
 | optimizer-copy-to-main-grad: 0.74 | optimizer-clip-main-grad: 7.63 | optimizer-copy-main-to-model-params: 4.91 | optimizer: 27.84 | ba
tch-generator: 17.53
 iteration       98/     100 | consumed samples:          980 | elapsed time per iteration (ms): 6581.4 | learning rate: 4.594E-06 | glob
al batch size:    10 | lm loss: 9.115416E+00 | loss scale: 1.0 | grad norm: 2.372 | number of skipped iterations:   0 | number of nan ite
rations:   0 |
time (ms) | forward-compute: 2307.72 | backward-compute: 4231.55 | backward-params-all-reduce: 3.55 | backward-embedding-all-reduce: 0.03
 | optimizer-copy-to-main-grad: 0.77 | optimizer-clip-main-grad: 7.73 | optimizer-copy-main-to-model-params: 4.89 | optimizer: 27.92 | ba
tch-generator: 17.61
 iteration       99/     100 | consumed samples:          990 | elapsed time per iteration (ms): 6593.7 | learning rate: 4.641E-06 | glob
al batch size:    10 | lm loss: 9.044435E+00 | loss scale: 1.0 | grad norm: 2.570 | number of skipped iterations:   0 | number of nan ite
rations:   0 |
time (ms) | forward-compute: 2312.31 | backward-compute: 4238.56 | backward-params-all-reduce: 3.54 | backward-embedding-all-reduce: 0.03
 | optimizer-copy-to-main-grad: 0.81 | optimizer-clip-main-grad: 7.73 | optimizer-copy-main-to-model-params: 4.91 | optimizer: 28.02 | ba
tch-generator: 18.13
 iteration      100/     100 | consumed samples:         1000 | elapsed time per iteration (ms): 6585.2 | learning rate: 4.687E-06 | glob
al batch size:    10 | lm loss: 9.020534E+00 | loss scale: 1.0 | grad norm: 2.889 | number of skipped iterations:   0 | number of nan ite
rations:   0 |
time (ms) | forward-compute: 2310.96 | backward-compute: 4230.31 | backward-params-all-reduce: 3.57 | backward-embedding-all-reduce: 0.03
 | optimizer-copy-to-main-grad: 0.85 | optimizer-clip-main-grad: 8.04 | optimizer-copy-main-to-model-params: 5.00 | optimizer: 28.54 | ba
tch-generator: 20.49
[after training is done] datetime: 2023-07-19 20:06:36
```

TencentOS Server 3 运行主要深度学习推理框架及热门模型示例

TencentOS Server 3 运行环境及目录

最近更新时间：2024-08-12 14:44:51

本指导适用于在 TencentOS Server 3 上以 Docker 的方式运行主要深度学习推理框架及热门模型。

硬软件环境

- Description: TencentOS Server release 3.1 (Final)。
- Architecture: x86_64。
- CPU op-mode(s): 32-bit, 64-bit。
- CUDA Version: 12.2。
- GPU: NVIDIA L40。

操作系统详情：

```
LSB Version:    :core-4.1-amd64:core-4.1-noarch
Distributor ID: TencentOSServer
Description:    TencentOS Server release 3.1 (Final)
Release:        3.1
Codename:       Final
```

CPU 详情：

```
Architecture:            x86_64
CPU op-mode(s):          32-bit, 64-bit
Byte Order:              Little Endian
CPU(s):                  48
On-line CPU(s) list:     0-47
Thread(s) per core:      2
Core(s) per socket:      24
Socket(s):               1
NUMA node(s):            1
Vendor ID:               AuthenticAMD
BIOS Vendor ID:          Red Hat
CPU family:              25
Model:                   17
Model name:              AMD EPYC 9K84 96-Core Processor
BIOS Model name:         3.0
Stepping:                0
CPU MHz:                 2600.026
BogoMIPS:                5200.05
Hypervisor vendor:       KVM
Virtualization type:     full
L1d cache:               32K
L1i cache:               32K
L2 cache:                1024K
L3 cache:                32768K
NUMA node0 CPU(s):       0-47
Flags:                   fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm constant_tsc rep_good nopl nonstop_tsc cpuid extd_apicid amd_dcm tsc_known_freq
 pni pclmulqdq monitor ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm cmp_legacy cr8_legacy abm sse4a misalignsse 3dnowprefetch osvw topoext perfctr_core invpcid_single ibpb v
mmcall fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid avx512f avx512dq rdseed adx smap avx512ifma clflushopt clwb avx512cd sha_ni avx512bw avx512vl xsaveopt xsavec xgetbv1 avx512_bf16 clzero xsaveerptr wbnoinvd arat avx512vbmi umip avx512_v
bmi2 vaes vpclmulqdq avx512_vnni avx512_bitalg avx512_vpopcntdq rdpid fsrm
```

GPU 详情：

```
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.161.07             Driver Version: 535.161.07   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| 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 L40                     On  | 00000000:23:00.0 Off |                    0 |
| N/A   28C    P0              52W / 300W |      0MiB / 46068MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
```

PCI 总线设备情况：

```
-+-[0000:20]---00.0-[21-23]----00.0-[22-23]----00.0-[23]----00.0  NVIDIA Corporation AD102GL [L40]
 \-[0000:00]-+-00.0  Intel Corporation 82G33/G31/P35/P31 Express DRAM Controller
             +-01.0-[01-02]----00.0-[02]--+-01.0  Cirrus Logic GD 5446
             |                            +-02.0  Red Hat, Inc. Virtio network device
             |                            +-03.0  Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) High Definition Audio Controller
             |                            +-04.0  NEC Corporation uPD720200 USB 3.0 Host Controller
             |                            +-05.0  Red Hat, Inc. Virtio block device
             |                            \-06.0  Red Hat, Inc. Virtio memory balloon
             +-02.0-[03-04]----00.0-[04]--
             +-03.0-[05-06]----00.0-[06]--
             +-04.0  Red Hat, Inc. QEMU PCIe Expander bridge
             +-1f.0  Intel Corporation 82801IB (ICH9) LPC Interface Controller
             +-1f.2  Intel Corporation 82801IR/IO/IH (ICH9R/DO/DH) 6 port SATA Controller [AHCI mode]
             \-1f.3  Intel Corporation 82801I (ICH9 Family) SMBus Controller
```

指南目录

环境准备（必做）

- 环境准备

TendorRT

- Stable Diffusion
- ResNet
- Bert

vLLM

- opt
- LLaMA

TensorRT-LLM

- Baichuan
- ChatGLM

OpenVINO

- CLIP
- BLIP

HuggingFace TGI（Text Generation Inference）

- LLaMA
- Baichuan

LMDeploy

- ChatGLM
- LLaMA

环境准备

最近更新时间：2024-08-02 09:49:01

Docker 环境准备

安装 Docker

若有旧版本，需要删除旧版本，不然会有冲突。

```
yum remove docker* -y
```

安装 tlinux-release-docker-ce 包后，会默认使用 http://mirrors.tencent.com/docker-ce/ 目录。

```
#tlinux3.x安装docker-ce repo文件
yum install tencentos-release-docker-ce -y
yum install docker-ce -y
```

启动 Docker daemon

> ⚠ **注意：**
> 为了防止后续运行容器时出现如下错误：
> ERROR: Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?
> 该错误是由于 Docker daemon 没有正确启动导致的，需要先开启 Docker daemon。

```
#开启Docker daemon (server)
sudo systemctl start docker

#使用以下命令检查Docker daemon是否正在运行，此时应该看到Docker Client和Docker Server都在运行中
sudo systemctl status docker
```

安装 NVIDIA Container Toolkit

在容器里运行模型，如果想要在 GPU 上运行，则需要安装 NVIDIA Container Toolkit，可参见 NVIDIA Container Toolkit 安装指引 。

```
#配置安装所需要的库
curl -s -L https://nvidia.github.io/libnvidia-container/stable/rpm/nvidia-container-toolkit.repo | \
sudo tee /etc/yum.repos.d/nvidia-container-toolkit.repo

#配置库以便可以使用各种包
sudo yum-config-manager --enable nvidia-container-toolkit-experimental

#安装NVIDIA Container Toolkit包
sudo yum install -y nvidia-container-toolkit
```

配置 Docker

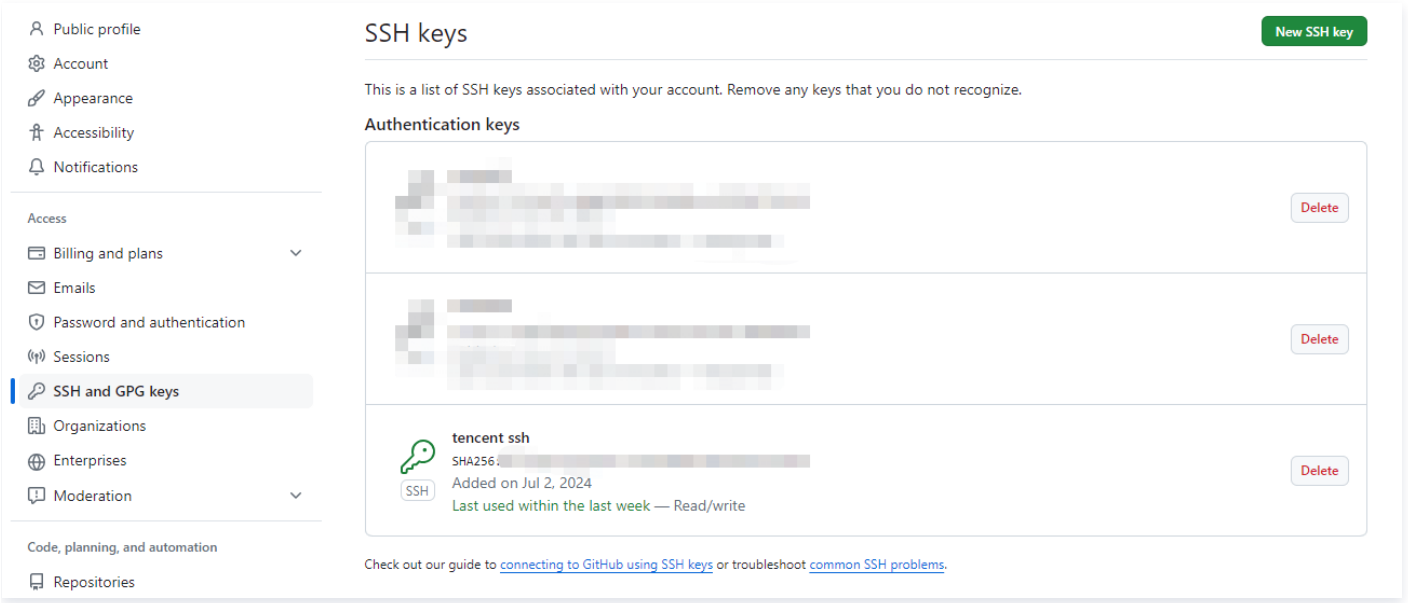
```
#使用nvidia-ctk命令修改并更新/etc/docker/daemon.json主机上的文件，以便Docker可以使用NVIDIA容器运行。
sudo nvidia-ctk runtime configure --runtime=docker

#重新启动Docker
sudo systemctl restart docker
```

配置客户端与远程 GitHub 仓库的连接

> **注意：**
> 为了防止后续拉取 GitHub 仓库代码出现如下错误：
> fatal: Could not read from remote repository.
> 则说明客户端还没有配置与远程 GitHub 仓库的连接，需要先配置 ssh 远程连接。

```
#生成密钥，其中youremail@example.com需替换成自己在GitHub上注册账号时所用的邮箱
ssh-keygen -t rsa -C "youremail@example.com"
```

打开生成的 /.ssh/id_rsa，复制密钥，在 GitHub 网站登录自己的账户，在账户选项中选择**Settings > SSH and GPG keys > New SSH key**，把密钥填写到账户中（注意填写时的格式要求）。




```
#测试客户端是否和GitHub远程连接上，出现Success则配置成功
ssh -T git@github.com
```

安装 git 包

1. 检查系统是否安装 git：

```
git
```

> **注意：**
> 如果出现 –bash: git: command not found，则说明环境还没有安装 git 包。

2. 安装 git：

```
#安装git
sudo yum -y install git

#检查git版本
git --version
```

如果此时正常出现 git 的版本号，则说明安装成功。

参考文档

- NVIDIA Container Toolkit 安装指引

- Hugging Face

TensorRT Stable Diffusion

最近更新时间：2024-08-13 09:55:21

本指导适用于在 TencentOS Server 3上使用 TensorRT 推理框架运行 Stable Diffusion 的官方 Demo，以 Docker 方式启动。

TensorRT 环境准备

1. 从 GitHub 下载 TensorRT 开源仓库到本地。

```
#下载TensorRT开源仓库到本地，版本为8.6.1
git clone git@github.com:NVIDIA/TensorRT.git -b release/8.6 --single-branch
cd TensorRT
```

2. 启动 TensorRT NGC 容器。

```
docker run -it --gpus all --name=tensorrt -e HF_ENDPOINT="https://hf-mirror.com" -v $PWD:/workspace
nvcr.io/nvidia/pytorch:23.06-py3 /bin/bash
```

此时会从 nvcr 拉取 docker 镜像，请确保网络环境较好，直到镜像里所有层下载成功。成功后会直接进入容器内部。

检查必要的包的版本

1. 确保使用较新的 pip 包和 TensorRT 包，并使用命令更新。

```
#更新pip包
python3 -m pip install --upgrade pip
```

2. 检查 TensorRT 版本，最低要求为8.6.0。

```
#检查TensorRT版本
python3 -c 'import tensorrt;print(tensorrt.__version__)'
```

本指导下载的 TensorRT 版本为8.6.1，如版本低于8.6.0，请卸载 TensorRT 并重新安装。

```
#卸载TensorRT包
python3 -m pip uninstall tensorrt

#下载新的TensorRT包，且指定版本，这里指定为8.6.1
python3 -m pip install tensorrt==8.6.1
python3 -c 'import tensorrt;print(tensorrt.__version__)'
```

下载运行框架和模型必要的包

需要安装的包位于 demo/Diffusion/requirements.txt 里，需要安装里面所有的包。

```
#安装运行框架和模型所需的包
export TRT_OSSPATH=/workspace
cd $TRT_OSSPATH/demo/Diffusion
pip3 install -r requirements.txt

#创建输出文件夹(注意一定要先创建文件夹再运行模型，否则后续会报错找不到路径)
mkdir -p onnx engine output
export onnx_dir=onnx
export engine_dir=engine
export output_dir=output
```

到此环境已经备好，所需的包也已经准备完毕。

```
#基础包              对应版本
diffusers           0.14.0
onnx                1.13.1
onnx-graphsurgeon   0.3.26
onnxruntime         1.14.1
polygraphy          0.47.1
tensorrt            8.6.1.6
tokenizers          0.13.2
torch               1.13.0
transformers        4.26.1
```

运行模型

下载模型权重地址换源

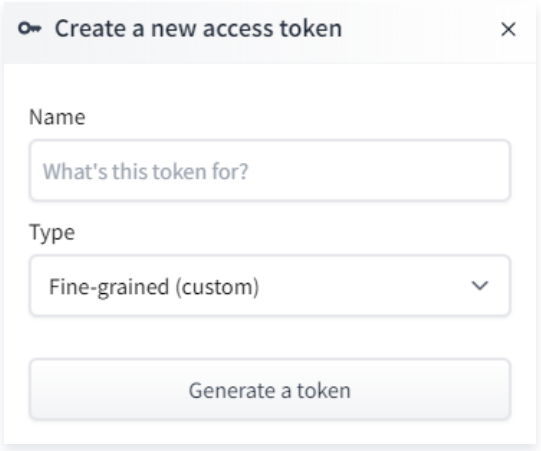
由于中国大陆无法下载 Hugging Face 网站模型，首先需要对下载网站换源，使用国内镜像网站的 HF-Mirror 模型。

> **说明：**
> 如果 docker run 的时候加上了 `-e HF_ENDPOINT="https://hf-mirror.com"`，则此步可以跳过。

```
#单次有效，退出容器且暂停容器运行后失效，再次重启进入容器需重新输入此条命令
export HF_ENDPOINT="https://hf-mirror.com"

#设为默认，永久有效，即便退出容器且暂停容器运行，再次进入容器后也可直接运行模型（推荐使用此方法）
echo 'export HF_ENDPOINT="https://hf-mirror.com"' >> ~/.bashrc
```

Stable Diffusion 模型在 Hugging Face 下载需要 User access tokens，登录 Hugging Face 网站获取 User access tokens，可参见 Hugging Face User access tokens 。

1. 单击个人账户 >Settings > Access Tokens > New token：



2. Name 输入创建的 token 名，Type 选 Fine-grained 即可，将生成的 token 输入到系统环境中。

```
#输入token，example_token为在网站上生成的token
export HF_TOKEN=example_token
```

Demo Stable Diffusion 有三种用法：

- 根据给定的 prompt 生成图像（demo_txt2img.py）。
- 根据给定的 prompt 对已有图像进行风格迁移（demo_img2img.py）。
- 根据给定的 prompt 对已有图像的mask分割区域做修改（demo_inpaint.py）。

不同用法的具体可选参数可以使用以下命令查看：

```
#查看
```

```
python3 demo_txt2img.py --help
python3 demo_img2img.py --help
python3 demo_inpaint.py --help
```

根据给定的 prompt 生成图像（demo_txt2img.py）

1. 首先，查看运行文件的可选参数：

```
python3 demo_txt2img.py --help

#可选参数如下:
prompt                Text prompt(s) to guide image generation
options:
-h, --help            show this help message and exit
--version {1.4,1.5,2.0,2.0-base,2.1,2.1-base}    Version of Stable Diffusion
--negative-prompt [NEGATIVE_PROMPT ...]
--repeat-prompt {1,2,4,8,16}    Number of times to repeat the prompt (batch size multiplier)
--height HEIGHT       Height of image to generate (must be multiple of 8)
--width WIDTH         Height of image to generate (must be multiple of 8)
--denoising-steps DENOISING_STEPS
                      Number of denoising steps
--onnx-opset {7,8,9,10,11,12,13,14,15,16,17}
                      Select ONNX opset version to target for exported models
--onnx-dir ONNX_DIR   Output directory for ONNX export
--onnx-refit-dir ONNX_REFIT_DIR
                      ONNX models to load the weights from
--force-onnx-export   Force ONNX export of CLIP, UNET, and VAE models
--force-onnx-optimize
                      Force ONNX optimizations for CLIP, UNET, and VAE models
--engine-dir ENGINE_DIR
                      Output directory for TensorRT engines
--force-engine-build  Force rebuilding the TensorRT engine
--build-static-batch  Build TensorRT engines with fixed batch size.
--build-dynamic-shape
                      Build TensorRT engines with dynamic image shapes.
--build-enable-refit  Enable Refit option in TensorRT engines during build.
--build-all-tactics   Build TensorRT engines using all tactic sources.
--timing-cache TIMING_CACHE
                      Path to the precached timing measurements to accelerate build.
--num-warmup-runs NUM_WARMUP_RUNS
                      Number of warmup runs before benchmarking performance
--nvtx-profile        Enable NVTX markers for performance profiling
--seed SEED           Seed for random generator to get consistent results
--use-cuda-graph      Enable cuda graph
--output-dir OUTPUT_DIR
                      Output directory for logs and image artifacts
--hf-token HF_TOKEN   HuggingFace API access token for downloading model checkpoints
-v, --verbose         Show verbose output
--scheduler {DDIM,EulerA,LMSD,DPM,PNDM}
                      Scheduler for diffusion process
```

2. 运行模型：

```
#根据prompt生成图像，使用的Stable Diffusion版本为2.0，生成图像的分辨率为512*512
python3 demo_txt2img.py "a beautiful photograph of Mt. Fuji during cherry blossom" --hf-token=$HF_TOKEN
--verbose --version=2.0 --onnx-dir=$onnx_dir --engine-dir=$engine_dir --output-dir=$output_dir --
seed=456123 --repeat-prompt=1 --width=512 --height=512 --num-warmup-runs=10
```

3. 输入命令后会开始下载模型权重，请确保较好的网络连接。

运行结束后可看到如下结果（参见）：

```
|------------|--------------|
|   Module   |   Latency    |
|------------|--------------|
|    CLIP    |       4.29 ms |
| UNet x 50  |     803.43 ms |
|  VAE-Dec   |      40.22 ms |
|------------|--------------|
|  Pipeline  |     846.50 ms |
|------------|--------------|
Saving image 1 / 1 to: output/txt2img-fp16-a_beautifu-1-3070.png
```

可以在 demo/Diffusion/output/ 文件夹下看到生成的图像：

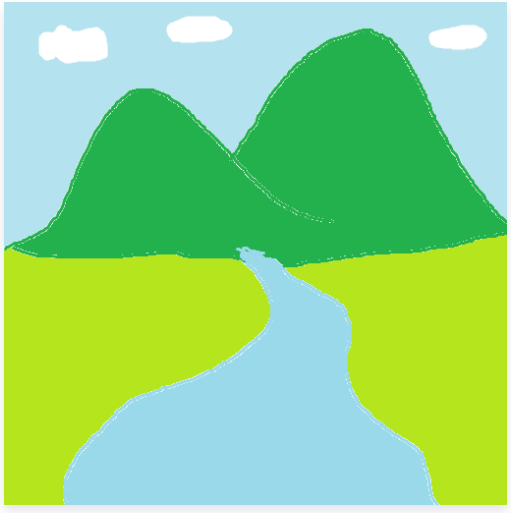
根据给定的 prompt 对已有图像进行风格迁移（demo_img2img.py）

直接运行模型

1. 直接运行的话会对网上下载的示例图像做风格迁移：

```
python3 demo_img2img.py "photorealistic new zealand hills" --hf-token=$HF_TOKEN --verbose --version=2.0
--onnx-dir=$onnx_dir --engine-dir=$engine_dir --output-dir=$output_dir --seed=456123 --repeat-prompt=1
--width=512 --height=512 --num-warmup-runs=10
```

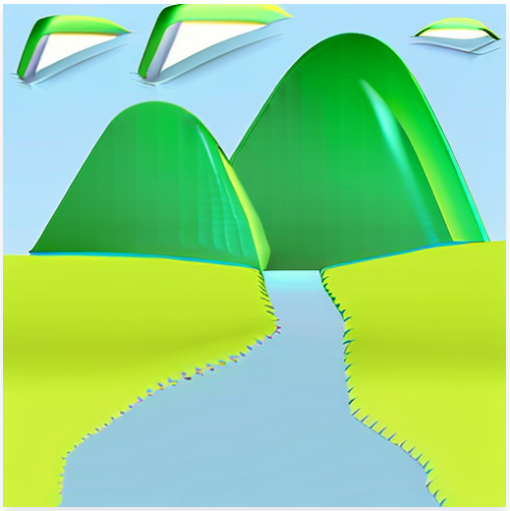
示例图像：



2. 运行结束后可看到如下结果（参见）：

```
|------------|--------------|
|   Module   |   Latency    |
|------------|--------------|
|  VAE-Enc   |     12.92 ms |
|    CLIP    |      4.56 ms |
| UNet x 50  |    629.90 ms |
|  VAE-Dec   |     26.67 ms |
|------------|--------------|
|  Pipeline  |    674.55 ms |
|------------|--------------|
Saving image 1 / 1 to: output/img2img-fp16-photoreali-1-3321.png
```

根据输入的 prompt 做风格迁移后的图像：



选择自己的图像进行风格迁移

除了对示例图像做风格迁移，还可以对自己已有的任何图像做风格迁移，需要添加的参数为 --input-image。
例如在 output 文件夹下已有图像：

路径为 demo/Diffusion/output/txt2img-fp16-a_beautifu-1-2703.png。

```
#将图像变为抽象风格（注意输出路径的写法）
python3 demo_img2img.py "abstractive the Bund of Shanghai" --hf-token=$HF_TOKEN --verbose --version=2.0 --
onnx-dir=$onnx_dir --engine-dir=$engine_dir --output-dir=$output_dir --seed=456123 --repeat-prompt=1 --
width=512 --height=512 --num-warmup-runs=10 --input-image="$output_dir/txt2img-fp16-a_beautifu-1-2703.png"
```

运行结束后可看到如下结果（参见）：

```
|------------|--------------|
|   Module   |   Latency    |
|------------|--------------|
|  VAE-Enc   |     13.53 ms |
|    CLIP    |      4.48 ms |
| UNet x 50  |    621.11 ms |
|  VAE-Dec   |     26.97 ms |
|------------|--------------|
|  Pipeline  |    666.61 ms |
|------------|--------------|
Saving image 1 / 1 to: output/img2img-fp16-abstractiv-1-8447.png
```

根据输入的 prompt 做风格迁移后的图像：

根据给定的 prompt 对已有图像的 mask 分割区域做修改（demo_inpaint.py）

直接运行模型：

直接运行的话会对网上下载的示例图像和 mask 分割图像区域做修改：

```
python3 demo_inpaint.py "a mecha robot sitting on a bench" --hf-token=$HF_TOKEN --verbose --version=2.0 --
onnx-dir=$onnx_dir --engine-dir=$engine_dir --output-dir=$output_dir --seed=456123 --repeat-prompt=1 --
width=512 --height=512 --num-warmup-runs=10
```

示例图像：

mask 分割图像：

运行结束后可看到如下结果（参见）：

```
|------------|--------------|
|   Module   |   Latency    |
|------------|--------------|
|  VAE-Enc   |     12.60 ms |
|    CLIP    |      4.41 ms |
| UNet x 50  |    634.08 ms |
|  VAE-Dec   |     26.25 ms |
|------------|--------------|
|  Pipeline  |    680.40 ms |
|------------|--------------|
Saving image 1 / 1 to: output/inpaint-fp16-a_mecha_ro-1-9059.png
```

修改后的图像：

选择自己的图像进行修改：

选择自己的图像进行修改时，需要加上 **--input-image=<path to image>** 和 **--mask-image=<path to mask>** 两个参数，请确保示例图像和 mask 分割 label 图像具有同样的图像分辨率（高度和宽度相同）。如果两张图像分辨率不同，则会运行网上下载的图像进行模型推理。

注意事项

在切换不同版本的 Stable Diffusion 模型推理时，onnx 文件夹和 engine 文件夹需要新建，来存储不同版本模型的中间状态。

```
#使用1.5版本的Stable Diffusion时
mkdir -p onnx-1.5 engine-1.5
export onnx_dir=onnx-1.5
export engine_dir=engine-1.5

python3 demo_inpaint.py "a mecha robot sitting on a bench" --hf-token=$HF_TOKEN --verbose --version=1.5 --
onnx-dir=$onnx_dir --engine-dir=$engine_dir --output-dir=$output_dir --seed=456123 --repeat-prompt=1 --
width=512 --height=512 --num-warmup-runs=10
```

说明：
由于 OpenCloudOS 是 TencentOS Server 的开源版本，理论上上述文档当中的所有操作同样适用于 OpenCloudOS。

参考文档

- TensorRT Diffusion Demo
- Hugging Face 镜像网站
- Hugging Face User access tokens

ResNet

最近更新时间：2025-05-12 11:03:42

本指导适用于在 TencentOS Server 3 上使用 TensorRT 推理框架运行 ResNet 模型，以 Docker 方式启动。

TensorRT 环境准备

1. 从 GitHub 下载 TensorRT 开源仓库到本地。

```
#下载TensorRT开源仓库到本地，版本为8.6.1
git clone git@github.com:NVIDIA/TensorRT.git -b release/8.6 --single-branch
cd TensorRT
```

2. 启动 TensorRT NGC 容器。

```
docker run -it --gpus all --name=tensorrt -e HF_ENDPOINT="https://hf-mirror.com" -v $PWD:/workspace
nvcr.io/nvidia/pytorch:23.06-py3 /bin/bash
```

此时会从 nvcr 拉取 docker 镜像，请确保网络环境较好，直到镜像里所有层下载成功。成功后会直接进入容器内部。

检查必要的包的版本

1. 确保使用较新的 pip 包和 TensorRT 包，并使用命令更新。

```
#更新pip包
python3 -m pip install --upgrade pip
```

2. 检查 TensorRT 版本，最低要求为8.6.0。

```
#检查TensorRT版本
python3 -c 'import tensorrt;print(tensorrt.__version__)'
```

本指导下载的 TensorRT 版本为8.6.1，如版本低于8.6.0，请卸载 TensorRT 并重新安装。

```
#卸载TensorRT包
python3 -m pip uninstall tensorrt

#下载新的TensorRT包，且指定版本，这里指定为8.6.1
python3 -m pip install tensorrt==8.6.1
python3 -c 'import tensorrt;print(tensorrt.__version__)'
```

下载运行框架和模型必要的包

1. TensorRT 没有官方 ResNet demo 代码，我们自己创建代码以及安装运行模型需要的包。

```
#运行文件夹构建
export TRT_OSSPATH=/workspace
cd $TRT_OSSPATH/demo/
mkdir ResNet
cd ResNet
```

2. 安装运行框架和模型所需的包。

```
#安装torch2trt包，方便后续模型从torch转为TensorRT形式
git clone https://github.com/NVIDIA-AI-IOT/torch2trt
cd torch2trt
```

```
python setup.py install

#检查torch2trt是否安装好，安装好的话能看到对应的版本号等信息
pip show torch2trt
```

3. 安装好的话可以看到如下信息（参考）：

```
Name: torch2trt
Version: 0.5.0
Summary: An easy to use PyTorch to TensorRT converter
Home-page:
Author:
Author-email:
License: UNKNOWN
Location: /usr/local/lib/python3.10/dist-packages/torch2trt-0.5.0-py3.10.egg
Requires:
Required-by:
```

运行模型

回到 ResNet 文件夹下，创建 ResNet.py 文件，写入以下代码，本指导使用 Pytorch 和 TensorRT 两种方式运行 ResNet-50 模型：

```
import torch
import torch.nn.functional as F
import os
import requests
from pathlib import Path
from PIL import Image
import time
from torch2trt import torch2trt
from torch2trt import TRTModule
from torchvision.models.resnet import resnet50
from torchvision import transforms


transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(
    mean=[0.485, 0.456, 0.406],
    std=[0.229, 0.224, 0.225]
)])


# inference using Pytorch
# setup test input: download image and label file
sample_path = Path("data/cats_image.jpeg")
label_path = Path("data/imagenet_classes.txt")
model_trt_path = Path("model/resnet50_trt.pth")


# download image
if os.path.exists(sample_path):
    print("sample exists.")
else:
    print("download sample.")
    sample_path.parent.mkdir(parents=True, exist_ok=True)
    r = requests.get("https://hf-mirror.com/datasets/huggingface/cats-image/resolve/main/cats_image.jpeg")
```

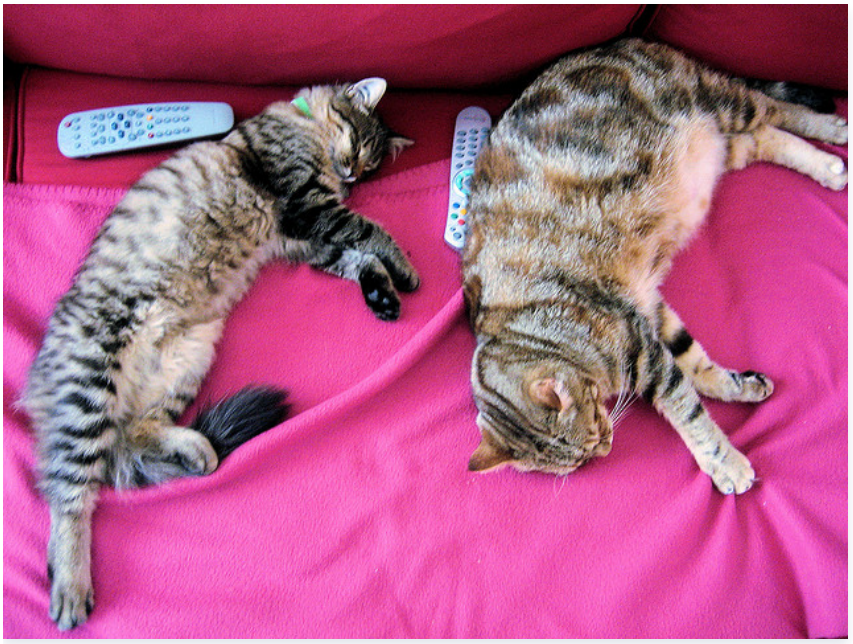
```
    with sample_path.open("wb") as f:
        f.write(r.content)


# download label file
if os.path.exists(label_path):
    print("label file exists.")
else:
    print("download label file.")
    label_path.parent.mkdir(parents=True, exist_ok=True)
    r =
requests.get("https://raw.githubusercontent.com/Lasagne/Recipes/master/examples/resnet50/imagenet_classes.
txt")


    with label_path.open("wb") as f:
        f.write(r.content)


# open image
image = Image.open(sample_path).convert("RGB")
inputs = transform(image)
batch_inputs = torch.unsqueeze(inputs, 0).cuda()


# load model and labels
model = resnet50(pretrained=True).eval().cuda()


with open('data/imagenet_classes.txt') as f:
  classes = [line.strip() for line in f.readlines()]


# inference by Pytorch
start = time.perf_counter()
output_logits = model(batch_inputs)
end = time.perf_counter() - start


output_prob = F.softmax(output_logits, dim=-1)
predicted_class_indices = torch.argmax(output_prob, dim=-1)
predicted_class = classes[predicted_class_indices]
print(f"Pytorch Predict class: {predicted_class}")
print(f"Pytorch Processing time: {end:.4f}")




# inference by TensorRT
# load TensorRT model
if os.path.exists(model_trt_path):
    print("TensorRT model exists.")
    model_trt = TRTModule()
    model_trt.load_state_dict(torch.load(model_trt_path))
else:
    print("Convert and save TensorRT model.")
    model_trt_path.parent.mkdir(parents=True, exist_ok=True)
```

```
    # Build TensorRT engine
    model_trt = torch2trt(model, [batch_inputs], fp16_mode=True).cuda()
    # save TensorRT model
    torch.save(model_trt.state_dict(), model_trt_path)


start = time.perf_counter()
output_trt_logits = model_trt(batch_inputs)
end = time.perf_counter() - start


output_trt_prob = F.softmax(output_trt_logits, dim=-1)
trt_predicted_class_indices = torch.argmax(output_trt_prob, dim=-1)
trt_predicted_class = classes[trt_predicted_class_indices]
print(f"TensorRT Predict class: {trt_predicted_class}")
print(f"TensorRT Processing time: {end:.4f}")
```

随后运行该代码：

```
python ResNet.py
```

运行后会先下载测试图像，label 名文件以及网络模型权重。这里我们使用 ResNet-50 模型进行图像分类任务，分类测试图像如下：



该测试图像来源于 ImageNet-1K，而模型最后一层是从2048通过fc层降维到1000，所以我们需要 ImageNet 的 label 文件 imagenet_classes.txt 来确定分类结果，1000个 label 名称如下：

```
tench, Tinca tinca
goldfish, Carassius auratus
great white shark, white shark, man-eater, man-eating shark, Carcharodon carcharias
tiger shark, Galeocerdo cuvieri
hammerhead, hammerhead shark
electric ray, crampfish, numbfish, torpedo
stingray
cock
hen
ostrich, Struthio camelus
brambling, Fringilla montifringilla
goldfinch, Carduelis carduelis
house finch, linnet, Carpodacus mexicanus
```

```
junco, snowbird
indigo bunting, indigo finch, indigo bird, Passerina cyanea
robin, American robin, Turdus migratorius
bulbul
jay
magpie
chickadee
water ouzel, dipper
kite
bald eagle, American eagle, Haliaeetus leucocephalus
vulture
great grey owl, great gray owl, Strix nebulosa
European fire salamander, Salamandra salamandra
common newt, Triturus vulgaris
eft
spotted salamander, Ambystoma maculatum
axolotl, mud puppy, Ambystoma mexicanum
bullfrog, Rana catesbeiana
tree frog, tree-frog
tailed frog, bell toad, ribbed toad, tailed toad, Ascaphus trui
loggerhead, loggerhead turtle, Caretta caretta
leatherback turtle, leatherback, leathery turtle, Dermochelys coriacea
mud turtle
terrapin
box turtle, box tortoise
banded gecko
common iguana, iguana, Iguana iguana
American chameleon, anole, Anolis carolinensis
whiptail, whiptail lizard
agama
frilled lizard, Chlamydosaurus kingi
alligator lizard
Gila monster, Heloderma suspectum
green lizard, Lacerta viridis
African chameleon, Chamaeleo chamaeleon
Komodo dragon, Komodo lizard, dragon lizard, giant lizard, Varanus komodoensis
African crocodile, Nile crocodile, Crocodylus niloticus
American alligator, Alligator mississipiensis
triceratops
thunder snake, worm snake, Carphophis amoenus
ringneck snake, ring-necked snake, ring snake
hognose snake, puff adder, sand viper
green snake, grass snake
king snake, kingsnake
garter snake, grass snake
water snake
vine snake
night snake, Hypsiglena torquata
boa constrictor, Constrictor constrictor
rock python, rock snake, Python sebae
Indian cobra, Naja naja
green mamba
sea snake
horned viper, cerastes, sand viper, horned asp, Cerastes cornutus
diamondback, diamondback rattlesnake, Crotalus adamanteus
sidewinder, horned rattlesnake, Crotalus cerastes
trilobite
harvestman, daddy longlegs, Phalangium opilio
scorpion
black and gold garden spider, Argiope aurantia
barn spider, Araneus cavaticus
```

```
garden spider, Aranea diadematа
black widow, Latrodectus mactans
tarantula
wolf spider, hunting spider
tick
centipede
black grouse
ptarmigan
ruffed grouse, partridge, Bonasa umbellus
prairie chicken, prairie grouse, prairie fowl
peacock
quail
partridge
African grey, African gray, Psittacus erithacus
macaw
sulphur-crested cockatoo, Kakatoe galerita, Cacatua galerita
lorikeet
coucal
bee eater
hornbill
hummingbird
jacamar
toucan
drake
red-breasted merganser, Mergus serrator
goose
black swan, Cygnus atratus
tusker
echidna, spiny anteater, anteater
platypus, duckbill, duckbilled platypus, duck-billed platypus, Ornithorhynchus anatinus
wallaby, brush kangaroo
koala, koala bear, kangaroo bear, native bear, Phascolarctos cinereus
wombat
jellyfish
sea anemone, anemone
brain coral
flatworm, platyhelminth
nematode, nematode worm, roundworm
conch
snail
slug
sea slug, nudibranch
chiton, coat-of-mail shell, sea cradle, polyplacophore
chambered nautilus, pearly nautilus, nautilus
Dungeness crab, Cancer magister
rock crab, Cancer irroratus
fiddler crab
king crab, Alaska crab, Alaskan king crab, Alaska king crab, Paralithodes camtschatica
American lobster, Northern lobster, Maine lobster, Homarus americanus
spiny lobster, langouste, rock lobster, crawfish, crayfish, sea crawfish
crayfish, crawfish, crawdad, crawdaddy
hermit crab
isopod
white stork, Ciconia ciconia
black stork, Ciconia nigra
spoonbill
flamingo
little blue heron, Egretta caerulea
American egret, great white heron, Egretta albus
bittern
crane
```

```
limpkin, Aramus pictus
European gallinule, Porphyrio porphyrio
American coot, marsh hen, mud hen, water hen, Fulica americana
bustard
ruddy turnstone, Arenaria interpres
red-backed sandpiper, dunlin, Erolia alpina
redshank, Tringa totanus
dowitcher
oystercatcher, oyster catcher
pelican
king penguin, Aptenodytes patagonica
albatross, mollymawk
grey whale, gray whale, devilfish, Eschrichtius gibbosus, Eschrichtius robustus
killer whale, killer, orca, grampus, sea wolf, Orcinus orca
dugong, Dugong dugon
sea lion
Chihuahua
Japanese spaniel
Maltese dog, Maltese terrier, Maltese
Pekinese, Pekingese, Peke
Shih-Tzu
Blenheim spaniel
papillon
toy terrier
Rhodesian ridgeback
Afghan hound, Afghan
basset, basset hound
beagle
bloodhound, sleuthhound
bluetick
black-and-tan coonhound
Walker hound, Walker foxhound
English foxhound
redbone
borzoi, Russian wolfhound
Irish wolfhound
Italian greyhound
whippet
Ibizan hound, Ibizan Podenco
Norwegian elkhound, elkhound
otterhound, otter hound
Saluki, gazelle hound
Scottish deerhound, deerhound
Weimaraner
Staffordshire bullterrier, Staffordshire bull terrier
American Staffordshire terrier, Staffordshire terrier, American pit bull terrier, pit bull terrier
Bedlington terrier
Border terrier
Kerry blue terrier
Irish terrier
Norfolk terrier
Norwich terrier
Yorkshire terrier
wire-haired fox terrier
Lakeland terrier
Sealyham terrier, Sealyham
Airedale, Airedale terrier
cairn, cairn terrier
Australian terrier
Dandie Dinmont, Dandie Dinmont terrier
Boston bull, Boston terrier
```

```
miniature schnauzer
giant schnauzer
standard schnauzer
Scotch terrier, Scottish terrier, Scottie
Xizang terrier, chrysanthemum dog
silky terrier, Sydney silky
soft-coated wheaten terrier
West Highland white terrier
Lhasa, Lhasa apso
flat-coated retriever
curly-coated retriever
golden retriever
Labrador retriever
Chesapeake Bay retriever
German short-haired pointer
vizsla, Hungarian pointer
English setter
Irish setter, red setter
Gordon setter
Brittany spaniel
clumber, clumber spaniel
English springer, English springer spaniel
Welsh springer spaniel
cocker spaniel, English cocker spaniel, cocker
Sussex spaniel
Irish water spaniel
kuvasz
schipperke
groenendael
malinois
briard
kelpie
komondor
Old English sheepdog, bobtail
Shetland sheepdog, Shetland sheep dog, Shetland
collie
Border collie
Bouvier des Flandres, Bouviers des Flandres
Rottweiler
German shepherd, German shepherd dog, German police dog, alsatian
Doberman, Doberman pinscher
miniature pinscher
Greater Swiss Mountain dog
Bernese mountain dog
Appenzeller
EntleBucher
boxer
bull mastiff
Xizang mastiff
French bulldog
Great Dane
Saint Bernard, St Bernard
Eskimo dog, husky
malamute, malemute, Alaskan malamute
Siberian husky
dalmatian, coach dog, carriage dog
affenpinscher, monkey pinscher, monkey dog
basenji
pug, pug-dog
Leonberg
Newfoundland, Newfoundland dog
```

```
Great Pyrenees
Samoyed, Samoyede
Pomeranian
chow, chow chow
keeshond
Brabancon griffon
Pembroke, Pembroke Welsh corgi
Cardigan, Cardigan Welsh corgi
toy poodle
miniature poodle
standard poodle
Mexican hairless
timber wolf, grey wolf, gray wolf, Canis lupus
white wolf, Arctic wolf, Canis lupus tundrarum
red wolf, maned wolf, Canis rufus, Canis niger
coyote, prairie wolf, brush wolf, Canis latrans
dingo, warrigal, warragal, Canis dingo
dhole, Cuon alpinus
African hunting dog, hyena dog, Cape hunting dog, Lycaon pictus
hyena, hyaena
red fox, Vulpes vulpes
kit fox, Vulpes macrotis
Arctic fox, white fox, Alopex lagopus
grey fox, gray fox, Urocyon cinereoargenteus
tabby, tabby cat
tiger cat
Persian cat
Siamese cat, Siamese
Egyptian cat
cougar, puma, catamount, mountain lion, painter, panther, Felis concolor
lynx, catamount
leopard, Panthera pardus
snow leopard, ounce, Panthera uncia
jaguar, panther, Panthera onca, Felis onca
lion, king of beasts, Panthera leo
tiger, Panthera tigris
cheetah, chetah, Acinonyx jubatus
brown bear, bruin, Ursus arctos
American black bear, black bear, Ursus americanus, Euarctos americanus
ice bear, polar bear, Ursus Maritimus, Thalarctos maritimus
sloth bear, Melursus ursinus, Ursus ursinus
mongoose
meerkat, mierkat
tiger beetle
ladybug, ladybeetle, lady beetle, ladybird, ladybird beetle
ground beetle, carabid beetle
long-horned beetle, longicorn, longicorn beetle
leaf beetle, chrysomelid
dung beetle
rhinoceros beetle
weevil
fly
bee
ant, emmet, pismire
grasshopper, hopper
cricket
walking stick, walkingstick, stick insect
cockroach, roach
mantis, mantid
cicada, cicala
leafhopper
```

```
lacewing, lacewing fly
dragonfly, darning needle, devil's darning needle, sewing needle, snake feeder, snake doctor, mosquito
hawk, skeeter hawk
damselfly
admiral
ringlet, ringlet butterfly
monarch, monarch butterfly, milkweed butterfly, Danaus plexippus
cabbage butterfly
sulphur butterfly, sulfur butterfly
lycaenid, lycaenid butterfly
starfish, sea star
sea urchin
sea cucumber, holothurian
wood rabbit, cottontail, cottontail rabbit
hare
Angora, Angora rabbit
hamster
porcupine, hedgehog
fox squirrel, eastern fox squirrel, Sciurus niger
marmot
beaver
guinea pig, Cavia cobaya
sorrel
zebra
hog, pig, grunter, squealer, Sus scrofa
wild boar, boar, Sus scrofa
warthog
hippopotamus, hippo, river horse, Hippopotamus amphibius
ox
water buffalo, water ox, Asiatic buffalo, Bubalus bubalis
bison
ram, tup
bighorn, bighorn sheep, cimarron, Rocky Mountain bighorn, Rocky Mountain sheep, Ovis canadensis
ibex, Capra ibex
hartebeest
impala, Aepyceros melampus
gazelle
Arabian camel, dromedary, Camelus dromedarius
llama
weasel
mink
polecat, fitch, foulmart, foumart, Mustela putorius
black-footed ferret, ferret, Mustela nigripes
otter
skunk, polecat, wood pussy
badger
armadillo
three-toed sloth, ai, Bradypus tridactylus
orangutan, orang, orangutang, Pongo pygmaeus
gorilla, Gorilla gorilla
chimpanzee, chimp, Pan troglodytes
gibbon, Hylobates lar
siamang, Hylobates syndactylus, Symphalangus syndactylus
guenon, guenon monkey
patas, hussar monkey, Erythrocebus patas
baboon
macaque
langur
colobus, colobus monkey
proboscis monkey, Nasalis larvatus
marmoset
```

```
capuchin, ringtail, Cebus capucinus
howler monkey, howler
titi, titi monkey
spider monkey, Ateles geoffroyi
squirrel monkey, Saimiri sciureus
Madagascar cat, ring-tailed lemur, Lemur catta
indri, indris, Indri indri, Indri brevicaudatus
Indian elephant, Elephas maximus
African elephant, Loxodonta africana
lesser panda, red panda, panda, bear cat, cat bear, Ailurus fulgens
giant panda, panda, panda bear, coon bear, Ailuropoda melanoleuca
barracouta, snoek
eel
coho, cohoe, coho salmon, blue jack, silver salmon, Oncorhynchus kisutch
rock beauty, Holocanthus tricolor
anemone fish
sturgeon
gar, garfish, garpike, billfish, Lepisosteus osseus
lionfish
puffer, pufferfish, blowfish, globefish
abacus
abaya
academic gown, academic robe, judge's robe
accordion, piano accordion, squeeze box
acoustic guitar
aircraft carrier, carrier, flattop, attack aircraft carrier
airliner
airship, dirigible
altar
ambulance
amphibian, amphibious vehicle
analog clock
apiary, bee house
apron
ashcan, trash can, garbage can, wastebin, ash bin, ash-bin, ashbin, dustbin, trash barrel, trash bin
assault rifle, assault gun
backpack, back pack, knapsack, packsack, rucksack, haversack
bakery, bakeshop, bakehouse
balance beam, beam
balloon
ballpoint, ballpoint pen, ballpen, Biro
Band Aid
banjo
bannister, banister, balustrade, balusters, handrail
barbell
barber chair
barbershop
barn
barometer
barrel, cask
barrow, garden cart, lawn cart, wheelbarrow
baseball
basketball
bassinet
bassoon
bathing cap, swimming cap
bath towel
bathtub, bathing tub, bath, tub
beach wagon, station wagon, wagon, estate car, beach waggon, station waggon, waggon
beacon, lighthouse, beacon light, pharos
beaker
```

```
bearskin, busby, shako
beer bottle
beer glass
bell cote, bell cot
bib
bicycle-built-for-two, tandem bicycle, tandem
bikini, two-piece
binder, ring-binder
binoculars, field glasses, opera glasses
birdhouse
boathouse
bobsled, bobsleigh, bob
bolo tie, bolo, bola tie, bola
bonnet, poke bonnet
bookcase
bookshop, bookstore, bookstall
bottlecap
bow
bow tie, bow-tie, bowtie
brass, memorial tablet, plaque
brassiere, bra, bandeau
breakwater, groin, groyne, mole, bulwark, seawall, jetty
breastplate, aegis, egis
broom
bucket, pail
buckle
bulletproof vest
bullet train, bullet
butcher shop, meat market
cab, hack, taxi, taxicab
caldron, cauldron
candle, taper, wax light
cannon
canoe
can opener, tin opener
cardigan
car mirror
carousel, carrousel, merry-go-round, roundabout, whirligig
carpenter's kit, tool kit
carton
car wheel
cash machine, cash dispenser, automated teller machine, automatic teller machine, automated teller,
automatic teller, ATM
cassette
cassette player
castle
catamaran
CD player
cello, violoncello
cellular telephone, cellular phone, cellphone, cell, mobile phone
chain
chainlink fence
chain mail, ring mail, mail, chain armor, chain armour, ring armor, ring armour
chain saw, chainsaw
chest
chiffonier, commode
chime, bell, gong
china cabinet, china closet
Christmas stocking
church, church building
cinema, movie theater, movie theatre, movie house, picture palace
```

```
cleaver, meat cleaver, chopper
cliff dwelling
cloak
clog, geta, patten, sabot
cocktail shaker
coffee mug
coffeepot
coil, spiral, volute, whorl, helix
combination lock
computer keyboard, keypad
confectionery, confectionary, candy store
container ship, containership, container vessel
convertible
corkscrew, bottle screw
cornet, horn, trumpet, trump
cowboy boot
cowboy hat, ten-gallon hat
cradle
crane
crash helmet
crate
crib, cot
Crock Pot
croquet ball
crutch
cuirass
dam, dike, dyke
desk
desktop computer
dial telephone, dial phone
diaper, nappy, napkin
digital clock
digital watch
dining table, board
dishrag, dishcloth
dishwasher, dish washer, dishwashing machine
disk brake, disc brake
dock, dockage, docking facility
dogsled, dog sled, dog sleigh
dome
doormat, welcome mat
drilling platform, offshore rig
drum, membranophone, tympan
drumstick
dumbbell
Dutch oven
electric fan, blower
electric guitar
electric locomotive
entertainment center
envelope
espresso maker
face powder
feather boa, boa
file, file cabinet, filing cabinet
fireboat
fire engine, fire truck
fire screen, fireguard
flagpole, flagstaff
flute, transverse flute
folding chair
```

```
football helmet
forklift
fountain
fountain pen
four-poster
freight car
French horn, horn
frying pan, frypan, skillet
fur coat
garbage truck, dustcart
gasmask, respirator, gas helmet
gas pump, gasoline pump, petrol pump, island dispenser
goblet
go-kart
golf ball
golfcart, golf cart
gondola
gong, tam-tam
gown
grand piano, grand
greenhouse, nursery, glasshouse
grille, radiator grille
grocery store, grocery, food market, market
guillotine
hair slide
hair spray
half track
hammer
hamper
hand blower, blow dryer, blow drier, hair dryer, hair drier
hand-held computer, hand-held microcomputer
handkerchief, hankie, hanky, hankey
hard disc, hard disk, fixed disk
harmonica, mouth organ, harp, mouth harp
harp
harvester, reaper
hatchet
holster
home theater, home theatre
honeycomb
hook, claw
hoopskirt, crinoline
horizontal bar, high bar
horse cart, horse-cart
hourglass
iPod
iron, smoothing iron
jack-o'-lantern
jean, blue jean, denim
jeep, landrover
jersey, T-shirt, tee shirt
jigsaw puzzle
jinrikisha, ricksha, rickshaw
joystick
kimono
knee pad
knot
lab coat, laboratory coat
ladle
lampshade, lamp shade
laptop, laptop computer
```

```
lawn mower, mower
lens cap, lens cover
letter opener, paper knife, paperknife
library
lifeboat
lighter, light, igniter, ignitor
limousine, limo
liner, ocean liner
lipstick, lip rouge
Loafer
lotion
loudspeaker, speaker, speaker unit, loudspeaker system, speaker system
loupe, jeweler's loupe
lumbermill, sawmill
magnetic compass
mailbag, postbag
mailbox, letter box
maillot
maillot, tank suit
manhole cover
maraca
marimba, xylophone
mask
matchstick
maypole
maze, labyrinth
measuring cup
medicine chest, medicine cabinet
megalith, megalithic structure
microphone, mike
microwave, microwave oven
military uniform
milk can
minibus
miniskirt, mini
minivan
missile
mitten
mixing bowl
mobile home, manufactured home
Model T
modem
monastery
monitor
moped
mortar
mortarboard
mosque
mosquito net
motor scooter, scooter
mountain bike, all-terrain bike, off-roader
mountain tent
mouse, computer mouse
mousetrap
moving van
muzzle
nail
neck brace
necklace
nipple
notebook, notebook computer
```

```
obelisk
oboe, hautboy, hautbois
ocarina, sweet potato
odometer, hodometer, mileometer, milometer
oil filter
organ, pipe organ
oscilloscope, scope, cathode-ray oscilloscope, CRO
overskirt
oxcart
oxygen mask
packet
paddle, boat paddle
paddlewheel, paddle wheel
padlock
paintbrush
pajama, pyjama, pj's, jammies
palace
panpipe, pandean pipe, syrinx
paper towel
parachute, chute
parallel bars, bars
park bench
parking meter
passenger car, coach, carriage
patio, terrace
pay-phone, pay-station
pedestal, plinth, footstall
pencil box, pencil case
pencil sharpener
perfume, essence
Petri dish
photocopier
pick, plectrum, plectron
pickelhaube
picket fence, paling
pickup, pickup truck
pier
piggy bank, penny bank
pill bottle
pillow
ping-pong ball
pinwheel
pirate, pirate ship
pitcher, ewer
plane, carpenter's plane, woodworking plane
planetarium
plastic bag
plate rack
plow, plough
plunger, plumber's helper
Polaroid camera, Polaroid Land camera
pole
police van, police wagon, paddy wagon, patrol wagon, wagon, black Maria
poncho
pool table, billiard table, snooker table
pop bottle, soda bottle
pot, flowerpot
potter's wheel
power drill
prayer rug, prayer mat
printer
```

```
prison, prison house
projectile, missile
projector
puck, hockey puck
punching bag, punch bag, punching ball, punchball
purse
quill, quill pen
quilt, comforter, comfort, puff
racer, race car, racing car
racket, racquet
radiator
radio, wireless
radio telescope, radio reflector
rain barrel
recreational vehicle, RV, R.V.
reel
reflex camera
refrigerator, icebox
remote control, remote
restaurant, eating house, eating place, eatery
revolver, six-gun, six-shooter
rifle
rocking chair, rocker
rotisserie
rubber eraser, rubber, pencil eraser
rugby ball
rule, ruler
running shoe
safe
safety pin
saltshaker, salt shaker
sandal
sarong
sax, saxophone
scabbard
scale, weighing machine
school bus
schooner
scoreboard
screen, CRT screen
screw
screwdriver
seat belt, seatbelt
sewing machine
shield, buckler
shoe shop, shoe-shop, shoe store
shoji
shopping basket
shopping cart
shovel
shower cap
shower curtain
ski
ski mask
sleeping bag
slide rule, slipstick
sliding door
slot, one-armed bandit
snorkel
snowmobile
snowplow, snowplough
```

```
soap dispenser
soccer ball
sock
solar dish, solar collector, solar furnace
sombrero
soup bowl
space bar
space heater
space shuttle
spatula
speedboat
spider web, spider's web
spindle
sports car, sport car
spotlight, spot
stage
steam locomotive
steel arch bridge
steel drum
stethoscope
stole
stone wall
stopwatch, stop watch
stove
strainer
streetcar, tram, tramcar, trolley, trolley car
stretcher
studio couch, day bed
stupa, tope
submarine, pigboat, sub, U-boat
suit, suit of clothes
sundial
sunglass
sunglasses, dark glasses, shades
sunscreen, sunblock, sun blocker
suspension bridge
swab, swob, mop
sweatshirt
swimming trunks, bathing trunks
swing
switch, electric switch, electrical switch
syringe
table lamp
tank, army tank, armored combat vehicle, armoured combat vehicle
tape player
teapot
teddy, teddy bear
television, television system
tennis ball
thatch, thatched roof
theater curtain, theatre curtain
thimble
thresher, thrasher, threshing machine
throne
tile roof
toaster
tobacco shop, tobacconist shop, tobacconist
toilet seat
torch
totem pole
tow truck, tow car, wrecker
```

```
toyshop
tractor
trailer truck, tractor trailer, trucking rig, rig, articulated lorry, semi
tray
trench coat
tricycle, trike, velocipede
trimaran
tripod
triumphal arch
trolleybus, trolley coach, trackless trolley
trombone
tub, vat
turnstile
typewriter keyboard
umbrella
unicycle, monocycle
upright, upright piano
vacuum, vacuum cleaner
vase
vault
velvet
vending machine
vestment
viaduct
violin, fiddle
volleyball
waffle iron
wall clock
wallet, billfold, notecase, pocketbook
wardrobe, closet, press
warplane, military plane
washbasin, handbasin, washbowl, lavabo, wash-hand basin
washer, automatic washer, washing machine
water bottle
water jug
water tower
whiskey jug
whistle
wig
window screen
window shade
Windsor tie
wine bottle
wing
wok
wooden spoon
wool, woolen, woollen
worm fence, snake fence, snake-rail fence, Virginia fence
wreck
yawl
yurt
web site, website, internet site, site
comic book
crossword puzzle, crossword
street sign
traffic light, traffic signal, stoplight
book jacket, dust cover, dust jacket, dust wrapper
menu
plate
guacamole
consomme
```

```
hot pot, hotpot
trifle
ice cream, icecream
ice lolly, lolly, lollipop, popsicle
French loaf
bagel, beigel
pretzel
cheeseburger
hotdog, hot dog, red hot
mashed potato
head cabbage
broccoli
cauliflower
zucchini, courgette
spaghetti squash
acorn squash
butternut squash
cucumber, cuke
artichoke, globe artichoke
bell pepper
cardoon
mushroom
Granny Smith
strawberry
orange
lemon
fig
pineapple, ananas
banana
jackfruit, jak, jack
custard apple
pomegranate
hay
carbonara
chocolate sauce, chocolate syrup
dough
meat loaf, meatloaf
pizza, pizza pie
potpie
burrito
red wine
espresso
cup
eggnog
alp
bubble
cliff, drop, drop-off
coral reef
geyser
lakeside, lakeshore
promontory, headland, head, foreland
sandbar, sand bar
seashore, coast, seacoast, sea-coast
valley, vale
volcano
ballplayer, baseball player
groom, bridegroom
scuba diver
rapeseed
daisy
yellow lady's slipper, yellow lady-slipper, Cypripedium calceolus, Cypripedium parviflorum
```

```
corn
acorn
hip, rose hip, rosehip
buckeye, horse chestnut, conker
coral fungus
agaric
gyromitra
stinkhorn, carrion fungus
earthstar
hen-of-the-woods, hen of the woods, Polyporus frondosus, Grifola frondosa
bolete
ear, spike, capitulum
toilet tissue, toilet paper, bathroom tissue
```

> **注意：**
> 如果下载测试图像，label 名称文件以及模型权重太慢，可以使用浏览器下载：
> - 在浏览器输入 https://hf-mirror.com/datasets/huggingface/cats-image/resolve/main/cats_image.jpeg，将下载好的图片放在data/cats_image.jpeg下。
> - 在浏览器输入 https://raw.githubusercontent.com/Lasagne/Recipes/master/examples/resnet50/imagenet_classes.txt，将下载好的 txt 文件放在data/imagenet_classes.txt下。
> - 在浏览器输入 https://download.pytorch.org/models/resnet50-0676ba61.pth，将下载好的模型权重放在root/.cache/torch/hub/checkpoints/resnet50-0676ba61.pth 下。

图像和 label 文件下载完成后，会先将图像进行 transform 以及变换为 batch 形式以适配网络输入；随后加载模型进行推理得到使用 Pytorch 的输出结果和推理时间；随后将模型转换为 TensorRT 形式并保存在 model/resnet50_trt.pth 下以便后续使用；最后使用 TensorRT 运行模型得到推理结果以及运行时间。输出结果如下（参考）：

```
download sample.
download label file.
Pytorch Predict class: tabby, tabby cat
Pytorch Processing time: 0.1641
Convert and save TensorRT model.
TensorRT Predict class: tabby, tabby cat
TensorRT Processing time: 0.0549
```

出现以上结果表明模型运行成功。可以看到无论是使用 Pytorch 还是 TensorRT 跑出的结果都是 tabby，tabby cat，但使用 TensorRT 推理时的运行时间要远小于单纯 Pytorch 运行。

注意事项

> **说明：**
> 由于 OpenCloudOS 是 TencentOS Server 的开源版本，理论上上述文档当中的所有操作同样适用于 OpenCloudOS。

参考文档

- Nvidia torch2trt image classification Demo
- Pytorch 离线下载并使用 torchvision.models 预训练模型
- 使用 PyTorch 中的预训练模型进行图像分类任务
- NVIDIA TensorRT GitHub
- Nvidia torch2trt GitHub
- Hugging Face cats-image 图像

Bert

最近更新时间：2024-08-13 09:55:21

本指导适用于在 TencentOS Server 3 上使用 TensorRT 推理框架运行 Bert 的官方 Demo，以 Docker 方式启动。

TensorRT 环境准备

1. 从 GitHub 下载 TensorRT 开源仓库到本地。

```
#下载TensorRT开源仓库到本地，版本为8.6.1
git clone git@github.com:NVIDIA/TensorRT.git -b release/8.6 --single-branch
cd TensorRT
```

2. 启动 TensorRT NGC 容器。

```
docker run -it --gpus all --name=tensorrt -e HF_ENDPOINT="https://hf-mirror.com" -v $PWD:/workspace
nvcr.io/nvidia/pytorch:23.06-py3 /bin/bash
```

此时会从 nvcr 拉取 docker 镜像，请确保网络环境较好，直到镜像里所有层下载成功。成功后会直接进入容器内部。

检查必要的包的版本

1. 确保使用较新的 pip 包和 TensorRT 包，并使用命令更新。

```
#更新pip包
python3 -m pip install --upgrade pip
```

2. 检查 TensorRT 版本，最低要求为8.6.0。

```
#检查TensorRT版本
python3 -c 'import tensorrt;print(tensorrt.__version__)'
```

本指导下载的 TensorRT 版本为8.6.1，如版本低于8.6.0，请卸载 TensorRT 并重新安装。

```
#卸载TensorRT包
python3 -m pip uninstall tensorrt

#下载新的TensorRT包，且指定版本，这里指定为8.6.1
python3 -m pip install tensorrt==8.6.1
python3 -c 'import tensorrt;print(tensorrt.__version__)'
```

下载运行框架和模型必要的包

将 pip 换为国内清华源以加快下载速度。

```
#将pip换成清华源
#设为默认，永久有效
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

随后安装运行模型必要的包：

```
#安装运行框架和模型所需的包
pip install pycuda onnx tensorflow torch
```

安装 NGC

由于 Demo 中下载模型需要从 Nvidia NGC 下载，我们首先需要安装 NGC。
按顺序执行以下命令安装：

```
cd /usr/local/bin
wget https://ngc.nvidia.com/downloads/ngccli_cat_linux.zip
unzip ngccli_cat_linux.zip
chmod u+x ngc-cli/ngc
echo "export PATH=\"\$PATH:$(pwd)/ngc-cli\"" >> ~/.bash_profile && source ~/.bash_profile
ngc config set
```

全部完成后会出现 `Enter API key [no-apikey]. Choices: [<VALID_APIKEY>, 'no-apikey']:` 等字样，回车两次出现：

```
...
Successfully validated configuration.
Saving configuration...
Successfully saved NGC configuration to /root/.ngc/config
```

则说明 NGC 安装成功。

> **注意：**
> 如果第二步 wget 下载速度太慢，可以单击此 链接 到浏览器中下载，下载完成后放入 /usr/local/bin 文件夹下即可。

运行模型

1. 首先下载 SQuAD v1.1 数据集：

```
export TRT_OSSPATH=/workspace
cd $TRT_OSSPATH/demo/BERT

bash ./scripts/download_squad.sh
```

2. 随后下载 Bert_large 模型，sequence 长度为128：

```
bash scripts/download_model.sh
```

> **说明：**
> 请保证网络状况较好，模型大约3.7G左右，如遇下载较慢，耐心等待即可。

模型下载完成后会出现如下字样（参考）：

```
Download status: COMPLETED
Downloaded local path model: /workspace/demo/BERT/models/fine-
tuned/bert_tf_ckpt_large_qa_squad2_amp_128_v19.03.1
Total files downloaded: 6
Total transferred: 3.75 GB
Started at: 2024-07-22 12:30:15
Completed at: 2024-07-22 12:53:34
Duration taken: 23m 18s
```

3. 创建 TensorRT engine：

```
mkdir -p engines
python3 builder.py -m models/fine-tuned/bert_tf_ckpt_large_qa_squad2_amp_128_v19.03.1/model.ckpt -o
engines/bert_large_128.engine -b 1 -s 128 --fp16 -c models/fine-
tuned/bert_tf_ckpt_large_qa_squad2_amp_128_v19.03.1
```

此 engine 最大 batchsize=1，sequence length=128，mixed precision=fp16，使用 BERT Large SQuAD v2 FP16 Sequence Length 128 checkpoint。

4. engine 创建好后在 engines 文件夹下可以看到 bert_large_128.engine 文件，窗口会出现以下字样（参考）：

```
...
[07/23/2024-07:21:16] [TRT] [W] TensorRT encountered issues when converting weights between types and
that could affect accuracy.
[07/23/2024-07:21:16] [TRT] [W] If this is not the desired behavior, please modify the weights or
retrain with regularization to adjust the magnitude of the weights.
[07/23/2024-07:21:16] [TRT] [W] Check verbose logs for the list of affected weights.
[07/23/2024-07:21:16] [TRT] [W] - 165 weights are affected by this issue: Detected subnormal FP16
values.
[07/23/2024-07:21:16] [TRT] [W] - 92 weights are affected by this issue: Detected values less than
smallest positive FP16 subnormal value and converted them to the FP16 minimum subnormalized value.
[07/23/2024-07:21:16] [TRT] [I] [MemUsageChange] TensorRT-managed allocation in building engine: CPU
+576, GPU +577, now: CPU 576, GPU 577 (MiB)
[07/23/2024-07:21:16] [TRT] [I] build engine in 34.906 Sec
[07/23/2024-07:21:17] [TRT] [I] Saving Engine to engines/bert_large_128.engine
[07/23/2024-07:21:17] [TRT] [I] Done.
```

5. 运行模型：

```
python3 inference.py -e engines/bert_large_128.engine -p "TensorRT is a high performance deep learning
inference platform that delivers low latency and high throughput for apps such as recommenders, speech
and image/video on NVIDIA GPUs. It includes parsers to import models, and plugins to support novel ops
and layers before applying optimizations for inference. Today NVIDIA is open-sourcing parsers and
plugins in TensorRT so that the deep learning community can customize and extend these components to
take advantage of powerful TensorRT optimizations for your apps." -q "What is TensorRT?" -v
models/fine-tuned/bert_tf_ckpt_large_qa_squad2_amp_128_v19.03.1/vocab.txt
```

这里我们给了一段话：

```
"TensorRT is a high performance deep learning inference platform that delivers low latency and high
throughput for apps such as recommenders, speech and image/video on NVIDIA GPUs. It includes parsers to
import models, and plugins to support novel ops and layers before applying optimizations for inference.
Today NVIDIA is open-sourcing parsers and plugins in TensorRT so that the deep learning community can
customize and extend these components to take advantage of powerful TensorRT optimizations for your
apps."
```

并给出问题：

```
What is TensorRT?
```

希望模型可以根据给的段落返回较好的问题的答案，运行脚本后返回结果如下（参考）：

```
...
Passage: TensorRT is a high performance deep learning inference platform that delivers low latency and
high throughput for apps such as recommenders, speech and image/video on NVIDIA GPUs. It includes
parsers to import models, and plugins to support novel ops and layers before applying optimizations for
inference. Today NVIDIA is open-sourcing parsers and plugins in TensorRT so that the deep learning
community can customize and extend these components to take advantage of powerful TensorRT
optimizations for your apps.

Question: What is TensorRT?
------------------------
Running inference in 16.378 Sentences/Sec
------------------------
```

```
Answer: 'a high performance deep learning inference platform'
With probability: 46.077
```

可以看到模型返回了输出结果，说明模型运行成功。

注意事项

> **说明：**
> 由于 OpenCloudOS 是 TencentOS Server 的开源版本，理论上上述文档当中的所有操作同样适用于 OpenCloudOS。

参考文档

- NVIDIA TensorRT Bert Demo
- NVIDIA NGC 安装指引
- NGC 安装报错指引
- NVIDIA TensorRT GitHub
- 清华 pipy 镜像源

vLLM
opt

最近更新时间：2024-08-13 09:55:21

本指导适用于在 TencentOS Server 3 上使用 vLLM 推理框架运行 opt 模型的官方 Demo，以 Docker 方式启动。

vLLM 环境准备

1. 从 GitHub 下载 vLLM 开源仓库到本地。

```
#下载vLLM开源仓库到本地
git clone https://github.com/vllm-project/vllm.git
cd vllm
```

2. 启动 vLLM 容器。

```
docker run --gpus all -it -e HF_ENDPOINT="https://hf-mirror.com" -v $PWD:/workspace --name=vllm --
ipc=host nvcr.io/nvidia/pytorch:23.10-py3 /bin/bash
```

此时会从 nvcr 拉取 docker 镜像，请确保网络环境较好，直到镜像里所有层下载成功。成功后会直接进入容器内部。

检查必要的包的版本

1. 确保使用较新的 pip 包，并使用命令更新。

```
#更新pip包
python3 -m pip install --upgrade pip
```

2. 此外，vLLM 需要确保 Python 版本在3.8以上才能正常工作，请检查 Python 版本。

```
python
```

本指导中安装的 Python 版本为3.10，如发现版本不在 3.8 - 3.11 之间，请重新安装 Python。

> ⚠ **注意：**
> vLLM 需要 GPU compute capability 7.0 或更高（例如 V100，T4，RTX20xx，A100，L4，H100等）。

安装 vLLM

可以使用以下两种方法在容器里安装 vLLM 框架：

```
#使用pip安装
pip install vllm
```

或者：

```
#使用vLLM GitHub仓库目录下的setup.py安装
pip install .
```

以上两种方式都可以安装 vLLM，请务必确保网络环境良好，否则容易安装失败。

> ⚠ **注意：**
> 如果尝试多次仍无法安装，请将 pip 换为国内清华源后再次尝试安装（此方法会极大的加快下载速度，强烈推荐！）。

```
#将pip换成清华源，以下方法二选一

#临时使用
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple some-package

#设为默认，永久有效（推荐）
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

vLLM 安装完成后，可以通过以下命令确认是否安装完成以及查看 vLLM 版本：

```
#确认vLLM是否安装完成以及查看安装的vLLM版本
python -c "import vllm; print(vllm.__version__)"
```

如果没有查找到 vLLM 库，请按照上述步骤重新安装。

运行模型

下载模型权重地址换源

1. 由于中国大陆无法下载 Hugging Face 网站模型，首先需要对下载网站换源，使用国内镜像网站的 HF-Mirror 模型。

   说明：
   如果 docker run 的时候加上了 `-e HF_ENDPOINT="https://hf-mirror.com"`，则此步可以跳过。

   ```
   #单次有效，退出容器且暂停容器运行后失效，再次进入容器需重新输入此条命令
   export HF_ENDPOINT="https://hf-mirror.com"

   #设为默认，永久有效，即便退出容器且暂停容器运行，再次进入容器后也可直接运行模型（推荐使用此方法）
   echo 'export HF_ENDPOINT="https://hf-mirror.com"' >> ~/.bashrc
   ```

2. 运行 opt-125m 的官方 Demo 位于 examples/offline_inference.py，运行该 Python 文件，则会自动开始下载模型并开始推理。

   ```
   #运行Demo
   python examples/offline_inference.py
   ```

   offline_inference.py 代码如下：

   ```
   from vllm import LLM, SamplingParams


   # Sample prompts.
   prompts = [
       "Hello, my name is",
       "The president of the United States is",
       "The capital of France is",
       "The future of AI is",
   ]
   # Create a sampling params object.
   sampling_params = SamplingParams(temperature=0.8, top_p=0.95)


   # Create an LLM.
   llm = LLM(model="facebook/opt-125m")
   # Generate texts from the prompts. The output is a list of RequestOutput objects
   # that contain the prompt, generated text, and other information.
   outputs = llm.generate(prompts, sampling_params)
   ```

```
# Print the outputs.
for output in outputs:
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print(f"Prompt: {prompt!r}, Generated text: {generated_text!r}")
```

该 Demo 会输入4条 prompts，通过 opt-125m 模型在 vLLM 推理框架下进行推理，最后给出生成的文字的结果。
生成的结果如下（参考）：

```
Prompt: 'Hello, my name is', Generated text: ' Joel, my dad is my friend and we are in a relationship.
I am'
Prompt: 'The president of the United States is', Generated text: ' speaking out against the release of
some State Department documents which show the Russians were involved'
Prompt: 'The capital of France is', Generated text: ' known as the “Proud French capital”. What is this
city'
Prompt: 'The future of AI is', Generated text: ' literally in danger of being taken by any other
company.\nAgreed. '
```

注意事项

> **说明：**
> 由于 OpenCloudOS 是 TencentOS Server 的开源版本，理论上上述文档当中的所有操作同样适用于 OpenCloudOS。

参考文档

- vllm GitHub
- vllm 安装指引
- 清华 pipy 镜像源
- Hugging Face 镜像源
- Hugging Face opt-125模型

LLaMA

最近更新时间：2024-08-13 09:55:21

本指导适用于在 TencentOS Server 3 上使用 vLLM 推理框架运行 LLaMA 模型，以 Docker 方式启动。

前置环境条件

请确保已经按照 opt 文档内进行操作，**运行模型**之前的所有步骤已经完成，并已经准备好了 vLLM 的所有必要环境。

运行模型

下载模型权重地址换源

1. 由于中国大陆无法下载 Hugging Face 网站模型，首先需要对下载网站换源，使用国内镜像网站的 HF-Mirror 模型。

> **说明：**
> 如果 docker run 的时候加上了 `-e HF_ENDPOINT="https://hf-mirror.com"`，则此步可以跳过。

```
#单次有效，退出容器且暂停容器运行后失效，再次进入容器需重新输入此条命令
export HF_ENDPOINT="https://hf-mirror.com"

#设为默认，永久有效，即便退出容器且暂停容器运行，再次进入容器后也可直接运行模型（推荐使用此方法）
echo 'export HF_ENDPOINT="https://hf-mirror.com"' >> ~/.bashrc
```

官方 Demo 位于 examples/offline_inference.py、offline_inference.py 代码如下：

```
from vllm import LLM, SamplingParams


# Sample prompts.
prompts = [
    "Hello, my name is",
    "The president of the United States is",
    "The capital of France is",
    "The future of AI is",
]
# Create a sampling params object.
sampling_params = SamplingParams(temperature=0.8, top_p=0.95)


# Create an LLM.
llm = LLM(model="facebook/opt-125m")
# Generate texts from the prompts. The output is a list of RequestOutput objects
# that contain the prompt, generated text, and other information.
outputs = llm.generate(prompts, sampling_params)
# Print the outputs.
for output in outputs:
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print(f"Prompt: {prompt!r}, Generated text: {generated_text!r}")
```

2. 将导入模型的代码进行更改并保存：

```
#原代码
llm = LLM(model="facebook/opt-125m")

#更改为：
```

```
llm = LLM(model="openlm-research/open_llama_7b")
```

3. 运行该 Python 文件，则会自动开始下载 LLaMA_7b 模型并开始推理。

```
#运行Demo
python examples/offline_inference.py
```

该 Demo 会输入4条 prompts，通过 LLaMA_7b 模型在 vLLM 推理框架下进行推理，最后给出生成的文字的结果。
生成的结果如下（参考）：

```
Prompt: 'Hello, my name is', Generated text: ' Dario and I work for the National Geographic Society. I
have been a National'
Prompt: 'The president of the United States is', Generated text: ' elected to serve as the head of the
executive branch of the federal government. The'
Prompt: 'The capital of France is', Generated text: ' Paris. It is situated on the river Seine. It is
the 4'
Prompt: 'The future of AI is', Generated text: ' now: how it is changing the recruitment
industry\nAuthor: Jonny Moran'
```

注意事项

> **说明：**
> 由于 OpenCloudOS 是 TencentOS Server 的开源版本，理论上上述文档当中的所有操作同样适用于 OpenCloudOS。

参考文档

- vllm GitHub
- vllm 安装指引
- Hugging Face 镜像网站
- Hugging Face open_llama_7b 模型

TensorRT-LLM Baichuan

最近更新时间：2024-08-13 09:55:21

本指导适用于在 TencentOS Server 3 上使用 TensorRT-LLM 推理框架运行 Baichuan 模型的官方 Demo，以 Docker 方式启动。

TensorRT-LLM 环境准备

1. 从 GitHub 下载 TensorRT-LLM 开源仓库到本地，由于 TensorRT-LLM 目前更新迭代频繁，为了后续版本的稳定运行，这里使用较稳定版本 v0.10.0 运行模型。

```
#下载TensorRT-LLM开源仓库到本地，版本号为v0.10.0
git clone -b v0.10.0 https://github.com/NVIDIA/TensorRT-LLM.git
cd TensorRT-LLM
```

2. 启动 TensorRT-LLM 容器。

```
docker run -it --name=tensorrtllm -e HF_ENDPOINT="https://hf-mirror.com" -v $PWD:/workspace --
runtime=nvidia --entrypoint /bin/bash --gpus all nvcr.io/nvidia/cuda:12.4.0-devel-ubuntu22.04
```

此时会从 nvcr 拉取 docker 镜像，请确保网络环境较好，直到镜像里所有层下载成功。成功后会直接进入容器内部。

> **注意：**
> 请勿按照官方文档的指引从 **nvidia/cuda:12.4.0-devel-ubuntu22.04** 拉取镜像。相反，应从 **nvcr.io/nvidia/cuda:12.4.0-devel-ubuntu22.04** 拉取镜像，因为从前者拉取镜像可能会因无法访问等原因失败。
> 参见网址：nvidia cuda 镜像 。

安装 TensorRT-LLM 以及必要的包

1. 安装运行 TensorRT-LLM 必要的包，通过 apt-get 下载。

```
#安装所有需要的包
apt-get update && apt-get -y install python3.10 python3-pip openmpi-bin libopenmpi-dev git git-lfs
```

> **说明：**
> TencentOS 3.1 基于 CentOS，但由于镜像环境为 Ubuntu 22.04，因此能够使用 apt-get 下载包。这并不矛盾，所以可以放心安装。

2. 将 pip 换为国内清华源以加快下载速度（此步骤可以不做，但强烈推荐，否则后续可能下载速度极慢）。

```
#将pip换成清华源
#设为默认，永久有效
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

安装 TensorRT-LLM，注意安装版本为 v0.10.0。

```
pip3 install tensorrt_llm==0.10.0 -U --extra-index-url https://pypi.nvidia.com
```

> **注意：**
> 通过执行此命令，您将安装 TensorRT-LLM 的较稳定版本 v0.10.0，但请注意，这可能不是最新版本！如果您希望安装 TensorRT-LLM 的最新版本，请添加参数 **--pre** 并且不要指定版本号。

3. 检查 TensorRT-LLM 是否安装成功：

```
python3 -c "import tensorrt_llm"
```

未出现包报错且出现 TensorRT-LLM 版本号则说明安装成功，否则需重新安装，本指导此时应正确显示安装的 TensorRT-LLM 的版本 v0.10.0。

> **警告：**
> 使用命令 `python3 -c "import tensorrt_llm"` 检查 TensorRT-LLM 是否安装成功时，**请勿在 /workspace 路径下验证**，因为 /workspace 里有tensorrt_llm 文件夹，`"import tensorrt_llm"` 时会错误的去 /workspace/tensorrt_llm 里导入包，从而导致 `ModuleNotFoundError: No module named 'tensorrt_llm.bindings'` 的报错。

运行模型

下载模型权重地址换源

1. 由于中国大陆无法下载 Hugging Face 网站模型，首先需要对下载网站换源，使用国内镜像网站的 HF-Mirror 模型。

> **说明：**
> 如果 docker run 的时候加上了 `-e HF_ENDPOINT="https://hf-mirror.com"`，则此步可以跳过。

```
#单次有效，退出容器且暂停容器运行后失效，再次重启进入容器需重新输入此条命令
export HF_ENDPOINT="https://hf-mirror.com"
```

> **注意：**
> 这里使用 `echo 'export HF_ENDPOINT="https://hf-mirror.com"' >> ~/.bashrc` 的命令仍然会导致下载失败，请勿使用。

安装运行 Baichuan 必要的环境

安装 TensorRT-LLM 时不是所有环境包都已安装好，运行模型时有些模型专属的包仍需单独安装。

```
cd workspace/examples/baichuan
pip install -r requirements.txt
```

下载模型并构建 TensorRT-LLM engine(s)

本指导运行 Baichuan-V1-13B-Chat 模型，使用单 GPU 推理以及 FP16。

1. 运行 convert_checkpoint.py 文件将下载的模型权重从 HuggingFace (HF) Transformers 格式变为 TensorRT-LLM 格式。

```
# Convert the Baichuan V1 13B model using a single GPU and FP16.
python3 convert_checkpoint.py --model_version v1_13b \
                              --model_dir baichuan-inc/Baichuan-13B-Chat \
                              --dtype float16 \
                              --output_dir ./tmp/baichuan_v1_13b/trt_ckpts/fp16/1-gpu/
```

运行后会开始下载模型并转换格式。

> **说明：**
> - 对于 Baichuan-V1-13B，可设置 `--model_dir` 参数为 `baichuan-inc/Baichuan-13B-Chat` 或者 `baichuan-inc/Baichuan-13B-Base`。
> - 对于Baichuan-V2-13B，可设置 `--model_dir` 参数为 `baichuan-inc/Baichuan2-13B-Chat` 或者 `baichuan-inc/Baichuan2-13B-Base` 。
>
> 除了13B 模型，还可以设置模型参数量 `--model_version` 为 `v1_7b` / `v1_13b` / `v2_7b` / `v2_13b`，注意：`--model_dir` 参数也要同时变换。
> 以下是更多运行示例供参考运行：
>
> ```
> # Convert the Baichuan V1 13B model using a single GPU and BF16.
> ```

```
python3 convert_checkpoint.py --model_version v1_13b \
                              --model_dir baichuan-inc/Baichuan-13B-Chat \
                              --dtype bfloat16 \
                              --output_dir ./tmp/baichuan_v1_13b/trt_ckpts/bf16/1-gpu/
```

```
# Convert the Baichuan V1 13B model using a single GPU and apply INT8 weight-only quantization.
python3 convert_checkpoint.py --model_version v1_13b \
                              --model_dir baichuan-inc/Baichuan-13B-Chat \
                              --dtype float16 \
                              --use_weight_only \
                              --output_dir ./tmp/baichuan_v1_13b/trt_ckpts/int8_weight_only/1-
gpu/
```

```
# Convert the Baichuan V1 13B model using a single GPU and apply INT4 weight-only quantization.#
Note that Baichuan V1 7B performs not well when using INT4 weight-only quantization.
python3 convert_checkpoint.py --model_version v1_13b \
                              --model_dir baichuan-inc/Baichuan-13B-Chat \
                              --dtype float16 \
                              --use_weight_only \
                              --weight_only_precision int4 \
                              --output_dir ./tmp/baichuan_v1_13b/trt_ckpts/int4_weight_only/1-
gpu/
```

```
# Convert Baichuan V1 13B using 2-way tensor parallelism.
python3 convert_checkpoint.py --model_version v1_13b \
                              --model_dir baichuan-inc/Baichuan-13B-Chat \
                              --dtype float16 \
                              --output_dir ./tmp/baichuan_v1_13b/trt_ckpts/fp16/1-gpu/ \
                              --world_size 2 \
                              --tp_size 2
```

更多运行示例请参见 NVIDIA TensorRT-LLM baichuan Demo 。

2. 下载完模型后，构建 TensorRT-LLM engine：

```
trtllm-build --checkpoint_dir ./tmp/baichuan_v1_13b/trt_ckpts/fp16/1-gpu/ \
             --output_dir ./tmp/baichuan_v1_13b/trt_engines/fp16/1-gpu/ \
             --gemm_plugin float16 \
             --max_batch_size=32 \
             --max_input_len=1024 \
             --max_output_len=512
```

构建完成后会看到以下结果（参考）：

```
...
[07/10/2024-11:07:27] [TRT] [I] Engine generation completed in 15.62 seconds.
[07/10/2024-11:07:27] [TRT] [I] [MemUsageStats] Peak memory usage of TRT CPU/GPU memory allocators: CPU
1250 MiB, GPU 25301 MiB
[07/10/2024-11:07:30] [TRT] [I] [MemUsageStats] Peak memory usage during Engine building and
serialization: CPU: 55097 MiB
[07/10/2024-11:07:30] [TRT-LLM] [I] Total time of building Unnamed Network 0: 00:00:19
[07/10/2024-11:07:30] [TRT] [I] Serialized 26 bytes of code generator cache.
[07/10/2024-11:07:30] [TRT] [I] Serialized 204697 bytes of compilation cache.
[07/10/2024-11:07:30] [TRT] [I] Serialized 16 timing cache entries
[07/10/2024-11:07:30] [TRT-LLM] [I] Timing cache serialized to model.cache
```

```
[07/10/2024-11:07:30] [TRT-LLM] [I] Serializing engine to ./tmp/baichuan_v1_13b/trt_engines/fp16/1-
gpu/rank0.engine...
[07/10/2024-11:07:39] [TRT-LLM] [I] Engine serialized. Total time: 00:00:08
[07/10/2024-11:07:39] [TRT-LLM] [I] Total time of building all engines: 00:00:29
```

运行模型

文本生成任务（run.py）

输入给模型 prompt，生成文字并返回。

```
# With fp16 inference
python ../run.py --input_text "世界上第二高的山峰是哪座？" \
                 --max_output_len=200 \
                 --tokenizer_dir baichuan-inc/Baichuan-13B-Chat \
                 --engine_dir=./tmp/baichuan_v1_13b/trt_engines/fp16/1-gpu/
```

运行结束后生成答案（参考）：

```
...
[TensorRT-LLM][INFO] MPI size: 1, rank: 0
[TensorRT-LLM][INFO] Rank 0 is using GPU 0
[TensorRT-LLM][INFO] TRTGptModel maxNumSequences: 32
[TensorRT-LLM][INFO] TRTGptModel maxBatchSize: 32
[TensorRT-LLM][INFO] TRTGptModel mMaxAttentionWindowSize: 1536
[TensorRT-LLM][INFO] TRTGptModel enableTrtOverlap: 0
[TensorRT-LLM][INFO] TRTGptModel normalizeLogProbs: 1
[TensorRT-LLM][INFO] Loaded engine size: 25305 MiB
[TensorRT-LLM][INFO] Allocated 5776.00 MiB for execution context memory.
[TensorRT-LLM][INFO] [MemUsageChange] TensorRT-managed allocation in IExecutionContext creation: CPU +0,
GPU +0, now: CPU 0, GPU 25300 (MiB)
[TensorRT-LLM][INFO] Max KV cache pages per sequence: 24
[TensorRT-LLM][INFO] Max tokens in paged KV cache: 15936. Allocating 13054771200 bytes.
Input [Text 0]: "世界上第二高的山峰是哪座？"
Output [Text 0 Beam 0]: "

世界上第二高的山峰是喀喇昆仑山脉的乔戈里峰(K2)，海拔8,611米(28,251英尺)。它是唯一一座海拔超过8,500米的14座海拔8,000米以上
山峰之一。乔戈里峰位于巴基斯坦和中国新疆的边界上，是登山者和探险家们梦寐以求的目标。"
```

文本总结任务（summarize.py）

根据已有的文字进行总结并返回，这里的示例为总结 cnn_dailymail 数据集里的文章。

```
# Run summarization using the Baichuan V1 13B model in FP16.
python3 ../summarize.py --test_trt_llm \
                        --hf_model_dir baichuan-inc/Baichuan-13B-Chat \
                        --data_type fp16 \
                        --engine_dir ./tmp/baichuan_v1_13b/trt_engines/fp16/1-gpu/
```

运行后会开始下载 cnn_dailymail 数据集，运行结束后返回结果。
以下为输入一篇文章以及引用后，模型输出的总结 Output（参考）：

```
...
[07/10/2024-12:45:20] [TRT-LLM] [I] TensorRT-LLM Generated :
[07/10/2024-12:45:20] [TRT-LLM] [I]  Input : ['(CNN)James Best, best known for his portrayal of bumbling
sheriff Rosco P. Coltrane on TV\'s "The Dukes of Hazzard," died Monday after a brief illness. He was 88.
Best died in hospice in Hickory, North Carolina, of complications from pneumonia, said Steve Latshaw, a
longtime friend and Hollywood colleague. Although he\'d been a busy actor for decades in theater and in
Hollywood, Best didn\'t become famous until 1979, when "The Dukes of Hazzard\'s" cornpone charms began
```

```
beaming into millions of American homes almost every Friday night. For seven seasons, Best\'s Rosco P. Coltrane chased the moonshine-running Duke boys back and forth across the back roads of fictitious Hazzard County, Georgia, although his "hot pursuit" usually ended with him crashing his patrol car. Although Rosco was slow-witted and corrupt, Best gave him a childlike enthusiasm that got laughs and made him endearing. His character became known for his distinctive "kew-kew-kew" chuckle and for goofy catchphrases such as "cuff \'em and stuff \'em!" upon making an arrest. Among the most popular shows on TV in the early \'80s, "The Dukes of Hazzard" ran until 1985 and spawned TV movies, an animated series and video games. Several of Best\'s "Hazzard" co-stars paid tribute to the late actor on social media. "I laughed and learned more from Jimmie in one hour than from anyone else in a whole year," co-star John Schneider, who played Bo Duke, said on Twitter. "Give Uncle Jesse my love when you see him dear friend." "Jimmy Best was the most constantly creative person I have ever known," said Ben Jones, who played mechanic Cooter on the show, in a Facebook post. "Every minute of his long life was spent acting, writing, producing, painting, teaching, fishing, or involved in another of his life\'s many passions." Born Jewel Guy on July 26, 1926, in Powderly, Kentucky, Best was orphaned at 3 and adopted by Armen and Essa Best, who renamed him James and raised him in rural Indiana. Best served in the Army during World War II before launching his acting career. In the 1950s and 1960s, he accumulated scores of credits, playing a range of colorful supporting characters in such TV shows as "The Twilight Zone," "Bonanza," "The Andy Griffith Show" and "Gunsmoke." He later appeared in a handful of Burt Reynolds\' movies, including "Hooper" and "The End." But Best will always be best known for his "Hazzard" role, which lives on in reruns. "Jimmie was my teacher, mentor, close friend and collaborator for 26 years," Latshaw said. "I directed two of his feature films, including the recent \'Return of the Killer Shrews,\' a sequel he co-wrote and was quite proud of as he had made the first one more than 50 years earlier." People we\'ve lost in 2015 . CNN\'s Stella Chan contributed to this story.']
[07/10/2024-12:45:20] [TRT-LLM] [I]
 Reference : ['James Best, who played the sheriff on "The Dukes of Hazzard," died Monday at 88
.\n"Hazzard" ran from 1979 to 1985 and was among the most popular shows on TV .']
[07/10/2024-12:45:20] [TRT-LLM] [I]
 Output : [['James Best, best known for his portrayal of bumbling sheriff Rosco P. Coltrane on TV\'s "The Dukes of Hazzard," died Monday after a brief illness. He was 88. Best died in hospice in Hickory, North Carolina, of complications from pneumonia, said Steve Latshaw, a longtime friend and Hollywood colleague. He was 88.']]
[07/10/2024-12:45:20] [TRT-LLM] [I] ------------------------------------------------------------
[07/10/2024-12:46:24] [TRT-LLM] [I] TensorRT-LLM (total latency: 64.10118317604065 sec)
[07/10/2024-12:46:24] [TRT-LLM] [I] TensorRT-LLM (total output tokens: 1617)
[07/10/2024-12:46:24] [TRT-LLM] [I] TensorRT-LLM (tokens per second: 25.22574342441143)
[07/10/2024-12:46:24] [TRT-LLM] [I] TensorRT-LLM beam 0 result
[07/10/2024-12:46:24] [TRT-LLM] [I]   rouge1 : 24.802194907901427
[07/10/2024-12:46:24] [TRT-LLM] [I]   rouge2 : 7.547955247617474
[07/10/2024-12:46:24] [TRT-LLM] [I]   rougeL : 18.5436038275683
[07/10/2024-12:46:24] [TRT-LLM] [I]   rougeLsum : 20.69098743372036
```

说明：

更多运行示例供参考：

```
# Run summarization using the Baichuan V1 13B model quantized to INT8.
python ../summarize.py --test_trt_llm \
                       --hf_model_dir baichuan-inc/Baichuan-13B-Chat \
                       --data_type fp16 \
                       --engine_dir ./tmp/baichuan_v1_13b/trt_engines/int8_weight_only/1-gpu/
```

```
# Run summarization using the Baichuan V1 13B model in FP16 using two GPUs.
mpirun -n 2 --allow-run-as-root \
    python ../summarize.py --test_trt_llm \
                           --hf_model_dir baichuan-inc/Baichuan-13B-Chat \
                           --data_type fp16 \
                           --engine_dir ./tmp/baichuan_v1_13b/trt_engines/fp16/2-gpu/
```

注意事项

> **说明：**
> 由于 OpenCloudOS 是 TencentOS Server 的开源版本，理论上上述文档当中的所有操作同样适用于 OpenCloudOS。

参考文档

- TensorRT-LLM 安装指引
- NVIDIA TensorRT-LLM GitHub
- TensorRT-LLM v0.10.0版本 log 日志
- NVIDIA TensorRT-LLM baichuan Demo
- nvidia cuda 镜像
- 清华 pipy 镜像源
- Hugging Face 镜像网站
- cnn_dailymail 数据集
- Hugging Face baichuan-inc/Baichuan-13B-Chat 模型

ChatGLM

最近更新时间：2024-08-13 09:55:21

本指导适用于在 TencentOS Server 3 上使用 TensorRT-LLM 推理框架运行 ChatGLM 模型的官方 Demo，以 Docker 方式启动。

前置环境条件

请确保已经按照 Baichuan 文档内进行操作，运行模型之前的所有步骤已经完成，并已经准备好了 TensorRT-LLM 的所有必要环境。

运行模型

下载模型权重地址换源

1. 由于中国大陆无法下载 Hugging Face 网站模型，首先需要对下载网站换源，使用国内镜像网站的 HF-Mirror 模型。

> **说明：**
> 如果 docker run 的时候加上了 `-e HF_ENDPOINT="https://hf-mirror.com"`，则此步可以跳过。

```
#单次有效，退出容器且暂停容器运行后失效，再次重启进入容器需重新输入此条命令
export HF_ENDPOINT="https://hf-mirror.com"
```

> **注意：**
> 这里使用 `echo 'export HF_ENDPOINT="https://hf-mirror.com"' >> ~/.bashrc` 的命令仍然会导致下载失败，请勿使用。

安装运行 ChatGLM 必要的环境

安装 TensorRT-LLM 时不是所有环境包都已安装好，运行模型时有些模型专属的包仍需单独安装。

```
cd workspace/examples/chatglm
pip install -r requirements.txt
```

下载模型并构建 TensorRT-LLM engine(s)

本指导运行 ChatGLM-V3-6B 模型，使用单 GPU 推理以及 FP16。

1. 运行 convert_checkpoint.py 文件将下载的模型权重从 HuggingFace (HF) Transformers 格式变为 TensorRT-LLM 格式。

```
# ChatGLM3-6B: single gpu, dtype float16
python3 convert_checkpoint.py --model_dir THUDM/chatglm3-6b --output_dir trt_ckpt/chatglm3_6b/fp16/1-
gpu
```

运行后会开始下载模型并转换格式。

> **说明：**
> 除了 ChatGLM3，还有 ChatGLM2，ChatGLM，GLM 等模型可以运行。
> 以下是更多运行示例供参考运行：

```
# ChatGLM3-6B: 2-way tensor parallelism
python3 convert_checkpoint.py --model_dir THUDM/chatglm3-6b --tp_size 2 --output_dir
trt_ckpt/chatglm3_6b/fp16/2-gpu
```

```
# Chatglm2-6B: single gpu, dtype float16
python3 convert_checkpoint.py --model_dir THUDM/chatglm2-6b --output_dir
trt_ckpt/chatglm2_6b/fp16/1-gpu
```

```
# Chatglm-6B: single gpu, dtype float16
python3 convert_checkpoint.py --model_dir THUDM/chatglm-6b --output_dir
trt_ckpt/chatglm_6b/fp16/1-gpu
```

```
# GLM-10B: single gpu, dtype float16
python3 convert_checkpoint.py --model_dir THUDM/glm-10b --output_dir trt_ckpt/glm_10b/fp16/1-gpu
```

更多运行示例请参见 NVIDIA TensorRT-LLM ChatGLM Demo 。

注意：

官方文档中的 `--model_dir` 格式与本指导不同，这是因为官方文档通过从 GitHub 仓库拉取并重新命名模型来下载模型。而在这里，我们直接通过 Hugging Face 下载模型，因此需要 `--model_dir` 的格式与模型在 Hugging Face 网站上的标准命名保持一致。

2. 下载完模型后，构建 TensorRT-LLM engine：

```
# ChatGLM3-6B: single-gpu engine
trtllm-build --checkpoint_dir trt_ckpt/chatglm3_6b/fp16/1-gpu \
             --gemm_plugin float16 \
             --output_dir trt_engines/chatglm3_6b/fp16/1-gpu
```

说明：

更多运行示例构建 TensorRT-LLM engine：

```
# ChatGLM3-6B: 2-way tensor parallelism
trtllm-build --checkpoint_dir trt_ckpt/chatglm3_6b/fp16/2-gpu \
             --gemm_plugin float16 \
             --output_dir trt_engines/chatglm3_6b/fp16/2-gpu
```

```
# ChatGLM2-6B: single-gpu engine with dtype float16, GPT Attention plugin, Gemm plugin
trtllm-build --checkpoint_dir trt_ckpt/chatglm2_6b/fp16/1-gpu \
             --gemm_plugin float16 \
             --output_dir trt_engines/chatglm2_6b/fp16/1-gpu
```

```
# ChatGLM-6B: single-gpu engine with dtype float16, GPT Attention plugin, Gemm plugin
trtllm-build --checkpoint_dir trt_ckpt/chatglm_6b/fp16/1-gpu \
             --gemm_plugin float16 \
             --output_dir trt_engines/chatglm_6b/fp16/1-gpu
```

```
# GLM-10B: single-gpu engine with dtype float16, GPT Attention plugin, Gemm plugin
trtllm-build --checkpoint_dir trt_ckpt/glm_10b/fp16/1-gpu \
             --gemm_plugin float16 \
             --output_dir trt_engines/glm_10b/fp16/1-gpu
```

构建完成后会看到以下结果（参考）：

```
...
[07/11/2024-03:30:54] [TRT] [I] Engine generation completed in 10.4344 seconds.
[07/11/2024-03:30:54] [TRT] [I] [MemUsageStats] Peak memory usage of TRT CPU/GPU memory allocators: CPU
1016 MiB, GPU 11911 MiB
```

```
[07/11/2024-03:30:55] [TRT] [I] [MemUsageStats] Peak memory usage during Engine building and
serialization: CPU: 28426 MiB
[07/11/2024-03:30:55] [TRT-LLM] [I] Total time of building Unnamed Network 0: 00:00:12
[07/11/2024-03:30:55] [TRT] [I] Serialized 26 bytes of code generator cache.
[07/11/2024-03:30:55] [TRT] [I] Serialized 148708 bytes of compilation cache.
[07/11/2024-03:30:55] [TRT] [I] Serialized 27 timing cache entries
[07/11/2024-03:30:55] [TRT-LLM] [I] Timing cache serialized to model.cache
[07/11/2024-03:30:55] [TRT-LLM] [I] Serializing engine to trt_engines/chatglm3_6b/fp16/1-
gpu/rank0.engine...
[07/11/2024-03:31:00] [TRT-LLM] [I] Engine serialized. Total time: 00:00:04
[07/11/2024-03:31:00] [TRT-LLM] [I] Total time of building all engines: 00:00:18
```

运行模型

文本生成任务（run.py）

输入给模型 prompt，生成文字并返回。

```
# Run the default engine of ChatGLM3-6B on single GPU, other model name is available if built.
python3 ../run.py --input_text "What's new between ChatGLM3-6B and ChatGLM2-6B?" \
                  --max_output_len 50 \
                  --tokenizer_dir THUDM/chatglm3-6b \
                  --engine_dir trt_engines/chatglm3_6b/fp16/1-gpu
```

运行结束后生成答案 Output（参考）：

```
...
[TensorRT-LLM][INFO] MPI size: 1, rank: 0
[TensorRT-LLM][INFO] Rank 0 is using GPU 0
[TensorRT-LLM][INFO] TRTGptModel maxNumSequences: 1
[TensorRT-LLM][INFO] TRTGptModel maxBatchSize: 1
[TensorRT-LLM][INFO] TRTGptModel mMaxAttentionWindowSize: 2048
[TensorRT-LLM][INFO] TRTGptModel enableTrtOverlap: 0
[TensorRT-LLM][INFO] TRTGptModel normalizeLogProbs: 1
[TensorRT-LLM][INFO] Loaded engine size: 11913 MiB
[TensorRT-LLM][INFO] Allocated 118.50 MiB for execution context memory.
[TensorRT-LLM][INFO] [MemUsageChange] TensorRT-managed allocation in IExecutionContext creation: CPU +0,
GPU +0, now: CPU 0, GPU 11910 (MiB)
[TensorRT-LLM][INFO] Max KV cache pages per sequence: 32
[TensorRT-LLM][INFO] Max tokens in paged KV cache: 1082496. Allocating 31037325312 bytes.
Input [Text 0]: "[gMASK] sop What's new between ChatGLM3-6B and ChatGLM2-6B?"
Output [Text 0 Beam 0]: "There is no new information about ChatGLM3-6B, but I heard that ChatGLM2-6B has
been updated. Can you tell me more about the updates to ChatGLM2-6B?"
```

说明：

以下是更多运行示例供参考运行：

```
# Run the default engine of ChatGLM3-6B on single GPU, using streaming output, other model name is
available if built.
python3 ../run.py --input_text "What's new between ChatGLM3-6B and ChatGLM2-6B?" \
                  --max_output_len 50 \
                  --tokenizer_dir THUDM/chatglm3-6b \
                  --engine_dir trt_engines/chatglm3_6b/fp16/1-gpu \
                  --streaming
```

GLM 模型还可以进行完形填空（生成 [MASK] 部分的字词）：

```
# Run the default engine of GLM3-10B on single GPU, other model name is available if built.
```

```
# Token "[MASK]" or "[sMASK]" or "[gMASK]" must be included in the prompt as the original model commanded.
python3 ../run.py --input_text "Peking University is [MASK] than Tsinghua University." \
                  --max_output_len 50 \
                  --tokenizer_dir THUDM/glm-10b \
                  --engine_dir trt_engines/glm_10b/fp16/1-gpu
```

以上示例仍需注意 `--tokenizer_dir` 参数的格式，需要与 Hugging Face 网站模型的标准命名格式一致。

文本总结任务（summarize.py）

根据已有的文字进行总结并返回，这里的示例为总结 cnn_dailymail 数据集里的文章。

```
# Run the summarization of ChatGLM3-6B task, other model name is available if built.
python3 ../summarize.py --test_trt_llm \
                        --hf_model_dir THUDM/chatglm3-6b \
                        --engine_dir trt_engines/chatglm3_6b/fp16/1-gpu
```

运行后会开始下载 cnn_dailymail 数据集，运行结束后返回结果。
以下为输入一篇文章以及引用后，模型输出的总结 Output（参考）：

```
...
[07/11/2024-03:57:49] [TRT-LLM] [I] TensorRT-LLM Generated :
[07/11/2024-03:57:49] [TRT-LLM] [I]  Input : ['(CNN)James Best, best known for his portrayal of bumbling sheriff Rosco P. Coltrane on TV\'s "The Dukes of Hazzard," died Monday after a brief illness. He was 88. Best died in hospice in Hickory, North Carolina, of complications from pneumonia, said Steve Latshaw, a longtime friend and Hollywood colleague. Although he\'d been a busy actor for decades in theater and in Hollywood, Best didn\'t become famous until 1979, when "The Dukes of Hazzard\'s" cornpone charms began beaming into millions of American homes almost every Friday night. For seven seasons, Best\'s Rosco P. Coltrane chased the moonshine-running Duke boys back and forth across the back roads of fictitious Hazzard County, Georgia, although his "hot pursuit" usually ended with him crashing his patrol car. Although Rosco was slow-witted and corrupt, Best gave him a childlike enthusiasm that got laughs and made him endearing. His character became known for his distinctive "kew-kew-kew" chuckle and for goofy catchphrases such as "cuff \'em and stuff \'em!" upon making an arrest. Among the most popular shows on TV in the early \'80s, "The Dukes of Hazzard" ran until 1985 and spawned TV movies, an animated series and video games. Several of Best\'s "Hazzard" co-stars paid tribute to the late actor on social media. "I laughed and learned more from Jimmie in one hour than from anyone else in a whole year," co-star John Schneider, who played Bo Duke, said on Twitter. "Give Uncle Jesse my love when you see him dear friend." "Jimmy Best was the most constantly creative person I have ever known," said Ben Jones, who played mechanic Cooter on the show, in a Facebook post. "Every minute of his long life was spent acting, writing, producing, painting, teaching, fishing, or involved in another of his life\'s many passions." Born Jewel Guy on July 26, 1926, in Powderly, Kentucky, Best was orphaned at 3 and adopted by Armen and Essa Best, who renamed him James and raised him in rural Indiana. Best served in the Army during World War II before launching his acting career. In the 1950s and 1960s, he accumulated scores of credits, playing a range of colorful supporting characters in such TV shows as "The Twilight Zone," "Bonanza," "The Andy Griffith Show" and "Gunsmoke." He later appeared in a handful of Burt Reynolds\' movies, including "Hooper" and "The End." But Best will always be best known for his "Hazzard" role, which lives on in reruns. "Jimmie was my teacher, mentor, close friend and collaborator for 26 years," Latshaw said. "I directed two of his feature films, including the recent \'Return of the Killer Shrews,\' a sequel he co-wrote and was quite proud of as he had made the first one more than 50 years earlier." People we\'ve lost in 2015 . CNN\'s Stella Chan contributed to this story.']
[07/11/2024-03:57:49] [TRT-LLM] [I]
 Reference : ['James Best, who played the sheriff on "The Dukes of Hazzard," died Monday at 88 .\n"Hazzard" ran from 1979 to 1985 and was among the most popular shows on TV .']
[07/11/2024-03:57:49] [TRT-LLM] [I]
 Output : [['James Best, the actor best known for his portrayal of bumbling sheriff Rosco P. Coltrane on "The Dukes of Hazzard," has died at 88 after a brief illness. He was a busy actor for decades in theater and in Hollywood, but became famous in 1979 when the show began running every week. Best\'s Rosco P. Coltrane became known for his distinctive "kew-kew-kew" chuck']]
```

```
[07/11/2024-03:57:49] [TRT-LLM] [I] ----------------------------------------------------------
[07/11/2024-03:58:15] [TRT-LLM] [I] TensorRT-LLM (total latency: 26.520079374313354 sec)
[07/11/2024-03:58:15] [TRT-LLM] [I] TensorRT-LLM (total output tokens: 1435)
[07/11/2024-03:58:15] [TRT-LLM] [I] TensorRT-LLM (tokens per second: 54.1099436297277)
[07/11/2024-03:58:15] [TRT-LLM] [I] TensorRT-LLM beam 0 result
[07/11/2024-03:58:15] [TRT-LLM] [I]   rouge1 : 23.372917363794926
[07/11/2024-03:58:15] [TRT-LLM] [I]   rouge2 : 7.036605777466268
[07/11/2024-03:58:15] [TRT-LLM] [I]   rougeL : 17.62796112937931
[07/11/2024-03:58:15] [TRT-LLM] [I]   rougeLsum : 20.722932087556057
```

说明：
以上示例仍需注意参数 `--hf_model_dir` 的格式，需要与 Hugging Face 网站模型的标准命名格式一致。
更多运行示例请参见 NVIDIA TensorRT-LLM ChatGLM Demo

注意事项

说明：
由于 OpenCloudOS 是 TencentOS Server 的开源版本，理论上上述文档当中的所有操作同样适用于 OpenCloudOS。

参考文档

- NVIDIA TensorRT-LLM GitHub
- TensorRT-LLM v0.10.0版本 log 日志
- NVIDIA TensorRT-LLM ChatGLM Demo
- Hugging Face 镜像网站
- cnn_dailymail 数据集
- Hugging Face THUDM/chatglm3-6b 模型

OpenVINO CLIP

最近更新时间：2024-08-13 09:55:21

本指导适用于在 TencentOS Server 3 上使用 OpenVINO 推理框架运行 CLIP 模型的官方 Demo，以 Docker 方式启动。

OpenVINO 环境准备

由于从 OpenVINO 官方 GitHub 仓库使用 Dockerfile 构建 OpenVINO 镜像速度较慢，这里我们通过直接拉取 Dockerhub 上 OpenVINO 的镜像包构建OpenVINO 开发环境。
由于直接从 DockerHub openvino 使用 Docker pull 拉取镜像会由于访问限制问题拉取失败，这里我们通过腾讯云 Docker 镜像源加速镜像下载。

配置腾讯云镜像加速源

1. 打开系统中 `/etc/docker/daemon.json` 配置文件：

```
vim /etc/docker/daemon.json
```

2. 按 i 切换至编辑模式，添加以下内容，请注意 json 代码的格式：

```
{
   "registry-mirrors": [
   "https://mirror.ccs.tencentyun.com"
  ]
}
```

添加完成后格式应类似于：

```
{
    "runtimes": {
        "nvidia": {
            "args": [],
            "path": "nvidia-container-runtime"
        }
    },
    "registry-mirrors": [
    "https://mirror.ccs.tencentyun.com"
  ]
}
```

随后按 ESC 即可退出编辑模式，再输入 !wq 即可保存并退出 vim 环境。

3. 此时 Docker daemon 会被关闭，需要重启 Docker。

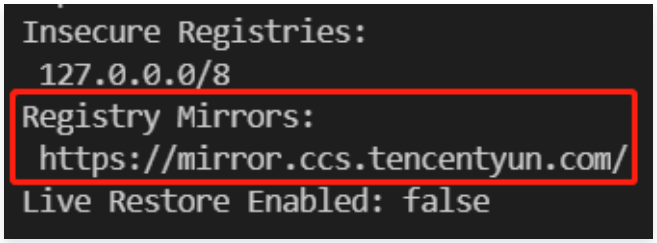
```
sudo systemctl restart docker
```

> ⚠ 注意：
> 如果此时出现以下错误：
> - Job for docker.service failed because the control process exited with error code。
> - See "systemctl status docker.service" and "journalctl -xe" for details。
>
> 说明添加的 `registry-mirrors` 没有按照 json 文件的格式添加，请重新检查格式是否正确，添加正确后请再次重启 Docker。

4. 查看 Docker 状态：

```
sudo docker info
```

此时 Clinet 和 Server 都正确运行，且出现 Registry Mirrors 腾讯云镜像源内容，则说明配置成功。



从 Dockerhub 拉取 OpenVINO 镜像

这里我们选择 ubuntu22_dev，且拉取版本为最新版本。

```
docker run -it -e HF_ENDPOINT="https://hf-mirror.com" --name openvino openvino/ubuntu22_dev:latest
/bin/bash
```

运行模型

模型环境准备

1. 此时应该在 /opt/intel/openvino_2024.2.0.15519/ 路径下，创建 Demo 文件夹来存放模型代码。

```
mkdir -p demo/CLIP
cd demo/CLIP
```

2. 将 pip 换为国内清华源以加快下载速度。

```
#将pip换成清华源
#设为默认，永久有效
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

3. 安装运行 CLIP 模型必要的环境（大部分运行模型需要的包已经在镜像里存在了，这里是检查一遍比较重要的包，确保后续运行不会出错）：

```
pip install --extra-index-url https://download.pytorch.org/whl/cpu "gradio>=4.19" "openvino>=2023.1.0"
"transformers[torch]>=4.30" "datasets" "nncf>=2.6.0" "torch>=2.1" Pillow "matplotlib>=3.4"
```

本指导使用的 CLIP 的 backbone 为 `clip-vit-base-patch16`，对于给定的图像并输入 N 条 Prompts 进行 zero-shot 推理，确定图像属于概率最高的一个 label，进行 CLIP zero-shot 图像分类任务。

下载模型权重地址换源

由于中国大陆无法下载 Hugging Face 网站模型，首先需要对下载网站换源，使用国内镜像网站的 HF-Mirror 模型。

> **说明：**
> 如果 docker run 的时候加上了 `-e HF_ENDPOINT="https://hf-mirror.com"`，则此步可以跳过。

```
#单次有效，退出容器且暂停容器运行后失效，再次重启进入容器需重新输入此条命令
export HF_ENDPOINT="https://hf-mirror.com"
```

> **注意：**
> 这里使用 `echo 'export HF_ENDPOINT="https://hf-mirror.com"' >> ~/.bashrc` 的命令仍然会导致下载失败，请勿使用。

创建模型代码

1. 在 demo/CLIP 文件夹下创建 clip.py 文件，并写入以下代码，这里我们使用原始 Pytorch 和 OpenVINO 两种方式实现图像分类推理：

```
from transformers import CLIPProcessor, CLIPModel
import requests
from pathlib import Path
from PIL import Image
import os
import openvino as ov
from scipy.special import softmax
import numpy as np


# load pre-trained model
model = CLIPModel.from_pretrained("openai/clip-vit-base-patch16")
# load preprocessor for model input
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch16")


# download sample
sample_path = Path("data/coco.jpg")
if os.path.exists(sample_path):
    print("sample exists.")
else:
    print("download sample.")
    sample_path.parent.mkdir(parents=True, exist_ok=True)
    r =
requests.get("https://storage.openvinotoolkit.org/repositories/openvino_notebooks/data/data/image/coco.
jpg")


    with sample_path.open("wb") as f:
        f.write(r.content)


image = Image.open(sample_path)


input_labels = [
    "cat",
    "dog",
    "wolf",
    "tiger",
    "man",
    "horse",
    "frog",
    "tree",
    "house",
    "computer",
]
text_descriptions = [f"This is a photo of a {label}" for label in input_labels]
print(text_descriptions)
inputs = processor(text=text_descriptions, images=[image], return_tensors="pt", padding=True)


results = model(**inputs)
logits_per_image = results["logits_per_image"]  # this is the image-text similarity score
probs = logits_per_image.softmax(dim=1).detach().numpy()  # we can take the softmax to get the label
probabilities
print("Label probs using Pytorch:", probs)
print("Pytorch prediction result:", input_labels[np.argmax(probs)])
```

```
# use OpenVINO to run the CLIP model
fp16_model_path = Path("clip-vit-base-patch16.xml")
model.config.torchscript = True


# convert model to openvino format
if not fp16_model_path.exists():
    ov_model = ov.convert_model(model, example_input=dict(inputs))
    ov.save_model(ov_model, fp16_model_path)


# create OpenVINO core object instance
core = ov.Core()
# check running device
device = "CPU"


# compile model for loading on device
compiled_model = core.compile_model(fp16_model_path, device)
# run inference on preprocessed data and get image-text similarity score
ov_logits_per_image = compiled_model(dict(inputs))[0]
# perform softmax on score
probs = softmax(ov_logits_per_image, axis=1)
print("Label probs using OpenVINO:", probs)
print("OpenVINO prediction result:", input_labels[np.argmax(probs)])
```

运行模型：

```
python3 clip.py
```

2. **首先会下载模型，随后会下载测试的图像，保存在 data/coco.jpg 下，图像如下图所示：**

> **注意：**
> **第一次下载图像可以会出现下载失败，这是由于网络原因导致的，多尝试几次即可成功下载测试图像。**

3. **接下来，我们定义了10个标签，分别是：**

```
"cat",
"dog",
"wolf",
"tiger",
```

```
  "man",
  "horse",
  "frog",
  "tree",
  "house",
  "computer",
```

并将10个 labels 分别带入 prompts 模板 `This is a photo of a {label}` 并作为最终 prompts 输出进 CLIP text encoder 生成 embeddings，最终与模型图像 vector 点乘得到 logits 预测结果。
得到 Pytorch 结果后，本示例会将模型转换为 OpenVINO 的形式再次推理一遍。请注意由于 OpenVINO 不支持使用 Nvidia GPU 而只能使用 Intel GPU 进行推理，所以本示例在 CPU 上推理模型。

模型输出

可以看到输入给 CLIP text encoder 的 prompts 全部为：

```
['This is a photo of a cat',
 'This is a photo of a dog',
 'This is a photo of a wolf',
 'This is a photo of a tiger',
 'This is a photo of a man',
 'This is a photo of a horse',
 'This is a photo of a frog',
 'This is a photo of a tree',
 'This is a photo of a house',
 'This is a photo of a computer']
```

输出的 Pytorch 推理和 OpenVINO 推理的结果如下（参考）：

```
#Pytorch推理结果
Label probs using Pytorch: [[6.9000979e-04 9.8858720e-01 3.0764646e-04 1.2136426e-04 7.0234956e-03
  6.7098060e-04 2.5553041e-04 1.6874878e-04 1.8124736e-03 3.6242907e-04]]
Pytorch prediction result: dog
#OpenVINO推理结果
Label probs using OpenVINO: [[7.1112876e-04 9.8869681e-01 3.1731726e-04 1.3581685e-04 6.7636110e-03
  6.6217897e-04 2.3762971e-04 1.6567315e-04 1.9223638e-03 3.8736628e-04]]
OpenVINO prediction result: dog
```

可以看到无论是 Pytorch 还是 OpenVINO，推理图像的结果均为 dog。

注意事项

> **说明：**
> 由于 OpenCloudOS 是 TencentOS Server 的开源版本，理论上上述文档当中的所有操作同样适用于 OpenCloudOS。

参考文档

- 腾讯云 Docker 镜像源配置
- DockerHub openvino/ubuntu22_dev 镜像
- openvino clip-zero-shot-classification-with-output Demo
- openvino Interactive Tutorials
- Hugging Face openai/clip-vit-base-patch16 模型
- Hugging Face 镜像网站

BLIP

最近更新时间：2024-08-13 09:55:21

本指导适用于在 TencentOS Server 3 上使用 OpenVINO 推理框架运行 BLIP 模型的官方 Demo，以 Docker 方式启动。

前置环境条件

请确保已经按照 CLIP 文档内进行操作，**运行模型之前的所有步骤已经完成，并已经准备好了 OpenVINO 的所有必要环境。**

运行模型

模型环境准备

1. 此时应该在 `/opt/intel/openvino_2024.2.0.15519/` 路径下，创建 Demo 文件夹来存放模型代码。

```
mkdir -p demo/BLIP
cd demo/BLIP
```

2. 将 pip 换为国内清华源以加快下载速度。

```
#将pip换成清华源
#设为默认，永久有效
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

3. 安装运行 BLIP 模型必要的环境（大部分运行模型需要的包已经在镜像里存在了，这里是检查一遍比较重要的包，确保后续运行不会出错）：

```
pip install --extra-index-url https://download.pytorch.org/whl/cpu "torch>=2.1.0" torchvision
"transformers>=4.26.0" "gradio>=4.19" "openvino>=2023.3.0" "datasets>=2.14.6" "tqdm" "matplotlib>=3.4"
```

本指导使用的 BLIP 的 backbone 为 `blip-vqa-base`，对于给定的图像，我们可以进行两种任务：

- **图像描述生成**（Image Caption）。
- **图像视觉问答**（Visual Question Answering）。

下载模型权重地址换源

由于中国大陆无法下载 Hugging Face 网站模型，首先需要对下载网站换源，使用国内镜像网站的 HF-Mirror 模型。

> **说明：**
> 如果 docker run 的时候加上了 `-e HF_ENDPOINT="https://hf-mirror.com"`，则此步可以跳过。

```
#单次有效，退出容器且暂停容器运行后失效，再次重启进入容器需重新输入此条命令
export HF_ENDPOINT="https://hf-mirror.com"
```

> **注意：**
> 这里使用 `echo 'export HF_ENDPOINT="https://hf-mirror.com"' >> ~/.bashrc` 的命令仍然会导致下载失败，请勿使用。

创建模型代码

1. 在 demo/BLIP 文件夹下创建 blip.py 文件，并写入以下代码：

```
import time
import os
import requests
from PIL import Image
from pathlib import Path
from transformers import BlipProcessor, BlipForQuestionAnswering
```

```
import torch
from pathlib import Path
import openvino as ov


from functools import partial
from blip_model import text_decoder_forward
from blip_model import OVBlipModel


# get model and processor
processor = BlipProcessor.from_pretrained("Salesforce/blip-vqa-base")
model = BlipForQuestionAnswering.from_pretrained("Salesforce/blip-vqa-base")


# setup test input: download image
sample_path = Path("data/demo.jpg")
if os.path.exists(sample_path):
    print("sample exists.")
else:
    print("download sample.")
    sample_path.parent.mkdir(parents=True, exist_ok=True)
    r = requests.get("https://storage.googleapis.com/sfr-vision-language-research/BLIP/demo.jpg")


    with sample_path.open("wb") as f:
        f.write(r.content)


# read image, prepare question
raw_image = Image.open(sample_path).convert("RGB")
question = "how many dogs are in the picture?"


# preprocess input data
inputs = processor(raw_image, question, return_tensors="pt")


# Question Answering
start = time.perf_counter()
out = model.generate(**inputs)
end = time.perf_counter() - start


# postprocess result
answer = processor.decode(out[0], skip_special_tokens=True)
print(f"question: {question} Answer: {answer}")
print(f"Processing time: {end:.4f} s\n")


# use OpenVINO to run the BLIP model


# blip vision model to OpenVINO format
VISION_MODEL_OV = Path("blip_vision_model.xml")
vision_model = model.vision_model
vision_model.eval()
```

```
# check that model works and save it outputs for reusage as text encoder input
with torch.no_grad():
    vision_outputs = vision_model(inputs["pixel_values"])


# if openvino model does not exist, convert it to IR
if not VISION_MODEL_OV.exists():
    # export pytorch model to ov.Model
    with torch.no_grad():
        ov_vision_model = ov.convert_model(vision_model, example_input=inputs["pixel_values"])
    # save model on disk for next usages
    ov.save_model(ov_vision_model, VISION_MODEL_OV)
    print(f"Vision model successfuly converted and saved to {VISION_MODEL_OV}")
else:
    print(f"Vision model will be loaded from {VISION_MODEL_OV}")




# blip text encoder to OpenVINO format
TEXT_ENCODER_OV = Path("blip_text_encoder.xml")
text_encoder = model.text_encoder
text_encoder.eval()


# if openvino model does not exist, convert it to IR
if not TEXT_ENCODER_OV.exists():
    # prepare example inputs
    image_embeds = vision_outputs[0]
    image_attention_mask = torch.ones(image_embeds.size()[:-1], dtype=torch.long)
    input_dict = {
        "input_ids": inputs["input_ids"],
        "attention_mask": inputs["attention_mask"],
        "encoder_hidden_states": image_embeds,
        "encoder_attention_mask": image_attention_mask,
    }
    # export PyTorch model
    with torch.no_grad():
        ov_text_encoder = ov.convert_model(text_encoder, example_input=input_dict)
    # save model on disk for next usages
    ov.save_model(ov_text_encoder, TEXT_ENCODER_OV)
    print(f"Text encoder successfuly converted and saved to {TEXT_ENCODER_OV}")
else:
    print(f"Text encoder will be loaded from {TEXT_ENCODER_OV}")




# blip text decoder to OpenVINO format
TEXT_DECODER_OV = Path("blip_text_decoder_with_past.xml")
text_decoder = model.text_decoder
text_decoder.eval()


# prepare example inputs
input_ids = torch.tensor([[30522]])  # begin of sequence token id
attention_mask = torch.tensor([[1]])  # attention mask for input_ids
encoder_hidden_states = torch.rand((1, 10, 768))  # encoder last hidden state from text_encoder
encoder_attention_mask = torch.ones((1, 10), dtype=torch.long)  # attention mask for encoder hidden
states
```

```
input_dict = {
    "input_ids": input_ids,
    "attention_mask": attention_mask,
    "encoder_hidden_states": encoder_hidden_states,
    "encoder_attention_mask": encoder_attention_mask,
}
text_decoder_outs = text_decoder(**input_dict)
# extend input dictionary with hidden states from previous step
input_dict["past_key_values"] = text_decoder_outs["past_key_values"]


text_decoder.config.torchscript = True
if not TEXT_DECODER_OV.exists():
    # export PyTorch model
    with torch.no_grad():
        ov_text_decoder = ov.convert_model(text_decoder, example_input=input_dict)
    # save model on disk for next usages
    ov.save_model(ov_text_decoder, TEXT_DECODER_OV)
    print(f"Text decoder successfuly converted and saved to {TEXT_DECODER_OV}")
else:
    print(f"Text decoder will be loaded from {TEXT_DECODER_OV}")




# create OpenVINO Core object instance
core = ov.Core()
# check running device
device = "CPU"


# load models on device
ov_vision_model = core.compile_model(VISION_MODEL_OV, device)
ov_text_encoder = core.compile_model(TEXT_ENCODER_OV, device)
ov_text_decoder_with_past = core.compile_model(TEXT_DECODER_OV, device)


text_decoder.forward = partial(text_decoder_forward,
ov_text_decoder_with_past=ov_text_decoder_with_past)
ov_model = OVBlipModel(model.config, model.decoder_start_token_id, ov_vision_model, ov_text_encoder,
text_decoder)


# Image Captioning
out = ov_model.generate_caption(inputs["pixel_values"], max_length=20)
caption = processor.decode(out[0], skip_special_tokens=True)
print(f"caption: {caption}")


# Question Answering
start = time.perf_counter()
out = ov_model.generate_answer(**inputs, max_length=20)
end = time.perf_counter() - start
answer = processor.decode(out[0], skip_special_tokens=True)
print(f"question: {question} Answer: {answer}")
print(f"Processing time: {end:.4f} s")
```

2. 创建 blip_model.py，写入以下代码：

```
import torch
import numpy as np
import openvino as ov
from typing import List, Dict
from transformers.modeling_outputs import CausalLMOutputWithCrossAttentions



def init_past_inputs(model_inputs: List):
    """
    Helper function for initialization of past inputs on first inference step
    Parameters:
      model_inputs (List): list of model inputs
    Returns:
      pkv (List[ov.Tensor]): list of filled past key values
    """
    pkv = []
    for input_tensor in model_inputs[4:]:
        partial_shape = input_tensor.partial_shape
        partial_shape[0] = 1
        partial_shape[2] = 0
        pkv.append(ov.Tensor(ov.Type.f32, partial_shape.get_shape()))
    return pkv



def postprocess_text_decoder_outputs(output: Dict):
    """
    Helper function for rearranging model outputs and wrapping to CausalLMOutputWithCrossAttentions
    Parameters:
      output (Dict): dictionary with model output
    Returns
      wrapped_outputs (CausalLMOutputWithCrossAttentions): outputs wrapped to
CausalLMOutputWithCrossAttentions format
    """
    logits = torch.from_numpy(output[0])
    past_kv = list(output.values())[1:]
    return CausalLMOutputWithCrossAttentions(
        loss=None,
        logits=logits,
        past_key_values=past_kv,
        hidden_states=None,
        attentions=None,
        cross_attentions=None,
    )



def text_decoder_forward(
    ov_text_decoder_with_past: ov.CompiledModel,
    input_ids: torch.Tensor,
    attention_mask: torch.Tensor,
    past_key_values: List[ov.Tensor],
    encoder_hidden_states: torch.Tensor,
    encoder_attention_mask: torch.Tensor,
    **kwargs
):
```

```
    """
    Inference function for text_decoder in one generation step
    Parameters:
      input_ids (torch.Tensor): input token ids
      attention_mask (torch.Tensor): attention mask for input token ids
      past_key_values (List[ov.Tensor] list of cached decoder hidden states from previous step
      encoder_hidden_states (torch.Tensor): encoder (vision or text) hidden states
      encoder_attention_mask (torch.Tensor): attnetion mask for encoder hidden states
    Returns
      model outputs (CausalLMOutputWithCrossAttentions): model prediction wrapped to
CausalLMOutputWithCrossAttentions class including predicted logits and hidden states for caching
    """
    inputs = [input_ids, attention_mask, encoder_hidden_states, encoder_attention_mask]
    if past_key_values is None:
        inputs.extend(init_past_inputs(ov_text_decoder_with_past.inputs))
    else:
        inputs.extend(past_key_values)
    outputs = ov_text_decoder_with_past(inputs)
    return postprocess_text_decoder_outputs(outputs)




class OVBlipModel:
    """
    Model class for inference BLIP model with OpenVINO
    """


    def __init__(
        self,
        config,
        decoder_start_token_id: int,
        vision_model,
        text_encoder,
        text_decoder,
    ):
        """
        Initialization class parameters
        """
        self.vision_model = vision_model
        self.vision_model_out = vision_model.output(0)
        self.text_encoder = text_encoder
        self.text_encoder_out = text_encoder.output(0)
        self.text_decoder = text_decoder
        self.config = config
        self.decoder_start_token_id = decoder_start_token_id
        self.decoder_input_ids = config.text_config.bos_token_id


    def generate_answer(self, pixel_values: torch.Tensor, input_ids: torch.Tensor, attention_mask:
torch.Tensor, **generate_kwargs):
        """
        Visual Question Answering prediction
        Parameters:
          pixel_values (torch.Tensor): preprocessed image pixel values
          input_ids (torch.Tensor): question token ids after tokenization
          attention_mask (torch.Tensor): attention mask for question tokens
        Retruns:
          generation output (torch.Tensor): tensor which represents sequence of generated answer token
ids
```

```
        """
        image_embed = self.vision_model(pixel_values.detach().numpy())[self.vision_model_out]
        image_attention_mask = np.ones(image_embed.shape[:-1], dtype=int)
        if isinstance(input_ids, list):
            input_ids = torch.LongTensor(input_ids)
        question_embeds = self.text_encoder(
            [
                input_ids.detach().numpy(),
                attention_mask.detach().numpy(),
                image_embed,
                image_attention_mask,
            ]
        )[self.text_encoder_out]
        question_attention_mask = np.ones(question_embeds.shape[:-1], dtype=int)


        bos_ids = np.full((question_embeds.shape[0], 1), fill_value=self.decoder_start_token_id)


        outputs = self.text_decoder.generate(
            input_ids=torch.from_numpy(bos_ids),
            eos_token_id=self.config.text_config.sep_token_id,
            pad_token_id=self.config.text_config.pad_token_id,
            encoder_hidden_states=torch.from_numpy(question_embeds),
            encoder_attention_mask=torch.from_numpy(question_attention_mask),
            **generate_kwargs,
        )
        return outputs


    def generate_caption(self, pixel_values: torch.Tensor, input_ids: torch.Tensor = None,
attention_mask: torch.Tensor = None, **generate_kwargs):
        """
        Image Captioning prediction
        Parameters:
          pixel_values (torch.Tensor): preprocessed image pixel values
          input_ids (torch.Tensor, *optional*, None): pregenerated caption token ids after
tokenization, if provided caption generation continue provided text
          attention_mask (torch.Tensor): attention mask for caption tokens, used only if input_ids
provided
        Retruns:
          generation output (torch.Tensor): tensor which represents sequence of generated caption token
ids
        """
        batch_size = pixel_values.shape[0]


        image_embeds = self.vision_model(pixel_values.detach().numpy())[self.vision_model_out]


        image_attention_mask = torch.ones(image_embeds.shape[:-1], dtype=torch.long)


        if isinstance(input_ids, list):
            input_ids = torch.LongTensor(input_ids)
        elif input_ids is None:
            input_ids = torch.LongTensor(
                [
                    [
                        self.config.text_config.bos_token_id,
                        self.config.text_config.eos_token_id,
```

```
                ]
            ]
        ).repeat(batch_size, 1)
    input_ids[:, 0] = self.config.text_config.bos_token_id
    attention_mask = attention_mask[:, :-1] if attention_mask is not None else None


    outputs = self.text_decoder.generate(
        input_ids=input_ids[:, :-1],
        eos_token_id=self.config.text_config.sep_token_id,
        pad_token_id=self.config.text_config.pad_token_id,
        attention_mask=attention_mask,
        encoder_hidden_states=torch.from_numpy(image_embeds),
        encoder_attention_mask=image_attention_mask,
        **generate_kwargs,
    )


    return outputs
```

代码会首先使用 Pytorch 运行模型进行图像视觉问答任务，随后会将模型转换为 OpenVINO 的形式进行图像描述生成和图像视觉问答任务。
运行模型：

```
python3 blip.py
```

首先会下载模型，随后会下载测试的图像，保存在 data/demo.jpg 下，图像如下图所示：

3. **随后会使用 Pytorch 的方法先进行视觉问答，并计算运行的时间，结果如下（参考）：**

```
question: how many dogs are in the picture? Answer: 1
Processing time: 0.2532 s
```

4. **接下来代码会将模型转换为 OpenVINO 的形式，并进行图像描述生成和图像视觉问答任务，结果如下（参考）：**

```
caption: dog is sitting on beach
question: how many dogs are in the picture? Answer: 1
```

```
Processing time: 0.1179 s
```

第一行为图像描述的结果，第二、三行为图像视觉问答的结果，可以看到使用 OpenVINO 进行视觉问答的推理速度要显著快于单纯使用 Pytorch 推理。

注意事项

> **说明：**
> 由于 OpenCloudOS 是 TencentOS Server 的开源版本，理论上上述文档当中的所有操作同样适用于 OpenCloudOS。

参考文档

- openvino blip-visual-language-processing Demo
- Hugging Face 镜像网站
- openvino Interactive Tutorials
- Hugging Face Salesforce/blip-vqa-base 模型

HuggingFace TGI LLaMA

最近更新时间：2024-08-13 09:55:21

本指导适用于在 TencentOS Server 3 上使用 HuggingFace TGI 推理框架运行 LLaMA 模型的官方 Demo，以 Docker 方式启动。

HuggingFace TGI 环境准备

拉取 HuggingFace TGI 相关镜像，并同时配置下载模型的镜像源，这里我们测试 LLaMA-7b 模型：

```
docker run -it --name HFTGI_llama_7b --gpus all -e HF_ENDPOINT="https://hf-mirror.com" -e HF_HUB_OFFLINE=1
-p 8080:80 -v $PWD/data:/data ghcr.io/huggingface/text-generation-inference:latest --model-id openlm-
research/open_llama_7b --trust-remote-code
```

说明：

该命令有以下几点需要说明：

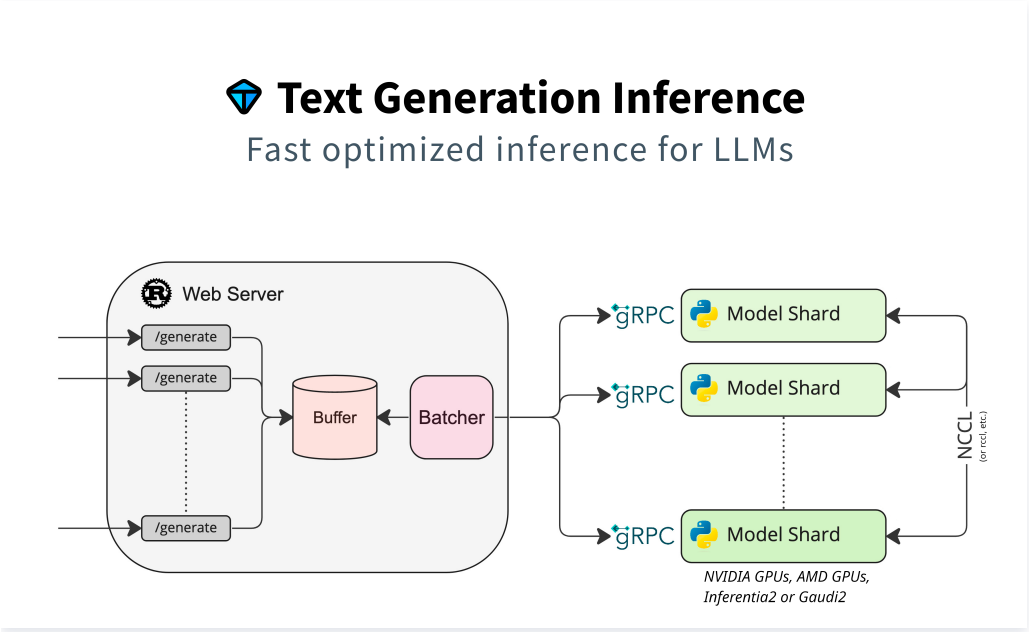
- 由于中国大陆无法下载 Hugging Face 网站模型，首先需要对下载网站换源，使用国内镜像网站的 HF-Mirror 模型。其中 `-e HF_ENDPOINT="https://hf-mirror.com"` 就是将下载模型的网站换为镜像网站， `-e` 必须加上。
- 如果模型为 Hugging Face 中的 Gated Models（即需要 User access tokens），请参见文档 Stable Diffusion 中添加 User access tokens 部分的内容。注意也要添加 `-e` ，即此时命令为 `-e HF_ENDPOINT="https://hf-mirror.com" -e HF_TOKEN=example_token` ，其中 example_token 为在网站上生成的 token。
- 由于国内无法连接 Hugging Face hub，代码会调用 Hugging Face API 并长时间等待响应，且最终还是无法连接。为了避免长时间等待，根据 router/src/main.rs 文件第212行代码 `if std::env::var("HF_HUB_OFFLINE") == Ok("1".to_string())` 设置环境变量 HF_HUB_OFFLINE=1，这样就不会去调用 Hugging Face API 而是直接 offline 推理，避免了长时间的等待。
- 该命令在拉取镜像之后会下载模型，为了避免容器如果被删除后模型反复下载，使用 `-v $PWD/data:/data` 将容器内部 `/data` 文件夹里下载的模型通过挂载的方式在服务器本地 `$PWD/data` 也进行保存。
- 下载镜像的地址为 `ghcr.io/huggingface/text-generation-inference:latest` ，由于 `ghcr.io` 国内可替代的镜像源几乎没有，请耐心等待下载。如出现 `docker: unexpected EOF` ， 等错误则是由于下载太慢导致，反复尝试下载即可。如实在下载失败次数太多，可以将 `ghcr.io` 改为 `ghcr.chenby.cn` ，可以加快下载速度（但不保证将来此镜像源还可以使用，仅供参考）。
- --model-id 参数为 openlm-research/open_llama_7b，该名称需要与 Hugging Face 官网名称一致。本指导我们测试 LLaMA-7b，请注意 LLaMA-3b-V2 模型经测试暂不支持使用 HuggingFace TGI 推理框架加速运行，会出现 `RuntimeError: Unsupported head size: 100` 的错误。

一旦镜像和模型都准备好，会看到如下内容（参考）：

```
...
INFO hf_hub: Token file not found "/root/.cache/huggingface/token"
INFO text_generation_launcher: Default `max_input_tokens` to 4095
INFO text_generation_launcher: Default `max_total_tokens` to 4096
INFO text_generation_launcher: Default `max_batch_prefill_tokens` to 4145
INFO text_generation_launcher: Using default cuda graphs [1, 2, 4, 8, 16, 32]
INFO download: text_generation_launcher: Starting check and download process for openlm-
research/open_llama_7b
INFO text_generation_launcher: Detected system cuda
INFO text_generation_launcher: Files are already present on the host. Skipping download.
INFO download: text_generation_launcher: Successfully downloaded weights for openlm-research/open_llama_7b
INFO shard-manager: text_generation_launcher: Starting shard rank=0
INFO text_generation_launcher: Detected system cuda
INFO text_generation_launcher: Server started at unix:///tmp/text-generation-server-0
INFO shard-manager: text_generation_launcher: Shard ready in 5.904282524s rank=0
INFO text_generation_launcher: Starting Webserver
WARN text_generation_router: router/src/main.rs:218: Offline mode active using cache defaults
INFO text_generation_router: router/src/main.rs:349: Using config Some(Llama)
```

```
WARN text_generation_router: router/src/main.rs:351: Could not find a fast tokenizer implementation for
openlm-research/open_llama_7b
WARN text_generation_router: router/src/main.rs:352: Rust input length validation and truncation is
disabled
WARN text_generation_router: router/src/main.rs:358: no pipeline tag found for model openlm-
research/open_llama_7b
WARN text_generation_router: router/src/main.rs:376: Invalid hostname, defaulting to 0.0.0.0
INFO text_generation_router::server: router/src/server.rs:1577: Warming up model
INFO text_generation_launcher: Cuda Graphs are enabled for sizes [32, 16, 8, 4, 2, 1]
INFO text_generation_router::server: router/src/server.rs:1604: Using scheduler V3
INFO text_generation_router::server: router/src/server.rs:1656: Setting max batch total tokens to 58928
INFO text_generation_router::server: router/src/server.rs:1894: Connected
```

由于 HuggingFace TGI 是使用 Web Server 的形式交互（如下图所示），可以看到最后显示 connected，则说明 Webserver 连接成功。



注意：

如果出现以下错误：

torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 64.00 MiB. GPU 。

如果报错说明显存不足，则无法运行。

退出容器后重启 Web Server

如果退出容器，且退出容器后容器停止运行，重新启动容器即可重启 Web Server，命令如下：

```
#重新启动容器
docker start HFTGI_llama_7b

#实时查看Web Server的运行状态
docker logs -f HFTGI_llama_7b
```

重新启动后会自动启动 Web Server 进程，即可通过服务器本地发送请求运行模型。

运行模型

当前窗口连接 Web Server 成功之后，请不要关掉窗口以及做任何可能使 Web Server 停止的活动（查看 GPU 显存使用情况即可知道 Web Server 是否还在运行中）。

以 bash 方式交互运行

另外开一个窗口，输入以下命令：

```
curl -v http://localhost:8080/generate -X POST -d '{"inputs":"What is Deep Learning?","parameters":
{"max_new_tokens":20}}' -H 'Content-Type: application/json'
```

inputs 则为输入给模型的 prompt，同时窗口返回输出如下（参考）：

```
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8080 (#0)
> POST /generate HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.61.1
> Accept: */*
> Content-Type: application/json
> Content-Length: 70
>
* upload completely sent off: 70 out of 70 bytes
< HTTP/1.1 200 OK
< content-type: application/json
< x-compute-type: 1-nvidia-l40
< x-compute-time: 0.426205521
< x-compute-characters: 22
< x-total-time: 426
< x-validation-time: 0
< x-queue-time: 0
< x-inference-time: 426
< x-time-per-token: 21
< x-prompt-tokens: 4075
< x-generated-tokens: 20
< content-length: 126
< vary: origin, access-control-request-method, access-control-request-headers
< access-control-allow-origin: *
< date: Thu, 18 Jul 2024 08:14:28 GMT
<
* Connection #0 to host localhost left intact
{"generated_text":"\nDeep Learning is a subfield of Machine Learning that uses artificial neural networks
to solve problems."}
```

说明模型使用 HuggingFace TGI 推理框架运行模型成功，同时在 Web Server 窗口可以看到以下内容：

```
INFO generate{parameters=GenerateParameters { best_of: None, temperature: None, repetition_penalty: None,
frequency_penalty: None, top_k: None, top_p: None, typical_p: None, do_sample: false, max_new_tokens:
Some(20), return_full_text: None, stop: [], truncate: None, watermark: false, details: false,
decoder_input_details: false, seed: None, top_n_tokens: None, grammar: None, adapter_id: None }
total_time="426.205521ms" validation_time="37.499µs" queue_time="64.159µs" inference_time="426.104203ms"
time_per_token="21.30521ms" seed="None"}: text_generation_router::server: router/src/server.rs:322:
Success
```

也证明 HuggingFace TGI 推理框架运行 LLaMA-7b 成功并正常返回输出。

以 TGI Client 方式运行

另外开一个窗口，首先在服务器本地安装运行 TGI Client 必要的包，使用 pip 安装：

```
#将pip换成清华源加快下载速度
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

```
#安装text-generation包
pip install text-generation
```

安装完成后，在服务器本地新建 TGI_Client.py 文件，输入以下代码：

```
from text_generation import Client


# Generate
client = Client("http://localhost:8080")
output = client.generate("Why is the sky blue?", max_new_tokens=92).generated_text
print(f"Generate Output: {output}")


# Generate stream
text = ""
for response in client.generate_stream("Why is the sky blue?", max_new_tokens=92):
    if not response.token.special:
        text += response.token.text
print(f"Generate stream Output: {text}")
```

执行代码文件：

```
python TGI_Client.py
```

以上代码会以 client.generate 的方式输出 output 以及 client.generate_stream 以字符流的形式输出，输出结果如下（参考）：

```
Generate Output:
The sky is blue because of the scattering of light by the atmosphere.
The scattering of light by the atmosphere is caused by the presence of molecules in the atmosphere.
The molecules in the atmosphere are made up of atoms.
The atoms in the atmosphere are made up of protons, neutrons, and electrons.
The protons, neutrons, and electrons in the atmosphere are arranged in a way that allows them to scatter
light.

Generate stream Output:
The sky is blue because of the scattering of light by the atmosphere.
The scattering of light by the atmosphere is caused by the presence of molecules in the atmosphere.
The molecules in the atmosphere are made up of atoms.
The atoms in the atmosphere are made up of protons, neutrons, and electrons.
The protons, neutrons, and electrons in the atmosphere are arranged in a way that allows them to scatter
light.
```

同时 Web Server 窗口会输出：

```
INFO compat_generate{default_return_full_text=true compute_type=Extension(ComputeType("1-nvidia-
l40"))}:generate_stream{parameters=GenerateParameters { best_of: None, temperature: None,
repetition_penalty: None, frequency_penalty: None, top_k: None, top_p: None, typical_p: None, do_sample:
false, max_new_tokens: Some(92), return_full_text: Some(false), stop: [], truncate: None, watermark:
false, details: true, decoder_input_details: false, seed: None, top_n_tokens: None, grammar: None,
adapter_id: None } total_time="1.942765325s" validation_time="22.049µs" queue_time="40.4µs"
inference_time="1.942703056s" time_per_token="21.116337ms" seed="None"}: text_generation_router::server:
router/src/server.rs:511: Success
```

表明运行成功。

运行 LangChain

另外开一个窗口，首先在服务器本地安装运行 LangChain 必要的包，使用 pip 安装：

```
#将pip换成清华源加快下载速度
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple

#安装必要的包
pip install langchain transformers langchain_community
```

安装完成后，在服务器本地新建 LangChain.py 文件，输入以下代码：

```
# Wrapper to TGI client with langchain


from langchain.llms import HuggingFaceTextGenInference
from langchain import PromptTemplate, LLMChain


inference_server_url_local = "http://localhost:8080"


llm_local = HuggingFaceTextGenInference(
    inference_server_url=inference_server_url_local,
    max_new_tokens=400,
    top_k=10,
    top_p=0.95,
    typical_p=0.95,
    temperature=0.7,
    repetition_penalty=1.03,
)


question = "whats 2 * (1 + 2)"
template = """Question: {question}
Answer: Let's think step by step."""


prompt = PromptTemplate(
    template=template,
    input_variables= ["question"]
)


llm_chain_local = LLMChain(prompt=prompt, llm=llm_local)
output = llm_chain_local({question})
print(output)
```

执行代码文件：

```
python LangChain.py
```

我们让模型计算2 * (1 + 2)，并逐步输出，模型输出如下（参考）：

```
{'question': {'whats 2 * (1 + 2)'}, 'text': '\nStep 1. We start with 2 * (1 + 2)\nStep 2. Now, we multiply
the first number by 2 and add it to the second number.\nStep 3. So we have 2 * (1 + 2) = 2 * 3\nStep 4.
Now we multiply the first number by 2 again and add it to the third number.\nStep 5. So we have 2 * 3 =
6\nStep 6. Now, we add the first number to the third number.\nStep 7. So we have 6 = 2 + 3\nStep 8. Now we
add the first number to the second number.\nStep 9. So we have 6 = 2 + 3 + 3\nStep 10. Now we add the
first number to the first number.\nStep 11. So we have 6 = 2 + 3 + 3 + 3\nStep 12. Now we add the first
```

```
number to the second number.\nStep 13. So we have 6 = 2 + 3 + 3 + 3 + 3\nStep 14. Now we add the first
number to the first number.\nStep 15. So we have 6 = 2 + 3 + 3 + 3 + 3 + 3\nStep 16. Now we add the first
number to the second number.\nStep 17. So we have 6 = 2 + 3 + 3 + 3 + 3 + 3 + 3\nStep 18. Now we add the
first number to the first number.\nStep 19. So we have 6 = 2 + 3 + 3 + 3 + 3 + 3 + 3 +'}
```

可以看到有正常输出，表明运行成功，同时 Web Server 窗口会输出：

```
INFO compat_generate{default_return_full_text=true compute_type=Extension(ComputeType("1-nvidia-
140"))}:generate{parameters=GenerateParameters { best_of: None, temperature: Some(0.7),
repetition_penalty: Some(1.03), frequency_penalty: None, top_k: Some(10), top_p: Some(0.95), typical_p:
Some(0.95), do_sample: false, max_new_tokens: Some(400), return_full_text: Some(false), stop: [],
truncate: None, watermark: false, details: true, decoder_input_details: false, seed: None, top_n_tokens:
None, grammar: None, adapter_id: None } total_time="8.646906066s" validation_time="20.12µs"
queue_time="65.869µs" inference_time="8.646820347s" time_per_token="21.61705ms"
seed="Some(15119903715577785552)"}: text_generation_router::server: router/src/server.rs:322: Success
```

注意事项

> **说明：**
> 由于 OpenCloudOS 是 TencentOS Server 的开源版本，理论上上述文档当中的所有操作同样适用于 OpenCloudOS。

参考文档

- Hugging Face 镜像网站
- Hugging Face User Access Tokens
- huggingface/text-generation-inference GitHub
- Hugging Face text-generation-inference basic_tutorials gated_model_access
- Hugging Face text-generation-inference supported_models
- Hugging Face text-generation-inference 使用指引
- DockerHub 镜像源1
- DockerHub 镜像源2

Baichuan

最近更新时间：2024-08-13 09:55:21

本指导适用于在 TencentOS Server 3 上使用 HuggingFace TGI 推理框架运行 Baichuan 模型的官方 Demo，以 Docker 方式启动。

HuggingFace TGI 环境准备

拉取 HuggingFace TGI 相关镜像，并同时配置下载模型的镜像源，这里我们测试 baichuan-13b-chat 模型：

```
docker run -it --name HFTGI_baichuan_13b_chat --gpus all -e HF_ENDPOINT="https://hf-mirror.com" -e
HF_HUB_OFFLINE=1 -p 8080:80 -v $PWD/data:/data ghcr.io/huggingface/text-generation-inference:latest --
model-id baichuan-inc/Baichuan-13B-Chat --trust-remote-code
```

如遇相关错误，请参见 LLaMA 文章的相关说明部分。
一旦镜像和模型都准备好，会看到如下内容（参考）：

```
...
INFO hf_hub: Token file not found "/root/.cache/huggingface/token"
INFO text_generation_launcher: Default `max_input_tokens` to 4095
INFO text_generation_launcher: Default `max_total_tokens` to 4096
INFO text_generation_launcher: Default `max_batch_prefill_tokens` to 4145
INFO text_generation_launcher: Using default cuda graphs [1, 2, 4, 8, 16, 32]
WARN text_generation_launcher: `trust_remote_code` is set. Trusting that model `baichuan-inc/Baichuan-13B-
Chat` do not contain malicious code.
INFO download: text_generation_launcher: Starting check and download process for baichuan-inc/Baichuan-
13B-Chat
INFO text_generation_launcher: Detected system cuda
WARN text_generation_launcher: No safetensors weights found for model baichuan-inc/Baichuan-13B-Chat at
revision None. Converting PyTorch weights to safetensors.
INFO text_generation_launcher: Convert: [1/3] -- Took: 0:00:10.643016
INFO text_generation_launcher: Convert: [2/3] -- Took: 0:00:09.686510
INFO text_generation_launcher: Convert: [3/3] -- Took: 0:00:06.149757
INFO download: text_generation_launcher: Successfully downloaded weights for baichuan-inc/Baichuan-13B-
Chat
INFO shard-manager: text_generation_launcher: Starting shard rank=0
INFO text_generation_launcher: Detected system cuda
INFO shard-manager: text_generation_launcher: Waiting for shard to be ready... rank=0
INFO text_generation_launcher: Server started at unix:///tmp/text-generation-server-0
INFO shard-manager: text_generation_launcher: Shard ready in 12.109051544s rank=0
INFO text_generation_launcher: Starting Webserver
WARN text_generation_router: router/src/main.rs:218: Offline mode active using cache defaults
INFO text_generation_router: router/src/main.rs:349: Using config Some(Baichuan)
WARN text_generation_router: router/src/main.rs:351: Could not find a fast tokenizer implementation for
baichuan-inc/Baichuan-13B-Chat
WARN text_generation_router: router/src/main.rs:352: Rust input length validation and truncation is
disabled
WARN text_generation_router: router/src/main.rs:358: no pipeline tag found for model baichuan-
inc/Baichuan-13B-Chat
WARN text_generation_router: router/src/main.rs:376: Invalid hostname, defaulting to 0.0.0.0
INFO text_generation_router::server: router/src/server.rs:1577: Warming up model
INFO text_generation_launcher: Cuda Graphs are enabled for sizes [32, 16, 8, 4, 2, 1]
INFO text_generation_router::server: router/src/server.rs:1604: Using scheduler V3
INFO text_generation_router::server: router/src/server.rs:1656: Setting max batch total tokens to 21872
INFO text_generation_router::server: router/src/server.rs:1894: Connected
```

由于 HuggingFace TGI 是使用 Web Server 的形式交互，可以看到最后显示 connected，则说明 Webserver 连接成功。

退出容器后重启 Web Server

如果退出容器，且退出容器后容器停止运行，重新启动容器即可重启 Web Server，命令如下：

```
#重新启动容器
docker start HFTGI_baichuan_13b_chat

#实时查看Web Server的运行状态
docker logs -f HFTGI_baichuan_13b_chat
```

重新启动后会自动启动 Web Server 进程，即可通过服务器本地发送请求运行模型。

运行模型

当前窗口连接 Web Server 成功之后，请不要关掉窗口以及做任何可能使 Web Server 停止的活动（查看 GPU 显存使用情况即可知道 Web Server 是否还在运行中）。

以 bash 方式交互运行

另外开一个窗口，输入以下命令：

```
curl -v http://localhost:8080/generate -X POST -d '{"inputs":"What is Deep Learning?","parameters":
{"max_new_tokens":68}}' -H 'Content-Type: application/json'
```

inputs 则为输入给模型的 prompt，同时窗口返回输出如下（参考）：

```
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8080 (#0)
> POST /generate HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.61.1
> Accept: */*
> Content-Type: application/json
> Content-Length: 70
>
* upload completely sent off: 70 out of 70 bytes
< HTTP/1.1 200 OK
< content-type: application/json
< x-compute-type: 1-nvidia-140
< x-compute-time: 2.758545962
< x-compute-characters: 22
< x-total-time: 2758
< x-validation-time: 0
< x-queue-time: 0
< x-inference-time: 2758
< x-time-per-token: 40
< x-prompt-tokens: 4027
< x-generated-tokens: 68
< content-length: 333
< vary: origin, access-control-request-method, access-control-request-headers
< access-control-allow-origin: *
< date: Thu, 18 Jul 2024 02:40:43 GMT
<
* Connection #0 to host localhost left intact
{"generated_text":"\nDeep learning is a subset of artificial intelligence (AI) that focuses on the
development of algorithms that can learn automatically from data. It can be used to learn from data and
improve the performance of machines. It can be used to learn from data and improve the performance of
artificial intelligence.\n"}
```

说明模型使用 HuggingFace TGI 推理框架运行模型成功，同时在 Web Server 窗口可以看到以下内容：

```
INFO generate{parameters=GenerateParameters { best_of: None, temperature: None, repetition_penalty: None,
frequency_penalty: None, top_k: None, top_p: None, typical_p: None, do_sample: false, max_new_tokens:
Some(68), return_full_text: None, stop: [], truncate: None, watermark: false, details: false,
decoder_input_details: false, seed: None, top_n_tokens: None, grammar: None, adapter_id: None }
total_time="2.758545962s" validation_time="34.56µs" queue_time="67.668µs" inference_time="2.758443884s"
time_per_token="40.565351ms" seed="None"}: text_generation_router::server: router/src/server.rs:322:
Success
```

以 TGI Client 方式运行

另外开一个窗口，首先在服务器本地安装运行 TGI Client 必要的包，使用 pip 安装：

```
#将pip换成清华源加快下载速度
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple

#安装text-generation包
pip install text-generation
```

安装完成后，在服务器本地新建 TGI_Client.py 文件，输入以下代码：

```
from text_generation import Client


# Generate
client = Client("http://localhost:8080")
output = client.generate("Why is the sky blue?", max_new_tokens=92).generated_text
print(f"Generate Output: {output}")


# Generate stream
text = ""
for response in client.generate_stream("Why is the sky blue?", max_new_tokens=92):
    if not response.token.special:
        text += response.token.text
print(f"Generate stream Output: {text}")
```

执行代码文件：

```
python TGI_Client.py
```

以上代码会以 client.generate 的方式输出 output 以及 client.generate_stream 以字符流的形式输出，输出结果如下（参考）：

```
Generate Output:
The sky is blue because of the Rayleigh scattering of sunlight. This is due to the Rayleigh scattering of
light by atmospheric gases. This scattering of sunlight by gases in the atmosphere, such as ozone, which
is responsible for the blue color of the sky. The shorter wavelengths of light.

Generate stream Output:
The sky is blue because of the Rayleigh scattering of sunlight. This is due to the Rayleigh scattering of
light by atmospheric gases. This scattering of sunlight by gases in the atmosphere, such as ozone, which
is responsible for the blue color of the sky. The shorter wavelengths of light.
```

同时 Web Server 窗口会输出：

```
INFO compat_generate{default_return_full_text=true compute_type=Extension(ComputeType("1-nvidia-
l40"))}:generate_stream{parameters=GenerateParameters { best_of: None, temperature: None,
repetition_penalty: None, frequency_penalty: None, top_k: None, top_p: None, typical_p: None, do_sample:
false, max_new_tokens: Some(72), return_full_text: Some(false), stop: [], truncate: None, watermark:
false, details: true, decoder_input_details: false, seed: None, top_n_tokens: None, grammar: None,
adapter_id: None } total_time="2.920170273s" validation_time="37.26µs" queue_time="65.668µs"
inference_time="2.920067525s" time_per_token="40.556493ms" seed="None"}: text_generation_router::server:
router/src/server.rs:511: Success
```

表明运行成功。

运行 LangChain

另外开一个窗口，首先在服务器本地安装运行 LangChain 必要的包，使用 pip 安装：

```
#将pip换成清华源加快下载速度
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple

#安装必要的包
pip install langchain transformers langchain_community
```

安装完成后，在服务器本地新建 LangChain.py 文件，输入以下代码：

```
# Wrapper to TGI client with langchain


from langchain.llms import HuggingFaceTextGenInference
from langchain import PromptTemplate, LLMChain


inference_server_url_local = "http://localhost:8080"


llm_local = HuggingFaceTextGenInference(
    inference_server_url=inference_server_url_local,
    max_new_tokens=400,
    top_k=10,
    top_p=0.95,
    typical_p=0.95,
    temperature=0.7,
    repetition_penalty=1.03,
)


question = "whats 2 * (1 + 2)"
template = """Question: {question}
Answer: Let's think step by step."""


prompt = PromptTemplate(
    template=template,
    input_variables= ["question"]
)


llm_chain_local = LLMChain(prompt=prompt, llm=llm_local)
output = llm_chain_local({question})
print(output)
```

执行代码文件：

```
python LangChain.py
```

我们让模型计算2 * (1 + 2)，并逐步输出，模型输出如下（参考）：

```
{'question': {'whats 2 * (1 + 2)'}, 'text': ' 1. What is the result of multiplication? Multiplication can
be done in two steps or not? The answer to this question will help you understand whether it works, and if
there are any solutions for a given solution. In order to evaluate its efficiency when solving problems
with an addition operation.\n\nProblem-solving approach based on problem transformation
methodology{together.}'}
```

可以看到有正常输出，表明运行成功，同时 Web Server 窗口会输出：

```
INFO compat_generate{default_return_full_text=true compute_type=Extension(ComputeType("1-nvidia-
140"))}:generate{parameters=GenerateParameters { best_of: None, temperature: Some(0.7),
repetition_penalty: Some(1.03), frequency_penalty: None, top_k: Some(10), top_p: Some(0.95), typical_p:
Some(0.95), do_sample: false, max_new_tokens: Some(400), return_full_text: Some(false), stop: [],
truncate: None, watermark: false, details: true, decoder_input_details: false, seed: None, top_n_tokens:
None, grammar: None, adapter_id: None } total_time="3.446902913s" validation_time="24.82µs"
queue_time="80.588µs" inference_time="3.446797695s" time_per_token="41.033305ms"
seed="Some(6173130784801218210)"}: text_generation_router::server: router/src/server.rs:322: Success
```

注意事项

> **说明：**
> 由于 OpenCloudOS 是 TencentOS Server 的开源版本，理论上上述文档当中的所有操作同样适用于 OpenCloudOS。

参考文档

- Hugging Face 镜像网站
- Hugging Face text-generation-inference GitHub
- Hugging Face Text Generation Inference (TGI) 使用指引

LMDeploy ChatGLM

最近更新时间：2024-08-13 09:55:21

本指导适用于在 TencentOS Server 3 上使用 LMDeploy 推理框架运行 ChatGLM 模型的官方 Demo，以 Docker 方式启动。

LMDeploy 环境准备

本文指导我们通过直接拉取 Dockerhub 上 LMDeploy 的镜像包构建 LMDeploy 开发环境。
由于直接从 Dockerhub openmmlab/lmdeploy 使用 Docker pull 拉取镜像会由于访问限制问题拉取失败，这里我们通过腾讯云 Docker 镜像源加速镜像下载。

配置腾讯云镜像加速源

1. 打开系统中 `/etc/docker/daemon.json` 配置文件：

```
vim /etc/docker/daemon.json
```

2. 按 i 切换至编辑模式，添加以下内容，请注意 `json` 代码的格式：

```
{
   "registry-mirrors": [
   "https://mirror.ccs.tencentyun.com"
  ]
}
```

添加完成后格式应类似于：

```
{
    "runtimes": {
        "nvidia": {
            "args": [],
            "path": "nvidia-container-runtime"
        }
    },
    "registry-mirrors": [
    "https://mirror.ccs.tencentyun.com"
  ]
}
```

3. 随后按 **ESC** 即可退出编辑模式，再输入 **!wq** 即可保存并退出 vim 环境。
4. 此时 Docker daemon 会被关闭，需要重启 Docker。

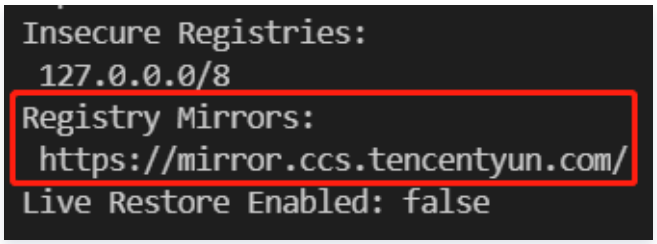
```
sudo systemctl restart docker
```

> ⚠ **注意：**
> 如果此时出现以下错误：
> - Job for docker.service failed because the control process exited with error code。
> - See "systemctl status docker.service" and "journalctl -xe" for details。
> - 说明添加的 **registry-mirrors** 没有按照 json 文件的格式添加，请重新检查格式是否正确，添加正确后请再次重启 Docker。

5. 查看 Docker 状态：

```
sudo docker info
```

此时 Clinet 和 Server 都正确运行，且出现 Registry Mirrors 内容，则说明配置成功。



从 Dockerhub 拉取 LMDeploy 镜像

这里我们选择最新版本进行拉取。

```
docker run -it --name lmdeploy --runtime nvidia --gpus all -e HF_ENDPOINT="https://hf-mirror.com" -v
$PWD:/root/.cache/huggingface openmmlab/lmdeploy:latest
```

> **说明：**
> - 由于中国大陆无法下载 Hugging Face 网站模型，这里使用国内镜像网站的 HF-Mirror 模型，所以加上 -e 的参数写入环境变量。
> - -v 使用挂载，其中 $PWD 为服务器本地存放模型的位置，这样在容器中下载的模型也会在服务器本地 $PWD 存放一份。

运行模型

模型环境准备

此时应该在 `/opt/lmdeploy` 路径下，创建 Demo 文件夹来存放模型代码。

```
mkdir -p demo/ChatGLM
cd demo/ChatGLM
```

创建模型代码

在 `demo/ChatGLM` 文件夹下创建 ChatGLM.py 文件，并写入以下代码，这里我们测试的是 chatglm2-6b 模型：

```
from transformers import AutoTokenizer, AutoModel
from lmdeploy import pipeline
import time


# LMDeploy testing
# load model
pipe = pipeline('THUDM/chatglm2-6b')


# inference
start = time.perf_counter()
response1 = pipe(['Hi, pls intro yourself'])
response2 = pipe(['Shanghai is'])
response3 = pipe(['晚上睡不着应该怎么办'])
end = time.perf_counter() - start


# print output and time
print(f"{response1}\n{response2}\n{response3}")
print(f"Processing time: {end:.4f}")
```

运行模型：

```
python3 ChatGLM.py
```

首先会下载模型，随后运行得到输出，输出如下（参考）：

```
[Response(text=' Hello! I am ChatGLM2-6B, an AI Assistant based on the GLM model developed by Knowledge
Engineering Group, Tsinghua University and Zhipu.AI. I have been trained to understand and respond to
various types of questions proposed by users.', generate_token_len=56, input_token_len=22, session_id=0,
finish_reason='stop', token_ids=[13755, 30992, 307, 674, 22011, 10461, 30944, 30943, 30941, 30978, 30949,
30932, 284, 11265, 10695, 1767, 331, 267, 12980, 30944, 2092, 3596, 422, 16397, 7556, 3439, 30932, 304,
9744, 30919, 5144, 1451, 293, 1192, 899, 30923, 30930, 23833, 30930, 307, 431, 628, 7594, 289, 1617, 293,
3758, 289, 2463, 3608, 290, 2554, 5126, 422, 3386, 30930], logprobs=None)]
[Response(text=' 上海是中国的一个城市，位于中国东部沿海地区。上海是中国重要的经济、文化和科技中心之一，也是全球著名的港口城
市。上海拥有悠久的历史和文化，也是中国近代化和现代化的象征之一。', generate_token_len=42, input_token_len=20,
session_id=0, finish_reason='stop', token_ids=[39537, 54532, 32914, 31623, 31733, 31123, 32308, 31626,
36282, 36986, 31808, 31155, 31770, 33657, 32333, 31675, 31201, 37229, 31748, 31673, 31905, 31123, 31677,
31879, 34811, 36437, 31733, 31155, 31770, 32104, 39230, 33397, 40565, 31123, 31677, 31626, 35838, 38785,
45877, 35423, 31905, 31155], logprobs=None)]
[Response(text=' 以下是一些有助于晚上睡觉的技巧：\n\n1. 维持规律的睡眠时间：尽量在同一时间上床睡觉和起床，即使在周末和假期也
应保持一致。\n\n2. 创建舒适的睡眠环境：保持卧室安静、黑暗、凉爽和舒适。如果需要，可以使用睡眠面罩、耳塞或空气净化器。\n\n3. 改
善睡眠前的活动：在睡前数小时内避免使用电子设备，如手机、电视或电脑。这些设备发出的光线和刺激可能会干扰睡眠。\n\n4. 放松身心：在
睡前半小时进行一些轻松的活动，如阅读或听轻柔的音乐，有助于放松身心。\n\n5. 限制咖啡因和酒精：避免在睡前数小时内摄入咖啡因和酒精，
这些物质会影响睡眠质量。\n\n6. 锻炼身体：适当的身体锻炼可以帮助放松身心，减轻压力，改善睡眠质量。\n\n如果这些方法不能解决你的问
题，可能需要咨询医生或睡眠专家。', generate_token_len=207, input_token_len=20, session_id=0,
finish_reason='stop', token_ids=[30910, 32040, 43742, 35434, 32820, 35228, 54530, 33741, 30954, 13, 13,
30939, 30930, 30910, 33945, 33700, 54530, 34192, 31643, 30954, 33840, 54534, 34342, 31643, 54547, 55629,
35228, 54542, 39360, 30932, 48002, 35042, 54542, 34469, 54574, 54720, 31983, 33297, 31155, 13, 13, 30943,
30930, 30910, 32991, 47108, 34192, 31747, 30954, 31983, 40210, 36055, 31201, 36038, 31201, 56165, 56558,
54542, 33894, 31155, 31654, 31665, 30932, 46566, 34192, 54610, 56553, 31201, 55808, 55959, 54746, 33300,
38125, 55083, 31155, 13, 13, 30966, 30930, 30910, 32537, 34192, 33711, 31651, 30954, 54534, 43612, 54744,
41942, 32317, 31695, 32157, 32077, 30932, 54627, 32113, 31201, 32092, 54746, 34032, 31155, 31704, 32077,
47357, 43642, 54542, 33464, 32934, 37272, 34192, 31155, 13, 13, 30972, 30930, 30910, 34794, 34532, 30954,
54534, 43612, 40132, 31636, 31784, 33550, 37498, 30932, 54627, 32232, 54746, 55208, 55215, 56059, 41512,
30932, 35434, 34794, 34532, 31155, 13, 13, 30970, 30930, 30910, 32922, 33745, 54674, 54542, 37314, 30954,
32317, 54534, 43612, 54744, 41942, 36620, 33745, 54674, 54542, 37314, 30932, 31704, 32580, 39041, 34192,
31838, 31155, 13, 13, 30978, 30930, 30910, 33748, 31902, 30954, 40215, 31902, 33748, 41230, 34794, 34532,
30932, 35763, 32507, 30932, 32537, 34192, 31838, 31155, 13, 13, 31654, 31704, 31847, 31727, 31868, 31822,
31639, 30932, 31667, 31665, 32539, 32718, 54746, 34192, 32114, 31155], logprobs=None)]
Processing time: 7.1545
```

可以看到输出的内容和运行的时间，表明运行成功。
如果只想输出所对应的 text 内容，只需更改如下代码：

```
#更改此行代码
print(f"{response1}\n{response2}\n{response3}")

#更改为
print(f"{response1[0].text}\n{response2[0].text}\n{response3[0].text}")
```

即可看到只有 text 文本的输出，输出如下（参见）：

```
Hello! I am ChatGLM2-6B, an AI Assistant based on the GLM model developed by Knowledge Engineering Group,
Tsinghua University and Zhipu.AI. I have been trained to understand and respond to various types of
questions proposed by users.

上海是中国的一个城市，位于中国东部沿海地区。它是中国最大的城市之一，也是世界第六大城市。上海是一个现代化的城市，拥有许多国际知
名的企业和金融机构。上海也是一个文化和艺术中心，有许多博物馆和剧院。
```

以下时是一些有助于放松和促进睡眠的技巧：

1. 创建一个放松的睡眠环境。确保房间安静、黑暗、凉爽和舒适。如果需要，可以使用睡眠面罩、耳塞或空气净化器来帮助创造一个更舒适的睡眠环境。

2. 创造一个睡前惯例。例如，洗澡，做一些伸展运动，或者喝一杯温热的饮料。

3. 避免在睡前吃大量的食物或饮料。特别是咖啡因和酒精。

4. 避免在睡前使用电子设备。这些设备会发出蓝光，可能会干扰睡眠。

5. 适当的运动。但请避免在睡前进行剧烈的运动，以免影响睡眠。

6. 放松技巧。试着进行冥想、瑜伽或渐进性肌肉松弛练习来放松身体和头脑。

7. 避免躺在床上超过20分钟。如果躺在床上无法入睡，不要继续躺在床上，可以去做一些轻松的活动，然后再回到床上。

8. 如果躺在床上无法入睡，不要看电视或使用电子设备。这些活动可能会干扰睡眠。

9. 考虑使用一些帮助入睡的产品，如香薰精油、舒适的枕头或床垫等。

10. 如果这些技巧不能解决你的问题，建议咨询医生或睡眠专家。他们可以为你提供更深入的帮助。

注意事项

> **说明：**
> 由于 OpenCloudOS 是 TencentOS Server 的开源版本，理论上上述文档当中的所有操作同样适用于 OpenCloudOS。

参考文档

- 安装 Docker 并配置镜像加速源
- Hugging Face 镜像网站
- lmdeploy LLM 离线推理 pipeline Demo

LLaMA

最近更新时间：2024-08-13 09:55:21

本指导适用于在 TencentOS Server 3 上使用 LMDeploy 推理框架运行 LLaMA 模型的官方 Demo，以 Docker 方式启动。

前置环境条件

请确保已经按照 ChatGLM 文档内进行操作，运行模型之前的所有步骤已经完成，并已经准备好了 LMDeploy 的所有必要环境。

运行模型

模型环境准备

此时应该在 `/opt/lmdeploy` 路径下，创建 Demo 文件夹来存放模型代码。

```
mkdir -p demo/LLaMA
cd demo/LLaMA
```

创建模型代码

在 demo/LLaMA 文件夹下创建 LLaMA.py 文件，并写入以下代码，这里我们测试的是 llama_7b 模型：

```
from transformers import AutoTokenizer, AutoModel
from lmdeploy import pipeline
import time




# LMDeploy testing
# load model
pipe = pipeline('openlm-research/open_llama_7b')


# inference
start = time.perf_counter()
response1 = pipe(['The president of the United States is'])
response2 = pipe(['Shanghai is'])
response3 = pipe(['The future of AI is'])
end = time.perf_counter() - start


# print output and time
print(f"{response1[0].text}\n{response2[0].text}\n{response3[0].text}")
print(f"Processing time: {end:.4f}")
```

运行模型：

```
python3 LLaMA.py
```

运行该 Python 文件，则会自动开始下载 LLaMA_7b 模型并开始推理，输出结果如下（参见）：

```
 not the commander in chief of the armed forces of the United States. The president is the commander in
chief of the armed forces of the United States in a very limited sense.
He is the commander in chief of the armed forces of the United States, but he is not the commander in
chief of the armed forces of the United States in the sense that he has any right to tell them what to do.
The commander in chief of the armed forces of the United States is the Congress of the United States. The
commander in chief of the armed forces of the United States is the Congress of the United States.
```

```
When the Congress of the United States says that it is going to go to war, the commander in chief of the
armed forces of the United States is the Congress of the United States.
When the Congress of the United States says that it is going to go to war, the commander in chief of the
armed forces of the United States is the Congress of the United States. The commander in chief of the
armed forces of the United States is the Congress of the United States.
When the Congress of the United States says that it is going to go to war, the commander in chief of the
armed forces of the United States is the Congress of the United States. When the Congress of the United
States says that it is going to go to war, the commander in chief of the armed forces of the United States
is the Congress of the United States.
When the Congress of the United States says that it is going to go to war, the commander in chief of the
armed forces of the United States is the Congress of the United States. When the Congress of the United
States says that it is going to go to war, the commander in chief of the armed forces of the United States
is the Congress of the United States. When the Congress of the United States says that it is going to go
to war, the commander in chief of the armed forces of the United States is the Congress of the United
States.
When the Congress of the United States says that it is going to go to war, the commander in chief of the
armed forces of the United States is the Congress of the United States. When the Congress of the United
States says that it is going to go to war, the commander in chief of the armed forces of the United States
is the Congress of the United States. When the Congress of the United States says that it is going to go
to war, the commander in chief of the armed forces of the United States is the Congress
 one of the most famous cities in China. It is also one of the largest cities in the world. The city is
located on the Huangpu River and is surrounded by the Yangtze River Delta. The city is the largest city in
China and is the second largest city in the world.
Shanghai is a very popular tourist destination. The city has a lot of attractions, including the Bund, the
Shanghai Museum, the Yu Garden, the Shanghai Tower, and the Shanghai Zoo. The city is also home to the
Shanghai World Expo.
The city is also home to the Shanghai International Film Festival. The city is also home to the Shanghai
International Film Festival. The city is also home to the Shanghai International Film Festival. The city
is also home to the Shanghai International Film Festival. The city is also home to the Shanghai
International Film Festival. The city is also home to the Shanghai International Film Festival. The city
is also home to the Shanghai International Film Festival. The city is also home to the Shanghai
International Film Festival. The city is also home to the Shanghai International Film Festival. The city
is also home to the Shanghai International Film Festival. The city is also home to the Shanghai
International Film Festival. The city is also home to the Shanghai International Film Festival. The city
is also home to the Shanghai International Film Festival. The city is also home to the Shanghai
International Film Festival. The city is also home to the Shanghai International Film Festival. The city
is also home to the Shanghai International Film Festival. The city is also home to the Shanghai
International Film Festival. The city is also home to the Shanghai International Film Festival. The city
is also home to the Shanghai International Film Festival. The city is also home to the Shanghai
International Film Festival. The city is also home to the Shanghai International Film Festival. The city
is also home to the Shanghai International Film Festival. The city is also home to the Shanghai
International Film Festival. The city is also home to the Shanghai International Film Festival. The city
is also home to the Shanghai International Film Festival. The city is also home to the Shanghai
International Film Festival. The city is also home to the Shanghai International Film Festival. The city
is also home to the Shanghai International Film Festival. The city is also home to the Shanghai
International Film Festival. The city is also home to the Shanghai International Film Festival. The city
is also home to the Shanghai International Film Festival. The city is also home to the Shanghai
International Film Festival. The city is also home to the Shanghai International Film Festival. The city
is also home to
 in the hands of the people
The AI revolution is here, but it is also not here. The promise of a world where humans and machines work
together to solve our biggest problems is exciting and has the potential to transform every industry.
However, the reality is that AI is still in its infancy and the risks associated with its misuse are real.
As we enter a new era of AI, it is important that we consider the ethical implications of this technology
and ensure that it is used for the benefit of all. This requires a multidisciplinary approach that
includes the perspectives of experts from a variety of fields, including ethics, law, and technology.
In this article, we will explore the ethical implications of AI and discuss how we can ensure that this
technology is used for the good of all.
AI's impact on society
The impact of AI on society is likely to be profound. It has the potential to transform how we live and
work, but it also has the potential to cause harm. As we enter a new era of AI, it is important that we
```

```
consider the ethical implications of this technology and ensure that it is used for the benefit of all.
AI and the future of work
AI is already having a significant impact on the future of work. As machines become increasingly capable
of performing tasks that were once thought to be the domain of humans, it is likely that many jobs will be
replaced by AI. This could lead to a decrease in employment and an increase in inequality.
AI and the future of healthcare
AI has the potential to revolutionize healthcare. It can be used to diagnose diseases, provide treatment
recommendations, and even perform surgeries. However, it is important to consider the ethical implications
of this technology.
AI and the future of education
AI has the potential to transform education. It can be used to create personalized learning experiences
for students, and it can also be used to assess student performance and provide feedback. However, it is
important to consider the ethical implications of this technology.
AI and the future of government
AI has the potential to transform how governments operate. It can be used to make decisions about policy,
allocate resources, and even provide personalized services to citizens. However, it is important to
consider the ethical implications of this technology.
AI and the future of security
AI has the potential to transform how we secure our homes, businesses, and countries. It can be used to
detect and prevent crime, and it can also be used to identify potential threats. However, it is important
to consider the ethical
Processing time: 30.2124
```

可以看到输出结果和运行的时间，说明运行成功。

注意事项

> **说明：**
> 由于 OpenCloudOS 是 TencentOS Server 的开源版本，理论上上述文档当中的所有操作同样适用于 OpenCloudOS。

参考文档

- Hugging Face 镜像网站
- lmdeploy LLM 离线推理 pipeline Demo

TencentOS Server 3 运行主要深度学习应用框架示例
TencentOS Server 3 运行环境及目录

最近更新时间：2024-08-12 14:44:52

本指导适用于在 TencentOS Server 3 上以 Docker 的方式运行主要深度学习应用框架示例。

硬软件环境

- **Description:** TencentOS Server release 3.1 (Final)。
- **Architecture:** x86_64。
- **CPU op-mode(s):** 32-bit, 64-bit。
- **CUDA Version:** 12.2。
- **GPU:** NVIDIA L40。

操作系统详情：

```
LSB Version:    :core-4.1-amd64:core-4.1-noarch
Distributor ID: TencentOSServer
Description:    TencentOS Server release 3.1 (Final)
Release:        3.1
Codename:       Final
```

CPU 详情：

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                48
On-line CPU(s) list:   0-47
Thread(s) per core:    2
Core(s) per socket:    24
Socket(s):             1
NUMA node(s):          1
Vendor ID:             AuthenticAMD
BIOS Vendor ID:        Red Hat
CPU family:            25
Model:                 17
Model name:            AMD EPYC 9K84 96-Core Processor
BIOS Model name:       3.0
Stepping:              0
CPU MHz:               2600.026
BogoMIPS:              5200.05
Hypervisor vendor:     KVM
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              32768K
NUMA node0 CPU(s):     0-47
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm constant_tsc rep_good nopl nonstop_tsc cpuid extd_apicid amd_dcm tsc_known_freq
 pni pclmulqdq monitor ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm cmp_legacy cr8_legacy abm sse4a misalignsse 3dnowprefetch osvw topoext perfctr_core invpcid_single ibpb v
mmcall fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid avx512f avx512dq rdseed adx smap avx512ifma clflushopt clwb avx512cd sha_ni avx512bw avx512vl xsaveopt xsavec xgetbv1 avx512_bf16 clzero xsaveerptr wbnoinvd arat avx512vbmi umip avx512_v
bmi2 vaes vpclmulqdq avx512_vnni avx512_bitalg avx512_vpopcntdq rdpid fsrm
```

GPU 详情：

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 535.161.07             Driver Version: 535.161.07   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| 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 L40                     On  | 00000000:23:00.0 Off |                    0 |
| N/A   28C    P0              52W / 300W |      0MiB / 46068MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
```

PCI 总线设备情况：

```
-+-[0000:20]---00.0-[21-23]----00.0-[22-23]----00.0-[23]----00.0  NVIDIA Corporation AD102GL [L40]
 \-[0000:00]-+-00.0  Intel Corporation 82G33/G31/P35/P31 Express DRAM Controller
             +-01.0-[01-02]----00.0-[02]--+-01.0  Cirrus Logic GD 5446
             |                            +-02.0  Red Hat, Inc. Virtio network device
             |                            +-03.0  Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) High Definition Audio Controller
             |                            +-04.0  NEC Corporation uPD720200 USB 3.0 Host Controller
             |                            +-05.0  Red Hat, Inc. Virtio block device
             |                            \-06.0  Red Hat, Inc. Virtio memory balloon
             +-02.0-[03-04]----00.0-[04]--
             +-03.0-[05-06]----00.0-[06]--
             +-04.0  Red Hat, Inc. QEMU PCIe Expander bridge
             +-1f.0  Intel Corporation 82801IB (ICH9) LPC Interface Controller
             +-1f.2  Intel Corporation 82801IR/IO/IH (ICH9R/DO/DH) 6 port SATA Controller [AHCI mode]
             \-1f.3  Intel Corporation 82801I (ICH9 Family) SMBus Controller
```

指南目录

环境准备（必做）

- 环境准备。

应用框架

- Stable Diffusion WebUI。
- Ollama & Open WebUI。
- ChatTTS & WebUI。

环境准备

最近更新时间：2024-08-07 15:55:21

Docker 环境准备

安装 Docker

若有旧版本，需要删除旧版本，不然会有冲突。

```
yum remove docker* -y
```

安装 tlinux-release-docker-ce 包后，会默认使用 http://mirrors.tencent.com/docker-ce/ 目录。

```
#tlinux3.x安装docker-ce repo文件
yum install tencentos-release-docker-ce -y
yum install docker-ce -y
```

启动 Docker daemon

> ⚠ **注意：**
> 为了防止后续运行容器时出现如下错误：
> ERROR: Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?
> 该错误是由于 Docker daemon 没有正确启动导致的，需要先开启 Docker daemon。

```
#开启Docker daemon (server)
sudo systemctl start docker

#使用以下命令检查Docker daemon是否正在运行，此时应该看到Docker Client和Docker Server都在运行中
sudo systemctl status docker
```

安装 NVIDIA Container Toolkit

在容器里运行模型，如果想要在 GPU 上运行，则需要安装 NVIDIA Container Toolkit，可参见 NVIDIA Container Toolkit 安装指引 。

```
#配置安装所需要的库
curl -s -L https://nvidia.github.io/libnvidia-container/stable/rpm/nvidia-container-toolkit.repo | \
sudo tee /etc/yum.repos.d/nvidia-container-toolkit.repo

#配置库以便可以使用各种包
sudo yum-config-manager --enable nvidia-container-toolkit-experimental

#安装NVIDIA Container Toolkit包
sudo yum install -y nvidia-container-toolkit
```

配置 Docker

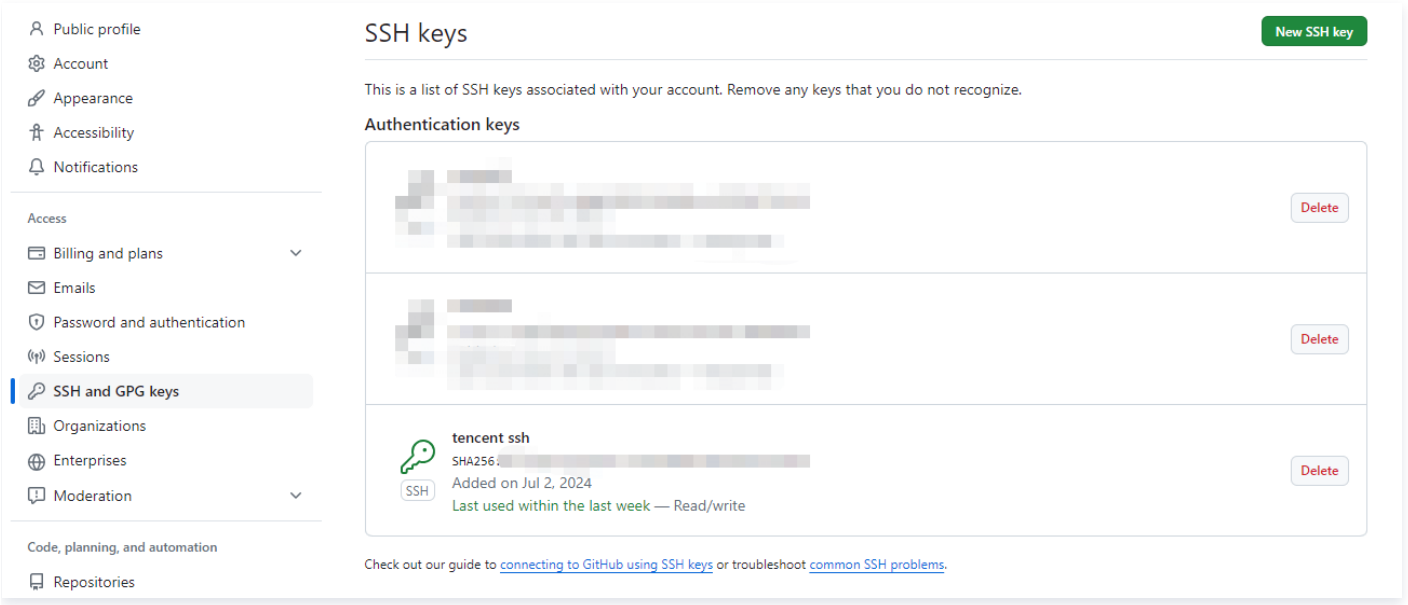
```
#使用nvidia-ctk命令修改并更新/etc/docker/daemon.json主机上的文件，以便Docker可以使用NVIDIA容器运行。
sudo nvidia-ctk runtime configure --runtime=docker

#重新启动Docker
sudo systemctl restart docker
```

配置客户端与远程 GitHub 仓库的连接

> **注意：**
> 为了防止后续拉取GitHub仓库代码出现如下错误：
> fatal: Could not read from remote repository.
> 则说明客户端还没有配置与远程 GitHub 仓库的连接，需要先配置 ssh 远程连接。

```
#生成密钥，其中youremail@example.com需替换成自己在GitHub上注册账号时所用的邮箱
ssh-keygen -t rsa -C "youremail@example.com"
```

打开生成的 /.ssh/id_rsa，复制密钥，在 GitHub 网站登录自己的账户，在账户选项中选择 **Setting > SSH and GPG keys > New SSH key**，把密钥填写到账户中（注意填写时的格式要求）。




```
#测试客户端是否和GitHub远程连接上，出现Success则配置成功
ssh -T git@github.com
```

安装 git 包

检查系统是否安装 git：

```
git
```

> **注意：**
> 如果出现 –bash: git: command not found，则说明环境还没有安装 git 包。

安装 git：

```
#安装git
sudo yum -y install git

#检查git版本
git --version
```

如果此时正常出现 git 的版本号，则说明安装成功。

参考文档

- NVIDIA Container Toolkit 安装指引

- Hugging Face

Stable Diffusion WebUI

最近更新时间：2024-08-13 09:55:21

本指导适用于在 TencentOS Server 3 上运行 Stable Diffusion WebUI 应用框架，以 Docker 方式启动。

Stable Diffusion WebUI 环境准备

从 GitHub 下载 Stable Diffusion WebUI Docker 开源仓库到本地。

```
git clone https://github.com/AbdBarho/stable-diffusion-webui-docker.git
cd stable-diffusion-webui-docker
```

安装对应环境以及下载模型

1. 为了加速模型下载以及防止模型由于网络限制下载失败，我们将下载模型的链接改为镜像网站。打开 stable-diffusion-webui-docker/services/download/links.txt，可以看到如下内容：

```
https://huggingface.co/runwayml/stable-diffusion-v1-5/resolve/main/v1-5-pruned-emaonly.ckpt
  out=Stable-diffusion/v1-5-pruned-emaonly.ckpt
https://huggingface.co/stabilityai/sd-vae-ft-mse-original/resolve/main/vae-ft-mse-840000-ema-
pruned.ckpt
  out=VAE/vae-ft-mse-840000-ema-pruned.ckpt
https://huggingface.co/runwayml/stable-diffusion-inpainting/resolve/main/sd-v1-5-inpainting.ckpt
  out=Stable-diffusion/sd-v1-5-inpainting.ckpt
https://github.com/TencentARC/GFPGAN/releases/download/v1.3.4/GFPGANv1.4.pth
  out=GFPGAN/GFPGANv1.4.pth
https://github.com/xinntao/Real-ESRGAN/releases/download/v0.1.0/RealESRGAN_x4plus.pth
  out=RealESRGAN/RealESRGAN_x4plus.pth
https://github.com/xinntao/Real-ESRGAN/releases/download/v0.2.2.4/RealESRGAN_x4plus_anime_6B.pth
  out=RealESRGAN/RealESRGAN_x4plus_anime_6B.pth
https://heibox.uni-heidelberg.de/f/31a76b13ea27482981b4/?dl=1
  out=LDSR/project.yaml
https://heibox.uni-heidelberg.de/f/578df07c8fc04ffbadf3/?dl=1
  out=LDSR/model.ckpt
```

其中将 `huggingface.co` 更改为 `hf-mirror.com`，在 `https://github.com` 前添加 `https://github.moeyy.xyz/`，更改后的内容为：

```
https://hf-mirror.com/runwayml/stable-diffusion-v1-5/resolve/main/v1-5-pruned-emaonly.ckpt
  out=Stable-diffusion/v1-5-pruned-emaonly.ckpt
https://hf-mirror.com/stabilityai/sd-vae-ft-mse-original/resolve/main/vae-ft-mse-840000-ema-pruned.ckpt
  out=VAE/vae-ft-mse-840000-ema-pruned.ckpt
https://hf-mirror.com/runwayml/stable-diffusion-inpainting/resolve/main/sd-v1-5-inpainting.ckpt
  out=Stable-diffusion/sd-v1-5-inpainting.ckpt
https://github.moeyy.xyz/https://github.com/TencentARC/GFPGAN/releases/download/v1.3.4/GFPGANv1.4.pth
  out=GFPGAN/GFPGANv1.4.pth
https://github.moeyy.xyz/https://github.com/xinntao/Real-
ESRGAN/releases/download/v0.1.0/RealESRGAN_x4plus.pth
  out=RealESRGAN/RealESRGAN_x4plus.pth
https://github.moeyy.xyz/https://github.com/xinntao/Real-
ESRGAN/releases/download/v0.2.2.4/RealESRGAN_x4plus_anime_6B.pth
  out=RealESRGAN/RealESRGAN_x4plus_anime_6B.pth
https://heibox.uni-heidelberg.de/f/31a76b13ea27482981b4/?dl=1
  out=LDSR/project.yaml
https://heibox.uni-heidelberg.de/f/578df07c8fc04ffbadf3/?dl=1
  out=LDSR/model.ckpt
```

说明：

- 要下载 Hugging Face 的内容，必须更换其网站；而 GitHub 的内容可以直接下载，尽管下载速度可能较慢。如果 GitHub 的镜像链接后续失效，也可以尝试使用原始地址下载。
- 若不想更换下载地址，也可以将链接放入浏览器中下载，下载完成后在 stable-diffusion-webui-docker/data/models 文件夹下以链接下方 out 的目录地址保存模型。例如 v1-5-pruned-emaonly.ckpt 保存的地址为 stable-diffusion-webui-

> docker/data/models/Stable-diffusion/v1-5-pruned-emaonly.ckpt，其他的以此类推即可。

2. 模型下载完成后可以看到如下内容（参考）：

```
...
download-1  | Download Results:
download-1  | gid   |stat|avg speed  |path/URI
download-1  | ======+====+===========+=======================================================
download-1  | 075dc9|OK  |       0B/s|/data/models/Stable-diffusion/v1-5-pruned-emaonly.ckpt
download-1  | 9c4025|OK  |       0B/s|/data/models/VAE/vae-ft-mse-840000-ema-pruned.ckpt
download-1  | c84a90|OK  |       0B/s|/data/models/Stable-diffusion/sd-v1-5-inpainting.ckpt
download-1  | f50c51|OK  |       0B/s|/data/models/LDSR/project.yaml
download-1  | c5bab1|OK  |       0B/s|/data/models/LDSR/model.ckpt
download-1  | d8b39c|OK  |    16MiB/s|/data/models/RealESRGAN/RealESRGAN_x4plus.pth
download-1  | 867f11|OK  |   651KiB/s|/data/models/RealESRGAN/RealESRGAN_x4plus_anime_6B.pth
download-1  | 911cd2|OK  |   6.5MiB/s|/data/models/GFPGAN/GFPGANv1.4.pth
download-1  |
download-1  | Status Legend:
download-1  | (OK):download completed.
download-1  | Checking SHAs...
download-1  | /data/models/LDSR/project.yaml: OK
download-1  | /data/models/RealESRGAN/RealESRGAN_x4plus_anime_6B.pth: OK
download-1  | /data/models/RealESRGAN/RealESRGAN_x4plus.pth: OK
download-1  | /data/models/GFPGAN/GFPGANv1.4.pth: OK
download-1  | /data/models/VAE/vae-ft-mse-840000-ema-pruned.ckpt: OK
download-1  | /data/models/LDSR/model.ckpt: OK
download-1  | /data/models/Stable-diffusion/sd-v1-5-inpainting.ckpt: OK
download-1  | /data/models/Stable-diffusion/v1-5-pruned-emaonly.ckpt: OK
download-1  | By using this software, you agree to the following licenses:
download-1  | https://github.com/AbdBarho/stable-diffusion-webui-docker/blob/master/LICENSE
download-1  | https://github.com/CompVis/stable-diffusion/blob/main/LICENSE
download-1  | https://github.com/AUTOMATIC1111/stable-diffusion-webui/blob/master/LICENSE.txt
download-1  | https://github.com/invoke-ai/InvokeAI/blob/main/LICENSE
download-1  | And licenses of all UIs, third party libraries, and extensions.
download-1 exited with code 0
```

其中 Download Results 的 stat 列显示全部下载 OK，以及 Checking SHAs...之后的每一个模型 SHA 安全散列算法都是匹配的并显示 OK。

3. 现在，通过命令可以看到 Stable Diffusion WebUI 的 Docker 容器已经被创建了：

```
docker ps -a

#可以看到如下内容（参考）
CONTAINER ID   IMAGE                                                COMMAND                  CREATED
STATUS                      PORTS     NAMES
3f58c9a2903c   webui-docker-download                                "/docker/download.sh"    25
minutes ago   Exited (0) 24 minutes ago            webui-docker-download-1
```

运行 Stable Diffusion WebUI

本指导的 WebUI 拥有两种 UI 界面：auto 和 comfy，这里分别介绍两种 UI 的安装。

auto UI

1. 该UI界面源自 GitHub AUTOMATIC1111/stable-diffusion-webui，首先需安装各种环境。
2. 打开 stable-diffusion-webui-docker/services/AUTOMATIC1111/Dockerfile，可以看到该文件包含多个 GitHub 链接。为了加快下载速度，可以将每个 GitHub 链接前面加上 GitHub 镜像网站，以使用国内镜像站点进行下载。此外，由于 Dockerfile 中包含 `pip install -r requirements_versions.txt` 命令，使用 pip 安装各种包可能会很慢，因此需要将其替换为国内镜像源。整体修改后的 Dockerfile 文件如下：

```
FROM alpine/git:2.36.2 as download

COPY clone.sh /clone.sh

RUN . /clone.sh stable-diffusion-webui-assets
https://github.moeyy.xyz/https://github.com/AUTOMATIC1111/stable-diffusion-webui-assets.git
6f7db241d2f8ba7457bac5ca9753331f0c266917

RUN . /clone.sh stable-diffusion-stability-ai https://github.moeyy.xyz/https://github.com/Stability-
AI/stablediffusion.git cf1d67a6fd5ea1aa600c4df58e5b47da45f6bdbf \
  && rm -rf assets data/**/*.png data/**/*.jpg data/**/*.gif

RUN . /clone.sh BLIP https://github.moeyy.xyz/https://github.com/salesforce/BLIP.git
48211a1594f1321b00f14c9f7a5b4813144b2fb9
RUN . /clone.sh k-diffusion https://github.moeyy.xyz/https://github.com/crowsonkb/k-diffusion.git
ab527a9a6d347f364e3d185ba6d714e22d80cb3c
RUN . /clone.sh clip-interrogator https://github.moeyy.xyz/https://github.com/pharmapsychotic/clip-
interrogator 2cf03aaf6e704197fd0dae7c7f96aa59cf1b11c9
RUN . /clone.sh generative-models https://github.moeyy.xyz/https://github.com/Stability-AI/generative-
models 45c443b316737a4ab6e40413d7794a7f5657c19f
RUN . /clone.sh stable-diffusion-webui-assets
https://github.moeyy.xyz/https://github.com/AUTOMATIC1111/stable-diffusion-webui-assets
6f7db241d2f8ba7457bac5ca9753331f0c266917



FROM pytorch/pytorch:2.3.0-cuda12.1-cudnn8-runtime

ENV DEBIAN_FRONTEND=noninteractive PIP_PREFER_BINARY=1

RUN --mount=type=cache,target=/var/cache/apt \
  apt-get update && \
  # we need those
  apt-get install -y fonts-dejavu-core rsync git jq moreutils aria2 \
  # extensions needs those
  ffmpeg libglfw3-dev libgles2-mesa-dev pkg-config libcairo2 libcairo2-dev build-essential

RUN pip install tb-nightly
RUN pip install --upgrade pip
RUN pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
ENV HF_ENDPOINT="https://hf-mirror.com"

WORKDIR /
RUN --mount=type=cache,target=/root/.cache/pip \
  git clone https://github.moeyy.xyz/https://github.com/AUTOMATIC1111/stable-diffusion-webui.git && \
  cd stable-diffusion-webui && \
  git reset --hard v1.9.4 && \
  pip install -r requirements_versions.txt
```

```
ENV ROOT=/stable-diffusion-webui

COPY --from=download /repositories/ ${ROOT}/repositories/
RUN mkdir ${ROOT}/interrogate && cp ${ROOT}/repositories/clip-interrogator/clip_interrogator/data/*
${ROOT}/interrogate

RUN --mount=type=cache,target=/root/.cache/pip \
  pip install pyngrok xformers==0.0.26.post1 \

git+https://github.moeyy.xyz/https://github.com/TencentARC/GFPGAN.git@8d2447a2d918f8eba5a4a01463fd48e45
126a379 \

git+https://github.moeyy.xyz/https://github.com/openai/CLIP.git@d50d76daa670286dd6cacf3bcd80b5e4823fc8e
1 \
  git+https://github.moeyy.xyz/https://github.com/mlfoundations/open_clip.git@v2.20.0

# there seems to be a memory leak (or maybe just memory not being freed fast enough) that is fixed by
this version of malloc
# maybe move this up to the dependencies list.
RUN apt-get -y install libgoogle-perftools-dev && apt-get clean
ENV LD_PRELOAD=libtcmalloc.so

COPY . /docker

RUN \
  # mv ${ROOT}/style.css ${ROOT}/user.css && \
  # one of the ugliest hacks I ever wrote \
  sed -i 's/in_app_dir = .*/in_app_dir = True/g' /opt/conda/lib/python3.10/site-
packages/gradio/routes.py && \
  git config --global --add safe.directory '*'

WORKDIR ${ROOT}
ENV NVIDIA_VISIBLE_DEVICES=all
ENV CLI_ARGS=""
EXPOSE 7860
ENTRYPOINT ["/docker/entrypoint.sh"]
CMD python -u webui.py --listen --port 7860 ${CLI_ARGS}
```

⚠ 注意：

- `RUN pip install tb-nightly` **命令应该位于** `RUN pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple` **之前。这是因为在将 pip 源更换为清华源之后安装 tb-nightly 会产生错误。因此，应首先使用原始的 pip 源安装 tb-nightly，然后再切换到清华源进行其他包的安装。**
- **设置** `ENV HF_ENDPOINT="https://hf-mirror.com"` **是为了从 Hugging Face 下载 Tokenizer。由于使用原网站可能导致下载失败，因此需要替换为镜像网站。**

3. **运行命令开始下载 UI 环境：**

```
docker compose --profile auto up --build
```

此时会经历较长时间的等待并下载环境，完成后可以看到如下内容（参考）：

```
...
auto-1  | Mounted .cache
auto-1  | Mounted config_states
auto-1  | mkdir: created directory '/stable-diffusion-webui/repositories/CodeFormer'
auto-1  | mkdir: created directory '/stable-diffusion-webui/repositories/CodeFormer/weights'
auto-1  | Mounted .cache
auto-1  | Mounted embeddings
auto-1  | Mounted config.json
auto-1  | Mounted models
auto-1  | Mounted styles.csv
auto-1  | Mounted ui-config.json
auto-1  | Mounted extensions
auto-1  | Installing extension dependencies (if any)
auto-1  | Calculating sha256 for /stable-diffusion-webui/models/Stable-diffusion/sd-v1-5-
inpainting.ckpt: Running on local URL:  http://0.0.0.0:7860
auto-1  |
auto-1  | To create a public link, set `share=True` in `launch()`.
auto-1  | Startup time: 5.0s (import torch: 1.8s, import gradio: 0.6s, setup paths: 0.9s, initialize
shared: 0.2s, other imports: 0.5s, load scripts: 0.3s, create ui: 0.2s, add APIs: 0.3s).
auto-1  | c6bbc15e3224e6973459ba78de4998b80b50112b0ae5b5c67113d56b4e366b19
auto-1  | Loading weights [c6bbc15e32] from /stable-diffusion-webui/models/Stable-diffusion/sd-v1-5-
inpainting.ckpt
auto-1  | Creating model from config: /stable-diffusion-webui/configs/v1-inpainting-inference.yaml
auto-1  | /opt/conda/lib/python3.10/site-packages/huggingface_hub/file_download.py:1150: FutureWarning:
`resume_download` is deprecated and will be removed in version 1.0.0. Downloads always resume when
possible. If you want to force a new download, use `force_download=True`.
auto-1  |   warnings.warn(
auto-1  | Applying attention optimization: xformers... done.
auto-1  | Model loaded in 4.9s (calculate hash: 3.2s, load weights from disk: 0.7s, create model: 0.2s,
apply weights to model: 0.3s, calculate empty prompt: 0.3s).
```

> **注意：**
> 运行命令下载 UI 环境可能会由于网络环境问题导致多次失败，但已经下载好的会缓存在容器中，重新输入命令会从上次中断的地方开始下载，所以请反复尝试直到下载成功为止。

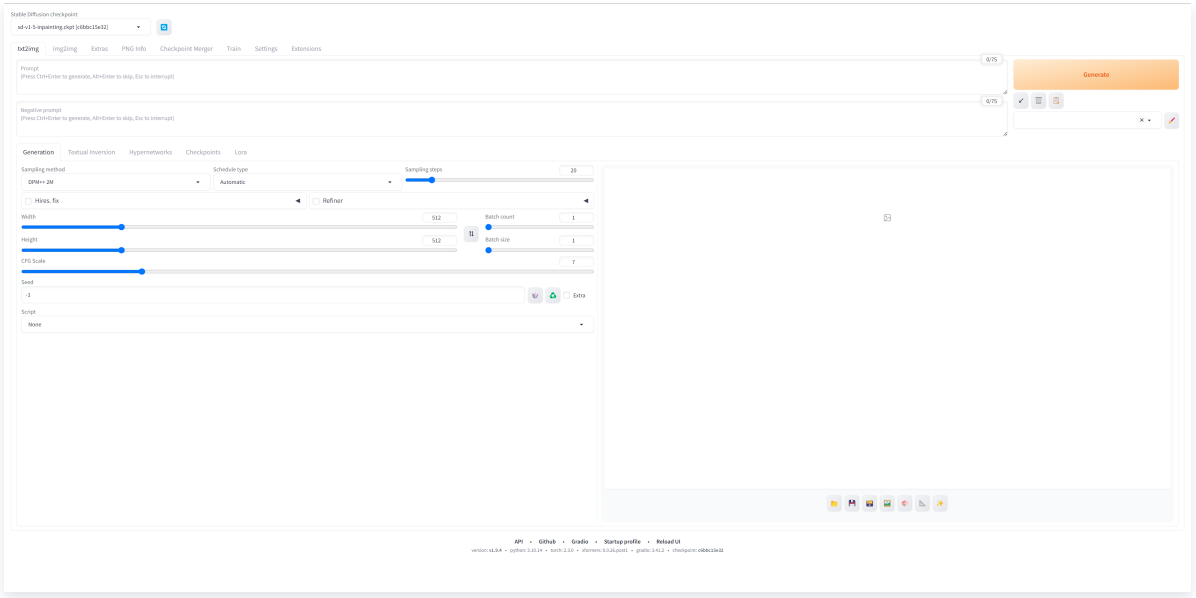
4. 同时，Auto 容器也已经被创建。新建一个窗口并输入以下命令：

```
docker ps
```

即可看到 auto UI的docker 容器已经被创建：

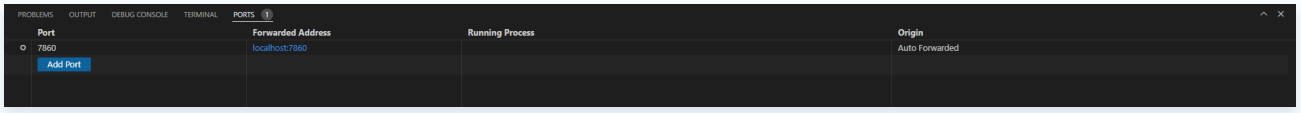
```
CONTAINER ID   IMAGE                                                COMMAND                  CREATED
STATUS                       PORTS                                      NAMES
93abc03d6600   sd-auto:78                                           "/docker/entrypoint.…"   2 days
ago       Exited (137) 21 minutes ago   0.0.0.0:7860->7860/tcp, :::7860->7860/tcp   webui-docker-auto-1
3f58c9a2903c   webui-docker-download                                "/docker/download.sh"    4 days
ago       Exited (137) 3 days ago                                                       webui-docker-
download-1
```

5. 本地打开浏览器（不是服务器端）输入网址 `http://`（服务器公网ip）`:7860/`，即可看到 auto UI的Stable Diffusion WebUI 界面：

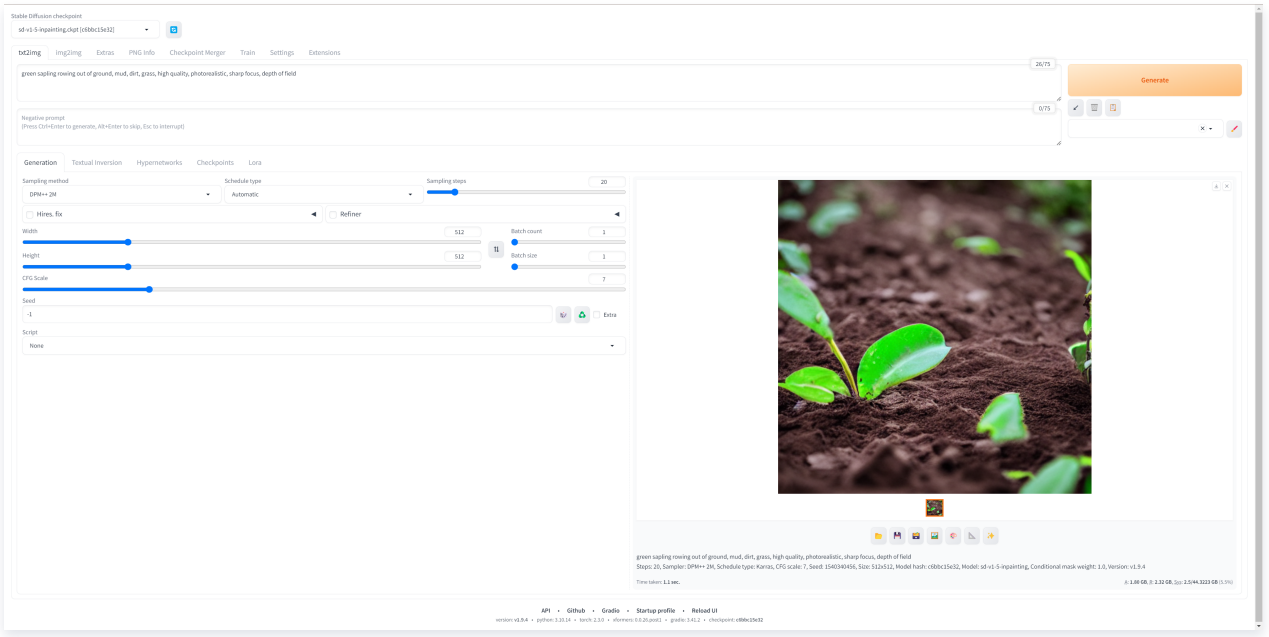


说明：

- 当使用公网 IP 访问服务器服务时，请确保将**7860**端口添加到安全组中以允许放行，从而使得**7860**端口上的服务可以通过公网被访问。
- 如果不想通过公网访问，想在本地浏览器里输入 `http://localhost:7860/` 直接访问 webui 界面，请通过 VSCode 连接上服务器之后，在终端的 PORTS 新添加**7860**端口服务，再打开浏览器即可看到 webui 服务（因为 VSCode 也有端口转发的功能）。



6. 这里我们演示如何使用 txt2img 生成图像。在提示栏中输入 `green sapling growing out of ground, mud, dirt, grass, high quality, photorealistic, sharp focus, depth of field`，单击右侧的**生成**，即可根据给定的提示生成图像：



生成的图像如下：

7. 生成的图像同时也会保存在 stable-diffusion-webui-docker/output/txt2img/ 当前日期 `/00000-1540340456.png` 目录下，同时窗口会显示如下信息：

```
100% 20/20 [00:00<00:00, 20.90it/s]
Total progress: 100% 20/20 [00:00<00:00, 22.10it/s]
```

表示生成图像运行成功。

comfy UI

1. 此 UI 界面源自 GitHub comfyanonymous/ComfyUI，首先需要安装各种环境。
2. 打开 stable-diffusion-webui-docker/services/comfy/Dockerfile，可以看到该文件包含一条 GitHub 链接。为了加快下载速度，可以在该 GitHub 链接前加上 GitHub 镜像网站地址，将其转换为国内镜像网站下载。此外，由于 Dockerfile 中也包含 `pip install -r requirements_versions.txt` 命令，使用 pip 安装各种包可能会很慢，因此需要将其替换为国内镜像源。经过修改，整个 Dockerfile 文件如下所示：

```
FROM pytorch/pytorch:2.3.0-cuda12.1-cudnn8-runtime


ENV DEBIAN_FRONTEND=noninteractive PIP_PREFER_BINARY=1


RUN apt-get update && apt-get install -y git && apt-get clean


RUN pip install --upgrade pip
RUN pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple


ENV ROOT=/stable-diffusion
RUN --mount=type=cache,target=/root/.cache/pip \
  git clone https://github.moeyy.xyz/https://github.com/comfyanonymous/ComfyUI.git ${ROOT} && \
  cd ${ROOT} && \
  git checkout master && \
  git reset --hard 276f8fce9f5a80b500947fb5745a4dde9e84622d && \
  pip install -r requirements.txt


WORKDIR ${ROOT}
COPY . /docker/
RUN chmod u+x /docker/entrypoint.sh && cp /docker/extra_model_paths.yaml ${ROOT}
```

```
ENV NVIDIA_VISIBLE_DEVICES=all PYTHONPATH="${PYTHONPATH}:${PWD}" CLI_ARGS=""
EXPOSE 7860
ENTRYPOINT ["/docker/entrypoint.sh"]
CMD python -u main.py --listen --port 7860 ${CLI_ARGS}
```

3. 运行命令开始下载 UI 环境：

```
docker compose --profile comfy up --build
```

此时同样会经历一定的等待时间并下载环境，完成后可以看到如下内容（参考）：

```
...
Attaching to comfy-1
comfy-1  | mkdir: created directory '/data/config/comfy'
comfy-1  | mkdir: created directory '/data/config/comfy/custom_nodes'
comfy-1  | Mounted .cache
comfy-1  | mkdir: created directory '/output/comfy'
comfy-1  | Mounted comfy
comfy-1  | mkdir: created directory '/data/config/comfy/input'
comfy-1  | Mounted input
comfy-1  | Total VRAM 45386 MB, total RAM 189442 MB
comfy-1  | pytorch version: 2.3.0
comfy-1  | Set vram state to: NORMAL_VRAM
comfy-1  | Device: cuda:0 NVIDIA L40 : cudaMallocAsync
comfy-1  | VAE dtype: torch.bfloat16
comfy-1  | Using pytorch cross attention
comfy-1  | ****** User settings have been changed to be stored on the server instead of browser
storage. ******
comfy-1  | ****** For multi-user setups add the --multi-user CLI argument to enable multiple user
profiles. ******
comfy-1  | Adding extra search path checkpoints /data/models/Stable-diffusion
comfy-1  | Adding extra search path configs /data/models/Stable-diffusion
comfy-1  | Adding extra search path vae /data/models/VAE
comfy-1  | Adding extra search path loras /data/models/Lora
comfy-1  | Adding extra search path upscale_models /data/models/RealESRGAN
comfy-1  | Adding extra search path upscale_models /data/models/ESRGAN
comfy-1  | Adding extra search path upscale_models /data/models/SwinIR
comfy-1  | Adding extra search path upscale_models /data/models/GFPGAN
comfy-1  | Adding extra search path hypernetworks /data/models/hypernetworks
comfy-1  | Adding extra search path controlnet /data/models/ControlNet
comfy-1  | Adding extra search path gligen /data/models/GLIGEN
comfy-1  | Adding extra search path clip /data/models/CLIPEncoder
comfy-1  | Adding extra search path embeddings /data/embeddings
comfy-1  | Adding extra search path custom_nodes /data/config/comfy/custom_nodes
comfy-1  |
comfy-1  | Import times for custom nodes:
comfy-1  |    0.0 seconds: /stable-diffusion/custom_nodes/websocket_image_save.py
comfy-1  |
comfy-1  | Starting server
comfy-1  |
comfy-1  | To see the GUI go to: http://0.0.0.0:7860
```

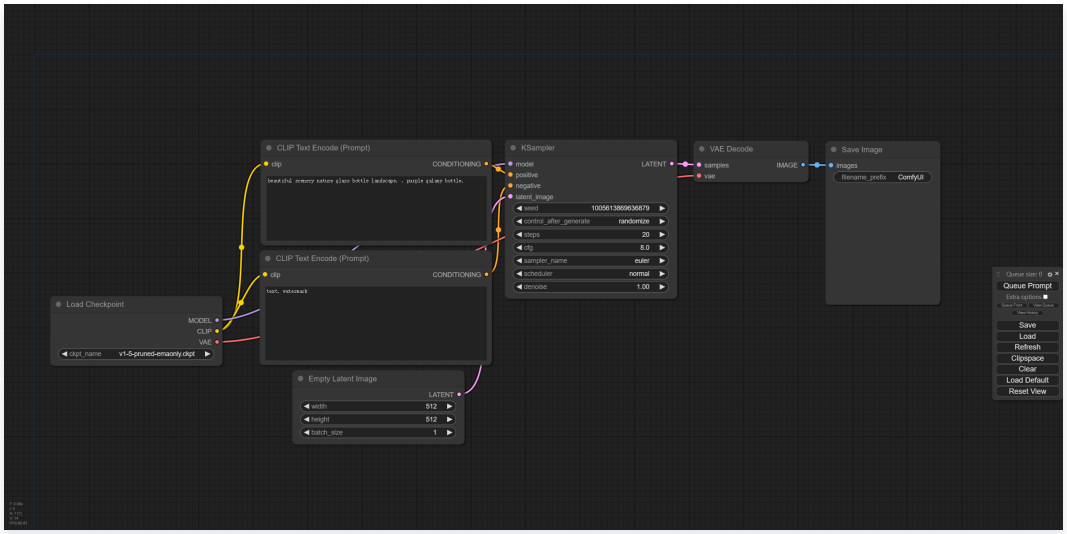
4. 表明 UI 环境安装成功。同时 comfy 容器也已经被创建，新建一个窗口输入以下命令：

```
docker ps
```

即可看到 comfy UI的docker 容器已经被创建：

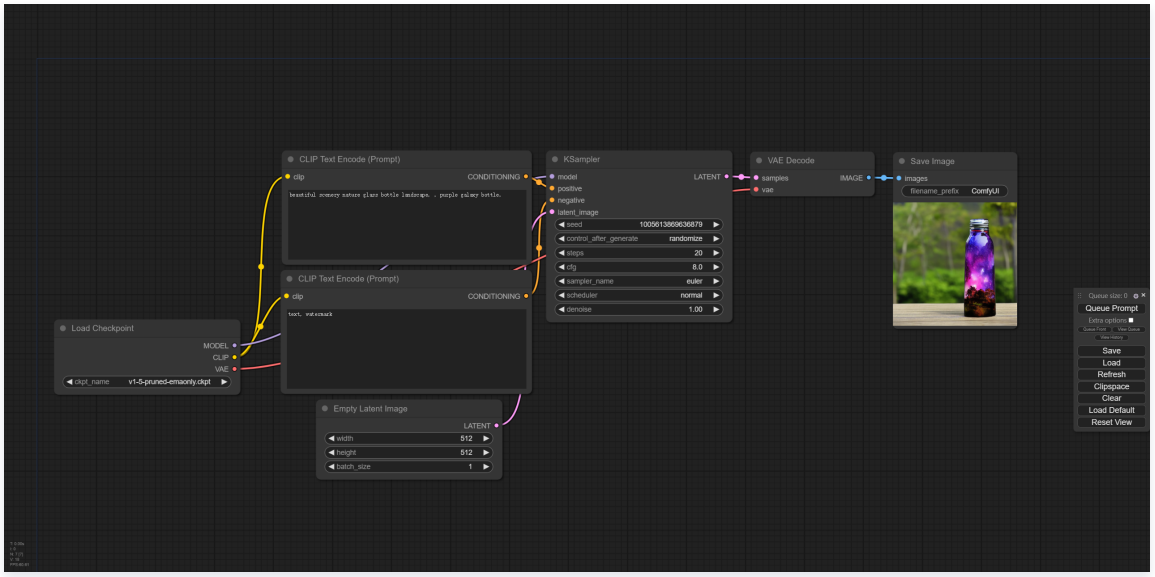
```
CONTAINER ID   IMAGE                                                COMMAND                  CREATED
STATUS                       PORTS                                                NAMES
16cd64facfba   sd-comfy:7                                           "/docker/entrypoint.…"   10
minutes ago   Up 10 minutes                0.0.0.0:7860->7860/tcp, :::7860->7860/tcp   webui-docker-
comfy-1
93abc03d6600   sd-auto:78                                           "/docker/entrypoint.…"   2 days
ago       Exited (137) 21 minutes ago                                                   webui-docker-auto-1
3f58c9a2903c   webui-docker-download                                "/docker/download.sh"    4 days
ago       Exited (137) 3 days ago                                                       webui-docker-
download-1
```

5. 本地打开浏览器输入网址 `http://（服务器公网ip）:7860/`，即可看到 comfy UI的Stable Diffusion WebUI 界面：



> **说明：**
> 与 auto UI 类似，如果您想通过 localhost 访问，请通过 VSCode 的 PORTS 功能添加7860端口进行端口转发服务，然后在本地运行 webui 服务。

6. 该 UI 界面的优势在于能够在生成图像时实时观察前向传播信号在各个模型模块之间的流动过程。单击右侧的 Queue Prompt，即可根据所提供的提示生成图像。



生成的图像如下：

7. 生成的图像同时也会保存在 stable-diffusion-webui-docker/output/comfy/ComfyUI_00001_.png 目录下，同时窗口会显示如下信息：

```
comfy-1  | got prompt
comfy-1  | model_type EPS
comfy-1  | Using pytorch attention in VAE
comfy-1  | Using pytorch attention in VAE
comfy-1  | Requested to load SD1ClipModel
comfy-1  | Loading 1 new model
comfy-1  | Requested to load BaseModel
comfy-1  | Loading 1 new model
100% 20/20 [00:00<00:00, 24.65it/s]
comfy-1  | Requested to load AutoencoderKL
comfy-1  | Loading 1 new model
comfy-1  | Prompt executed in 2.72 seconds
```

表示生成图像运行成功。

注意事项

> **说明：**
> 由于 OpenCloudOS 是 TencentOS Server 的开源版本，理论上上述文档当中的所有操作同样适用于 OpenCloudOS。

参考文档

- GitHub AbdBarho/stable-diffusion-webui-docker
- GitHub AbdBarho/stable-diffusion-webui-docker 安装指引
- GitHub AbdBarho/stable-diffusion-webui-docker 用法
- GitHub AUTOMATIC1111/stable-diffusion-webui
- GitHub comfyanonymous/ComfyUI
- GItHub 镜像加速
- Dockerfile修改 pip 镜像源
- GitHub gfpgan 安装错误指引

Ollama & Open WebUI

最近更新时间：2024-08-13 09:55:21

本指导适用于在 TencentOS Server 3 上运行 Ollama & Open WebUI 应用框架，以 Docker 方式启动。

Ollama 环境准备

本指导通过直接拉取 Dockerhub 上 Ollama 的镜像包运行 Ollama 应用框架。
由于直接从 DockerHub Ollama 使用 Docker pull 拉取镜像会由于访问限制问题拉取失败，这里我们通过腾讯云 Docker 镜像源加速镜像下载。

配置腾讯云镜像加速源

1. 打开系统中 `/etc/docker/daemon.json` 配置文件：

```
vim /etc/docker/daemon.json
```

2. 按 `i` 切换至编辑模式，添加以下内容，请注意 `json` 代码的格式：

```
{
   "registry-mirrors": [
   "https://mirror.ccs.tencentyun.com"
  ]
}
```

添加完成后格式应类似于：

```
{
    "runtimes": {
        "nvidia": {
            "args": [],
            "path": "nvidia-container-runtime"
        }
    },
    "registry-mirrors": [
    "https://mirror.ccs.tencentyun.com"
  ]
}
```

3. 随后按 `ESC` 即可退出编辑模式，再输入 `!wq` 即可保存并退出 vim 环境。
4. 此时，Docker daemon 将被关闭，需要重启 Docker。

```
sudo systemctl restart docker
```

> ⚠ **注意：**
> 如果此时出现以下错误：
> - Job for docker.service failed because the control process exited with error code.
> - See "systemctl status docker.service" and "journalctl -xe" for details.
>
> 说明添加的 `registry-mirrors` 没有按照 `json` 文件的格式添加，请重新检查格式是否正确，添加正确后请再次重启 Docker。

5. 查看 Docker 状态：

```
sudo docker info
```

此时 Clinet 和 Server 都正确运行，且出现 Registry Mirrors 腾讯云镜像源内容，则说明配置成功。

```
Insecure Registries:
 127.0.0.0/8
Registry Mirrors:
 https://mirror.ccs.tencentyun.com/
Live Restore Enabled: false
```

从 Dockerhub 拉取 Ollama 镜像

1. 先创建服务器本地挂在模型权重的地址：

```
mkdir ollama
cd ollama
```

2. 根据 DockerHub上Ollama 的指引，拉取使用 Nvidia GPU 的 Ollama 镜像：

```
docker run -d --gpus=all -v $PWD:/root/.ollama -p 11434:11434 --name ollama ollama/ollama
```

3. 由于使用了腾讯云镜像源，这里拉取镜像的速度会很快，镜像拉取成功之后，进入容器内部：

```
docker exec -it ollama bash
```

4. 在容器内部输入 `ollama` 相关命令，可以查看 ollama 的使用说明：

```
ollama

#输出:
--------------------------------------------------------------------
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
  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.
--------------------------------------------------------------------

#查看ollama版本
ollama -v

#输出（参考）:
ollama version is 0.3.0
```

运行模型

Ollama 模型运行

1. 在容器内部输入命令运行模型，这里我们运行 llama3.1-8b 模型，更多的模型可以参见 Ollama 模型库 。

```
ollama run llama3.1
```

2. 下载模型完成后，屏幕将显示如下：

```
pulling manifest
pulling 87048bcd5521... 100%
 ▕▏
▕▏ 4.7 GB
pulling 8cf247399e57... 100%
 ▕▏
▕▏ 1.7 KB
pulling f1cd752815fc... 100%
 ▕▏
▕▏  12 KB
pulling 56bb8bd477a5... 100%
 ▕▏
▕▏   96 B
pulling e711233e7343... 100%
 ▕▏
▕▏  485 B
verifying sha256 digest
writing manifest
removing any unused layers
success
```

此时就可以在命令行与模型对话：

```
>>> 你好
您好！我很高兴与您交谈。有何问题或需要讨论的内容吗？
```

3. 可以看到模型已成功运行并给出了回复。同时，在 root/.ollama/history 目录下可以找到之前输入的 `prompt` 指令。

Open WebUI

Open WebUI 是 Ollama 的可视化网页界面，通过 Open WebUI 可以更方便的与运行模型，与模型对话等。

1. 首先在服务器本地创建 Open WebUI 的挂载文件夹。

```
mkdir open-webui
cd open-webui
```

2. 接下来，拉取 Open WebUI 的镜像：

```
docker run -d -p 3000:8080 --add-host=host.docker.internal:host-gateway -v $PWD:/app/backend/data -e
HF_ENDPOINT="https://hf-mirror.com" --name open-webui --restart always ghcr.io/open-webui/open-
webui:main
```

3. 镜像拉取成功后，Open WebUI 容器即在后台运行，输入以下命令可以查看运行动态日志：

```
docker logs -f open-webui
```

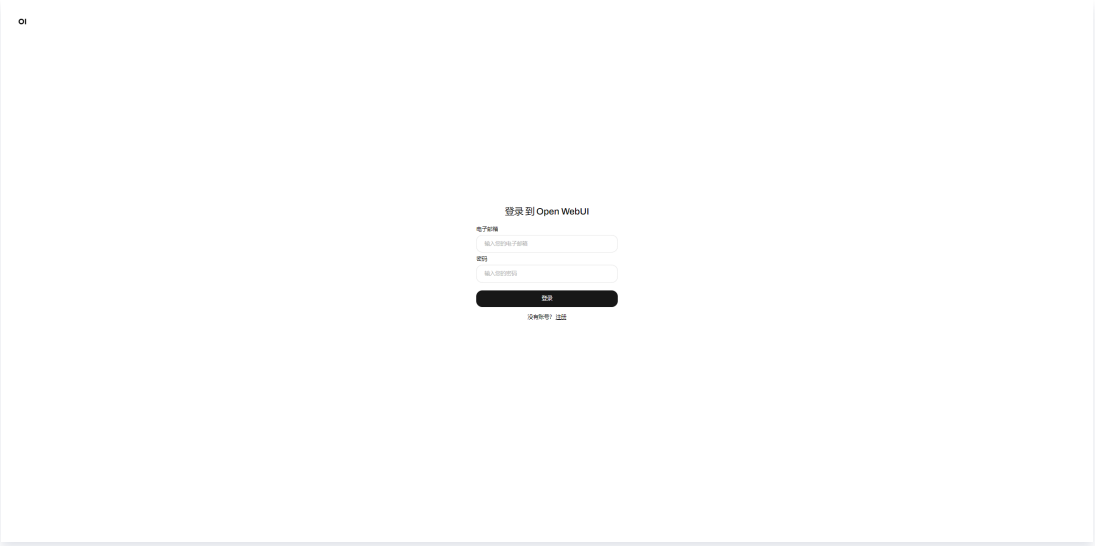
> ⚠ 注意：

> 下载镜像的地址为 `ghcr.io/open-webui/open-webui:main`，由于 `ghcr.io` 国内可替代的镜像源几乎没有，请耐心等待下载。如出现 `docker: unexpected EOF.` 等错误则是由于下载太慢导致，反复尝试下载即可。如实在下载失败次数太多，可以将 `ghcr.io` 改为 `ghcr.chenby.cn` 或 `ghcr.nju.edu.cn`，可以加快下载速度（但不保证将来此镜像源还可以使用，仅供参考）。

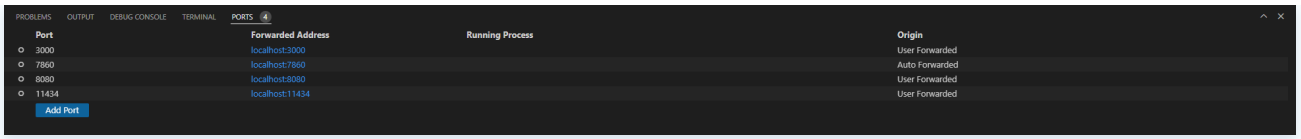
4. 通过查看运行日志，如果看到如下内容，则说明 Open WebUI 后台运行成功：

```
...
INFO:     Started server process [1]
INFO:     Waiting for application startup.
/app

  ___                    __        __   _     _   _ ___
 / _ \ _ __   ___ _ __   \ \      / /__| |__ | | | |_ _|
| | | | '_ \ / _ \ '_ \   \ \ /\ / / _ \ '_ \| | | || |
| |_| | |_) |  __/ | | |   \ V  V /  __/ |_) | |_| || |
 \___/| .__/ \___|_| |_|    \_/\_/ \___|_.__/ \___/|___|
      |_|


v0.3.10 - building the best open-source AI user interface.



INFO  [alembic.runtime.migration] Context impl SQLiteImpl.
INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
INFO  [alembic.runtime.migration] Running upgrade  -> 7e5b5dc7342b, init
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
...
```

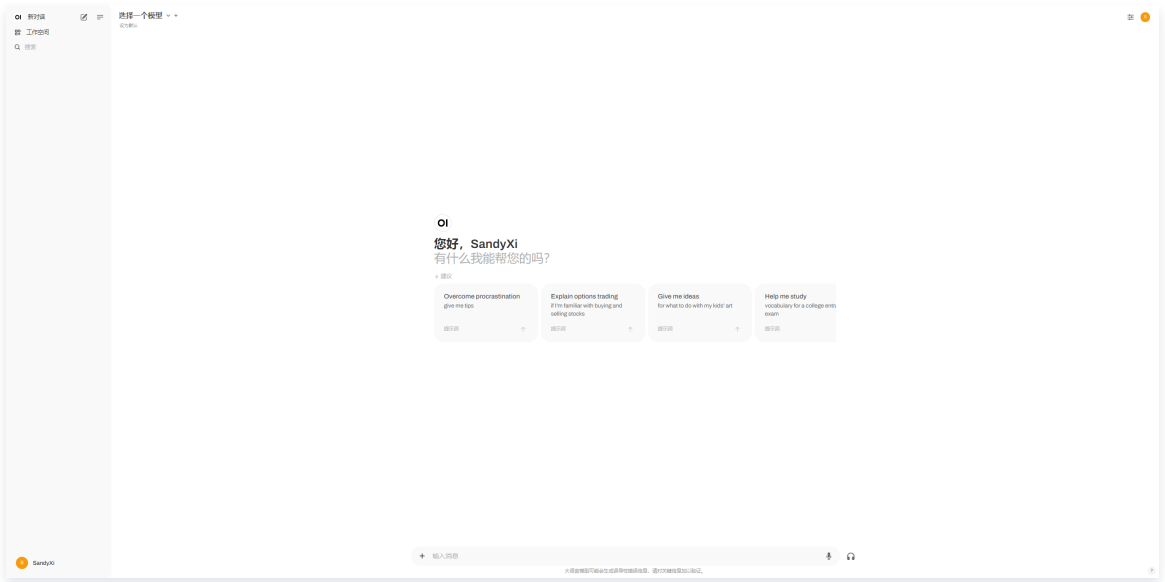
5. 此时在本地浏览器（非服务器端）里输入 `http://`（服务器公网ip）`:3000/`，即可看到 Open Webui 服务的内容。



> **说明：**
>
> - 当使用公网 IP 访问服务器服务时，请确保将服务器的3000端口添加到安全组中以允许放行，从而使得通过公网可以访问该端口的服务。
> - 如果不想通过公网访问，想在本地浏览器里输入 `http://localhost:3000/` 直接访问 webui 界面，请通过 VSCode 连接上服务器之后，在终端的 PORTS 新添加3000端口服务，再打开浏览器即可看到 webui 服务（因为 VSCode 也有端口转发的功能）。



6. 如图所示，首次进入时需要创建账号。单击**注册**以创建账号，创建成功后即可进入主界面。



7. 单击选择一个模型，您将看到之前下载的 Ollama LLaMA 3.1-8b 模型。选择该模型以开始对话：

有正常返回，则 Ollama Open WebUI 运行成功。

> **注意：**
> 如果 Open WebUI 无法使用 Ollama 的模型，则说明 Open WebUI 镜像没有检测到 Ollama 的服务端口，在运行 Open WebUI 的容器时加上 `-e OLLAMA_BASE_URL=http://localhost:11434` 即可。

注意事项

> **说明：**
> 由于 OpenCloudOS 是 TencentOS Server 的开源版本，理论上上述文档当中的所有操作同样适用于 OpenCloudOS。

参考文档

- 腾讯云镜像源

- GitHub Ollama
- Ollama 模型库
- GitHub open-webui
- Open Webui 使用教程
- Open WebUI 报错指引

ChatTTS & WebUI

最近更新时间：2024-08-13 09:55:21

本指导适用于在 TencentOS Server 3 上运行 ChatTTS & WebUI 应用框架，以 Docker 方式启动。

ChatTTS 环境准备

由于 ChatTTS 框架较新，目前还没有官方的 Docker 镜像。根据 ChatTTS 官方文档，可以通过 conda 来运行，因此我们将使用 Nvidia 的 PyTorch 镜像作为基础环境，该环境已经包含了运行 ChatTTS 所需的许多基础包。

1. 首先从 GitHub 下载 ChatTTS 开源仓库到本地。

```
git clone https://github.com/2noise/ChatTTS
cd ChatTTS
```

2. 启动 ChatTTS 容器。

```
docker run -it --gpus all -p 8080:8080 --name=chattts -e HF_ENDPOINT="https://hf-mirror.com" -v
$PWD:/workspace nvcr.io/nvidia/pytorch:23.06-py3 /bin/bash
```

3. 在运行命令后，系统将开始下载 PyTorch 镜像。请确保您的网络环境良好，以便成功下载镜像的所有层。下载成功后，系统将自动进入容器内部。

安装必要的包

此时应在 /workspace 目录下，我们需要安装运行模型所需要的包。

1. 将 pip 换为国内清华源以加快下载速度。

```
#将pip换成清华源
#设为默认，永久有效
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

2. 确保使用较新的 pip 包，并使用命令更新。

```
#更新pip包
python3 -m pip install --upgrade pip
```

3. 安装必要的包。

```
pip install --upgrade -r requirements.txt
```

此时下载包的速度应该较快，请耐心等待下载结束。

运行模型

webui

1. ChatTTS 提供了 webui 服务以便可视化的运行模型，运行以下命令来启动 webui 服务：

```
python examples/web/webui.py
```

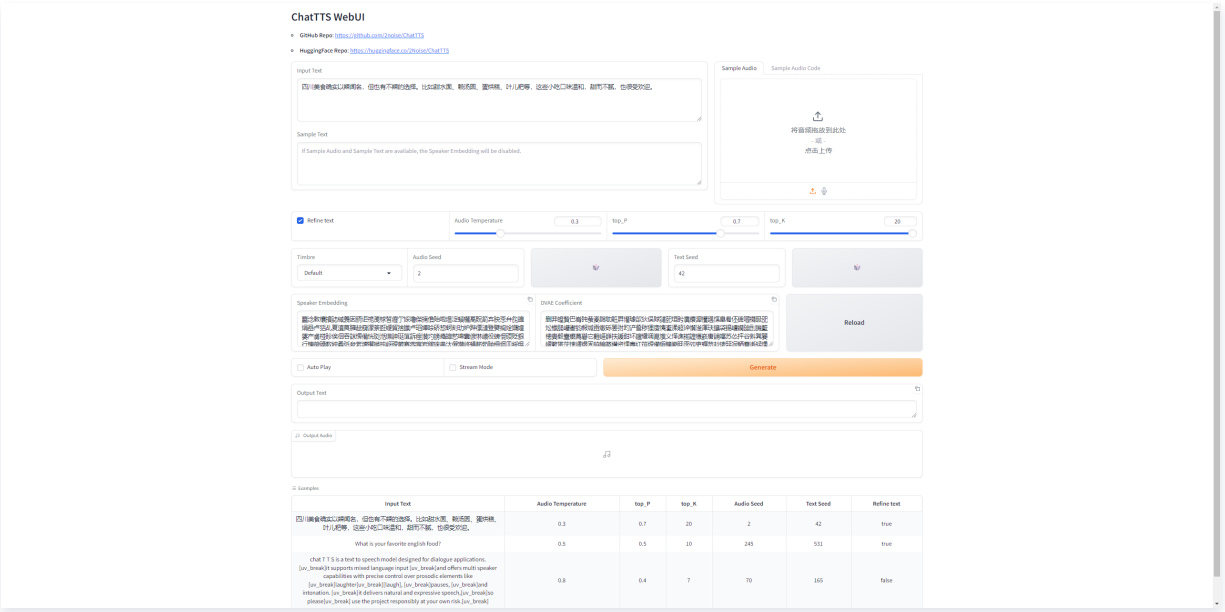
此时会开始下载模型，请耐心等待。

2. 模型下载完成后会启动 webui 服务，此时可以看到以下内容：

```
...
[+0000 20240801 04:43:37] [INFO] ChatTTS | core | tokenizer loaded.
[+0000 20240801 04:43:37] [INFO] ChatTTS | core | all models has been initialized.
 NeMo-text-processing :: INFO     :: Creating ClassifyFst grammars.
```

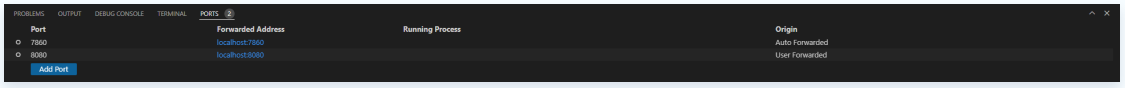
```
2024-08-01 04:43:57,179 WETEXT INFO found existing fst: /usr/local/lib/python3.10/dist-
packages/tn/zh_tn_tagger.fst
2024-08-01 04:43:57,179 WETEXT INFO                    /usr/local/lib/python3.10/dist-
packages/tn/zh_tn_verbalizer.fst
2024-08-01 04:43:57,179 WETEXT INFO skip building fst for zh_normalizer ...
[+0000 20240801 04:43:57] [INFO]  WebUI  | webui | Models loaded successfully.
Running on local URL:  http://0.0.0.0:8080
```

3. 此时，在本地浏览器（非服务器端）中输入 `http://`（服务器公网IP）`:8080/`，即可访问 ChatTTS Web UI 服务的界面。



说明：

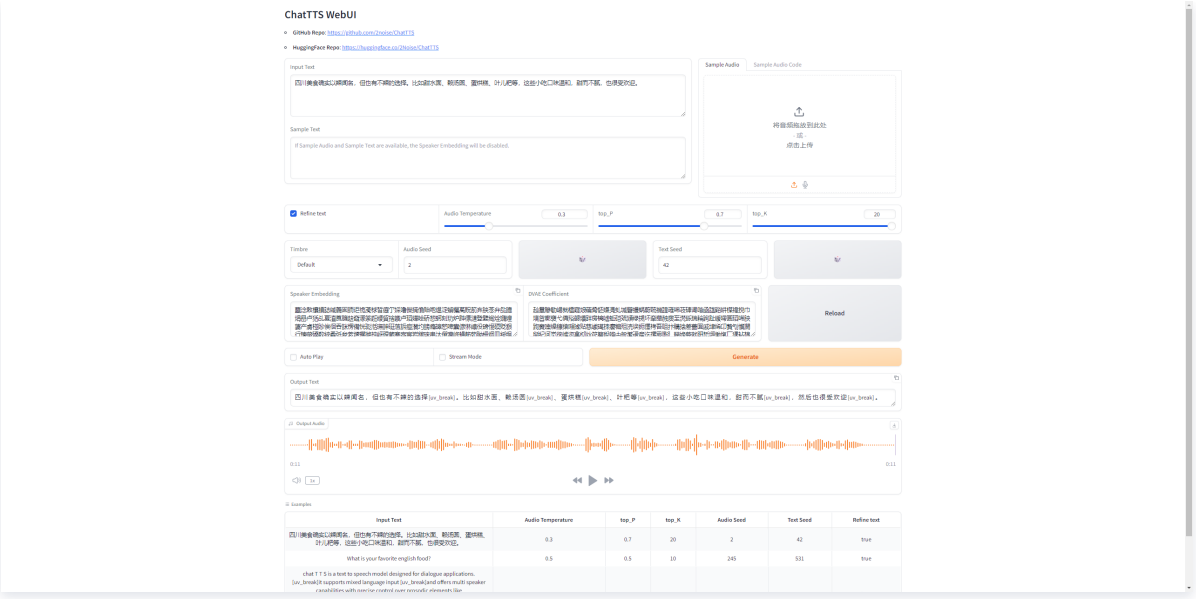
- 当使用公网 IP 访问服务器服务时，请确保将服务器的**8080**端口添加到安全组中并放行，以便通过公网访问**8080**端口的服务。
- 如果不想通过公网访问，想在本地浏览器里输入 `http://localhost:8080/` 直接访问 webui 界面，请通过 VSCode 连接上服务器之后，在终端的 PORTS 新添加**8080**端口服务，再打开浏览器即可看到 webui 服务（因为 VSCode 也有端口转发的功能）。



4. 这里我们以网站给的例子：

四川美食确实以辣闻名，但也有不辣的选择。例如甜水面、赖汤圆、蛋烘糕、叶儿粑等，这些小吃口味温和，甜而不腻，也很受欢迎。

将这段话转为语音，单击 **Generate** 后耐心等待，生成语音结束后可以看到输出文本的语调变化以及语音的波形图。



5. 单击播放即可听到生成的语音。在容器里的 tmp/gradio 目录下也可以看到输出的语音文件。

命令行运行

1. 通过命令行也可以直接运行，输入以下命令：

```
python examples/cmd/run.py "ChatTTS是最好的文字转语音框架" "腾讯是全球最好的互联网企业之一"
```

2. 以下是文字转语音的两句话。输入后，模型将开始进行文字到语音的转换。运行结束后，您将看到以下内容：

```
...
[+0000 20240801 08:00:40] [INFO] Command | run | Inference completed.
[+0000 20240801 08:00:41] [INFO] Command | run | Audio saved to output_audio_0.mp3
[+0000 20240801 08:00:41] [INFO] Command | run | Audio saved to output_audio_1.mp3
[+0000 20240801 08:00:41] [INFO] Command | run | Audio generation successful.
[+0000 20240801 08:00:41] [INFO] Command | run | ChatTTS process finished.
```

3. 在 ChatTTS 文件夹下，可以看到两个文件：`output_audio_0.mp3` 和 `output_audio_1.mp3`，表示输出成功。

Python 文件运行

1. 首先安装 ChatTTS 库。

```
pip install ChatTTS
```

2. 接着，创建一个名为 `Demo` 的文件夹，在该文件夹中创建一个名为 `basic_usage.py` 的文件，并输入相应的代码。

```
mkdir Demo
cd Demo
```

basic_usage.py 输入以下代码：

```
import ChatTTS
import torch
import torchaudio


chat = ChatTTS.Chat()
chat.load(compile=False) # Set to True for better performance
```

```
texts = ["上海拥有东方明珠，外滩，豫园等知名景点", "Tencent OS server is a stable and secure enterprise-class Linux"]

wavs = chat.infer(texts)

for i in range(len(wavs)):
    torchaudio.save(f"basic_output{i}.wav", torch.from_numpy(wavs[i]), 24000)
```

3. 请运行以下代码：

```
python Demo/basic_usage.py
```

4. 运行完成后，您可以在 ChatTTS 文件夹中看到输出的语音文件 `basic_output0.wav` 和 `basic_output1.wav`，这表明模型已成功运行。

注意事项

> **说明：**
> 由于 OpenCloudOS 是 TencentOS Server 的开源版本，理论上上述文档当中的所有操作同样适用于 OpenCloudOS。

参考文档

- GitHub ChatTTS
- 清华 pipy 镜像源

TencentOS Server 3 运行主要深度学习训练框架及热门模型示例

TencentOS Server 3 运行环境及目录

最近更新时间：2024-08-28 17:44:21

本指导适用于在 TencentOS Server 3 上以 Docker 的方式运行主要深度学习训练框架及热门模型。

硬软件环境

Description：TencentOS Server release 3.1 (Final)。
Architecture：x86_64。
CPU op-mode(s)：32-bit, 64-bit。
CUDA Version：12.2。
GPU：NVIDIA L40。
操作系统详情：

```
[root@VM-1-5-tencentos ~]#  lsb_release -a
LSB Version:    :core-4.1-amd64:core-4.1-noarch
Distributor ID: TencentOSServer
Description:    TencentOS Server release 3.1 (Final)
Release:        3.1
Codename:       Final
```

CPU 详情：

```
[root@VM-1-5-tencentos ~]#  lsb_release -a
LSB Version:    :core-4.1-amd64:core-4.1-noarch
Distributor ID: TencentOSServer
Description:    TencentOS Server release 3.1 (Final)
Release:        3.1
Codename:       Final
[root@VM-1-5-tencentos ~]# lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                384
On-line CPU(s) list:   0-383
Thread(s) per core:    2
Core(s) per socket:    96
Socket(s):             2
NUMA node(s):          2
Vendor ID:             AuthenticAMD
BIOS Vendor ID:        Red Hat
CPU family:            25
Model:                 17
Model name:            AMD EPYC 9K84 96-Core Processor
BIOS Model name:       3.0
Stepping:              0
CPU MHz:               2600.050
BogoMIPS:              5200.10
Hypervisor vendor:     KVM
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              32768K
NUMA node0 CPU(s):     0-191
NUMA node1 CPU(s):     192-383
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm constant_tsc rep_good nopl nonstop_tsc cpuid extd_apicid amd_dcm tsc_known_freq pni pclmulqdq monitor ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm cmp_legacy cr8_legacy abm sse4a misalignsse 3dnowprefetch osvw topoext perfctr_core invpcid_single ibpb vmmcall fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid avx512f avx512dq rdseed adx smap avx512ifma clflushopt clwb avx512cd sha_ni avx512bw avx512vl xsaveopt xsavec xgetbv1 avx512_bf16 clzero xsaveerptr wbnoinvd arat avx512vbmi umip avx512_vbmi2 vaes vpclmulqdq avx512_vnni avx512_bitalg avx512_vpopcntdq rdpid fsrm
```

GPU 详情：

```
[root@VM-1-5-tencentos ~]# nvidia-smi
Fri Aug  9 17:58:34 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.161.07             Driver Version: 535.161.07   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| 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 L40                     On  | 00000000:23:00.0 Off |                    0 |
| N/A   24C    P8              23W / 300W |      0MiB / 46068MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   1  NVIDIA L40                     On  | 00000000:33:00.0 Off |                    0 |
| N/A   26C    P8              23W / 300W |      0MiB / 46068MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   2  NVIDIA L40                     On  | 00000000:34:00.0 Off |                    0 |
| N/A   25C    P8              23W / 300W |      0MiB / 46068MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   3  NVIDIA L40                     On  | 00000000:43:00.0 Off |                    0 |
| N/A   27C    P8              35W / 300W |      0MiB / 46068MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   4  NVIDIA L40                     On  | 00000000:63:00.0 Off |                    0 |
| N/A   25C    P8              24W / 300W |      0MiB / 46068MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   5  NVIDIA L40                     On  | 00000000:83:00.0 Off |                    0 |
| N/A   26C    P8              23W / 300W |      0MiB / 46068MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   6  NVIDIA L40                     On  | 00000000:84:00.0 Off |                    0 |
| N/A   26C    P8              23W / 300W |      0MiB / 46068MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   7  NVIDIA L40                     On  | 00000000:A3:00.0 Off |                    0 |
| N/A   25C    P8              28W / 300W |      0MiB / 46068MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
```

PCI 总线情况：

```
[root@VM-1-5-tencentos ~]# lspci -tv
-+-[0000:a0]---00.0-[a1-a3]----00.0-[a2-a3]----00.0-[a3]----00.0  NVIDIA Corporation AD102GL [L40]
 +-[0000:80]---00.0-[81-84]----00.0-[82-84]--+-00.0-[83]----00.0  NVIDIA Corporation AD102GL [L40]
 |                                           \-01.0-[84]----00.0  NVIDIA Corporation AD102GL [L40]
 +-[0000:60]---00.0-[61-63]----00.0-[62-63]----00.0-[63]----00.0  NVIDIA Corporation AD102GL [L40]
 +-[0000:40]---00.0-[41-43]----00.0-[42-43]----00.0-[43]----00.0  NVIDIA Corporation AD102GL [L40]
 +-[0000:30]---00.0-[31-34]----00.0-[32-34]--+-00.0-[33]----00.0  NVIDIA Corporation AD102GL [L40]
 |                                           \-01.0-[34]----00.0  NVIDIA Corporation AD102GL [L40]
 +-[0000:20]---00.0-[21-23]----00.0-[22-23]----00.0-[23]----00.0  NVIDIA Corporation AD102GL [L40]
 \-[0000:00]-+-00.0  Intel Corporation 82G33/G31/P35/P31 Express DRAM Controller
             +-01.0-[01-02]----00.0-[02]--+-01.0  Cirrus Logic GD 5446
             |                            +-02.0  Red Hat, Inc. Virtio network device
             |                            +-03.0  Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) High Definition Audio Controller
             |                            +-04.0  NEC Corporation uPD720200 USB 3.0 Host Controller
             |                            +-05.0  Red Hat, Inc. Virtio block device
             |                            \-06.0  Red Hat, Inc. Virtio memory balloon
             +-02.0-[03-04]----00.0-[04]--
             +-03.0-[05-06]----00.0-[06]--
             +-04.0  Red Hat, Inc. QEMU PCIe Expander bridge
             +-05.0  Red Hat, Inc. QEMU PCIe Expander bridge
             +-06.0  Red Hat, Inc. QEMU PCIe Expander bridge
             +-07.0  Red Hat, Inc. QEMU PCIe Expander bridge
             +-08.0  Red Hat, Inc. QEMU PCIe Expander bridge
             +-09.0  Red Hat, Inc. QEMU PCIe Expander bridge
             +-1f.0  Intel Corporation 82801IB (ICH9) LPC Interface Controller
             +-1f.2  Intel Corporation 82801IR/IO/IH (ICH9R/DO/DH) 6 port SATA Controller [AHCI mode]
             \-1f.3  Intel Corporation 82801I (ICH9 Family) SMBus Controller
```

指南目录

环境准备（必做）

- 环境准备

DataParallel（DP）

- ResNet
- ViT

Distributed DataParallel（DDP）

- ResNet

- ViT

DeepSpeed

- ResNet
- Bert

Megatron

- GPT

环境准备

最近更新时间：2024-08-28 17:44:21

Docker 环境准备

安装 Docker

若有旧版本，需要删除旧版本，不然会有冲突。

```
yum remove docker* -y
```

安装 `tlinux-release-docker-ce` 包后，会默认使用 http://mirrors.tencent.com/docker-ce/ 目录。

```
#tlinux3.x安装docker-ce repo文件
yum install tencentos-release-docker-ce -y
yum install docker-ce -y
```

启动 Docker daemon

> ⚠ 注意：
> - 为了防止后续运行容器时出现如下错误：ERROR: Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?
> - 该错误是由于 Docker daemon 没有正确启动导致的，需要先开启 Docker daemon。

```
#开启Docker daemon (server)
sudo systemctl start docker

#使用以下命令检查Docker daemon是否正在运行，此时应该看到Docker Client和Docker Server都在运行中
sudo systemctl status docker
```

安装 NVIDIA Container Toolkit

在容器里运行模型，如果想要在 GPU 上运行，则需要安装 NVIDIA Container Toolkit，详情请参见：NVIDIA Container Toolkit 安装指引 。

```
#配置安装所需要的库
curl -s -L https://nvidia.github.io/libnvidia-container/stable/rpm/nvidia-container-toolkit.repo | \
sudo tee /etc/yum.repos.d/nvidia-container-toolkit.repo

#配置库以便可以使用各种包
sudo yum-config-manager --enable nvidia-container-toolkit-experimental

#安装NVIDIA Container Toolkit包
sudo yum install -y nvidia-container-toolkit
```

配置 Docker

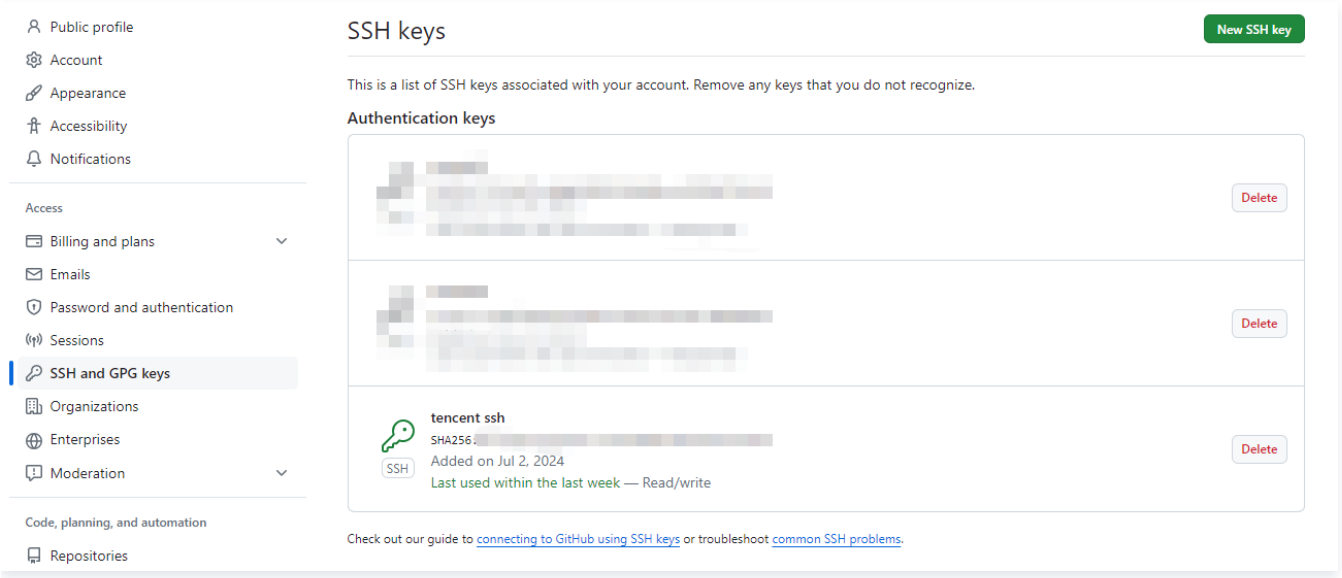
```
#使用nvidia-ctk命令修改并更新/etc/docker/daemon.json主机上的文件，以便Docker可以使用NVIDIA容器运行。
sudo nvidia-ctk runtime configure --runtime=docker

#重新启动Docker
sudo systemctl restart docker
```

配置客户端与远程 GitHub 仓库的连接

> ⚠ 注意：
> - 为了防止后续拉取 GitHub 仓库代码出现如下错误：fatal: Could not read from remote repository.
> - 则说明客户端还没有配置与远程 GitHub 仓库的连接，需要先配置 ssh 远程连接。

```
#生成密钥，其中youremail@example.com需替换成自己在GitHub上注册账号时所用的邮箱
ssh-keygen -t rsa -C "youremail@example.com"
```

打开生成的 /.ssh/id_rsa，复制密钥，在 GitHub 网站登录自己的账户，在账户选项中选择 **Settings -> SSH and GPG keys -> New SSH key**，把密钥填写到账户中(注意填写时的格式要求)。




```
#测试客户端是否和GitHub远程连接上，出现Success则配置成功
ssh -T git@github.com
```

安装 git 包

1. 检查系统是否安装 git：

```
git
```

> ⚠ 注意：
> 如果出现 –bash: git: command not found，则说明环境还没有安装 git 包。

2. 安装 git：

```
#安装git
sudo yum -y install git

#检查git版本
git --version
```

如果此时正常出现 git 的版本号，则说明安装成功。

参考文档

- NVIDIA Container Toolkit 安装指引
- Hugging Face 官网

DataParallel（DP）ResNet

最近更新时间：2024-08-29 10:19:52

本指导适用于在 TencentOS Server 3上使用 DataParallel（DP）训练框架运行 ResNet 模型，以 Docker 方式启动。

环境准备

1. 从 GitHub 下载 DP的Demo 开源仓库到本地。

```
git clone https://github.com/tczhangzhi/pytorch-distributed.git
cd pytorch-distributed
```

2. 由于 DP 是基于 Pytorch 开发的，我们使用 Pytorch 的基础镜像即可包含大部分运行环境。这里我们拉取 Nvidia 的基础镜像来启动容器。

```
docker run -it --gpus all --name=DP --ipc=host -v $PWD:/workspace nvcr.io/nvidia/pytorch:23.06-py3
/bin/bash
```

此时会从 nvcr 拉取 docker 镜像，请确保网络环境较好，直到镜像里所有层下载成功。成功后会直接进入容器内部。

数据集下载

我们使用 DP 训练框架运行 ResNet-50 模型，官方预训练 ResNet-50 模型最常用的数据集是 ImageNet-1K，所以这里我们使用 ImageNet-1K 训练模型。

1. 首先创建数据集的存放地址，放在 /workspace/datasets/imagenet 文件夹下，首先创建文件夹。

```
mkdir -p datasets/imagenet
cd datasets/imagenet
```

2. 在该目录下直接下载 ImageNet-1K 数据集，包括训练集，验证集和标签映射文件三个文件。

下载训练集：

```
wget https://image-net.org/data/ILSVRC/2012/ILSVRC2012_img_train.tar --no-check-certificate
```

下载验证集：

```
wget https://image-net.org/data/ILSVRC/2012/ILSVRC2012_img_val.tar --no-check-certificate
```

下载标签映射文件：

```
wget https://image-net.org/data/ILSVRC/2012/ILSVRC2012_devkit_t12.tar.gz --no-check-certificate
```

> ⚠ **注意：**
> 训练集包含1000个类，共计1281167张图片，大小有138G；验证集包含1000个类，共计50000张图片，大小有6G。请确保机器拥有较大的容量下载数据集。此外请确保网络环境较好，ImageNet 训练集较大需要较长时间下载。

训练集解压

1. 将训练集解压到 train 目录下。

```
mkdir train && tar -xvf ILSVRC2012_img_train.tar -C train && for x in `ls train/*tar`; do
fn=train/`basename $x .tar`; mkdir $fn; tar -xvf $x -C $fn; rm -f $fn.tar; done
```

2. 进入 train 目录下。

```
cd train
```

3. 查看该目录下的文件夹数量，若解压成功，正常情况下应该返回1000（代表1000个文件夹，数据集的1000个类）。

```
ls -lR|grep "^d"|wc -l
```

4. 查看该目录下的所有图片的数量，若解压成功，正常情况下应该返回1281167（代表一共有1281167张训练图像）。

```
ls -lR|grep "^-"|wc -l
```

验证集解压

1. 将验证集解压到 train 目录下。

```
mkdir val && tar xvf ILSVRC2012_img_val.tar -C ./val
```

2. 解压完成后，所有图像都在 val 文件夹里了。但是此时 val 下全是图像，没有被分到1000个文件夹里，需要将所有验证集图像分类到1000个类里去。
3. 解压标签映射文件 ILSVRC2012_devkit_t12.tar.gz，里面内容为每张图像对应的类别的映射关系。

```
tar -xzf ILSVRC2012_devkit_t12.tar.gz
```

4. 在 /imagenet 目录下创建 unzip.py 文件，写入以下代码：

```
from scipy import io
import os
import shutil

def move_valimg(val_dir='./val', devkit_dir='./ILSVRC2012_devkit_t12'):
    """
    move valimg to correspongding folders.
    val_id(start from 1) -> ILSVRC_ID(start from 1) -> WIND
    organize like:
    /val
       /n01440764
           images
       /n01443537
           images
        .....
    """
    # load synset, val ground truth and val images list
    synset = io.loadmat(os.path.join(devkit_dir, 'data', 'meta.mat'))

    ground_truth = open(os.path.join(devkit_dir, 'data', 'ILSVRC2012_validation_ground_truth.txt'))
    lines = ground_truth.readlines()
    labels = [int(line[:-1]) for line in lines]

    root, _, filenames = next(os.walk(val_dir))
    for filename in filenames:
        # val image name -> ILSVRC ID -> WIND
        val_id = int(filename.split('.')[0].split('_')[-1])
        ILSVRC_ID = labels[val_id-1]
        WIND = synset['synsets'][ILSVRC_ID-1][0][1][0]
        print("val_id:%d, ILSVRC_ID:%d, WIND:%s" % (val_id, ILSVRC_ID, WIND))

        # move val images
```

```
        output_dir = os.path.join(root, WIND)
        if os.path.isdir(output_dir):
            pass
        else:
            os.mkdir(output_dir)
        shutil.move(os.path.join(root, filename), os.path.join(output_dir, filename))

if __name__ == '__main__':
    move_valimg()
```

5. 执行该文件。

```
python unzip.py
```

这样验证集就被分到了1000个文件夹之下。

运行模型

DP 运行的文件为 dataparallel.py，其中有以下几处需要修改。

- 代码第118行，可以根据自己的 GPU 数量更改训练时使用的 GPU 数。例如我们这里拥有8块L40 GPU，我们将所有 GPU 用于训练模型。

```
#原有代码
gpus = [0, 1, 2, 3]

#更改为
gpus = [0, 1, 2, 3, 4, 5, 6, 7]
```

- 代码第375行会报错，将 view() 方法更改为 reshape() 方法即可。

```
#原有代码
correct_k = correct[:k].view(-1).float().sum(0, keepdim=True)

#更改为
correct_k = correct[:k].reshape(-1).float().sum(0, keepdim=True)
```

以上修改完之后，即可运行模型。这里我们测试的是 ResNet-50 模型通过 ImageNet-1K 数据集训练。

```
python dataparallel.py --data datasets/imagenet --arch resnet50 --epochs 120 --batch-size 256
```

此时会使用 ResNet-50 模型架构，从0开始训练120个 epochs，数据集在 datasets/imagenet 文件夹下，batchsize 为256，学习率 lr 为0.1，优化方法为 SGD 使用 momentum=0.9，权重衰退 weight decay=1e-4。参数设置来源于 ResNet 原论文：ResNet，从而尽可能的复现出论文里的结果。

> **说明：**
> 如需要更改 batchsize，学习率，momentum 参数和权重衰减参数，只需加上对应参数 `--batch-size`，`--learning-rate`，`--momentum`，`--weight-decay` 并设置想要的值即可。

此时会开始训练模型，由于我们设置 DP 汇总梯度的 GPU为 gpus[0]，（代码第138行 model = nn.DataParallel(model, device_ids=gpus, output_device=gpus[0])），所以 GPU0 的显存使用量会明显高于其他7块 GPU。
运行一个 epoch 的时间大约为半个小时，120个 epoch 大约需要两天训练完成。训练花费的时间可以在 dataparallel.csv 文件里找到。运行模型过程中会保存当前 epoch 训练结束之后的模型权重以及到当前 epoch 为止在验证集上最好性能的 epoch 的模型权重。
训练时命令行会出现训练时每一个 iteration 和测试时每一个 iteration 的性能以及每一个 epoch 测试完成之后的总的性能（例如第74个 epoch 的参考如下，top1 Acc 复现的结果还是比较接近论文中的结果）：

```
...
```

```
Test: [  0/196] Time  1.253 ( 1.253)    Loss 4.9941e-01 (4.9941e-01)    Acc@1  86.33 ( 86.33)   Acc@5
98.05 ( 98.05)
Test: [ 10/196] Time  0.919 ( 0.551)    Loss 8.8788e-01 (6.3331e-01)    Acc@1  80.86 ( 83.91)   Acc@5
93.36 ( 96.09)
Test: [ 20/196] Time  0.878 ( 0.518)    Loss 7.3491e-01 (6.8267e-01)    Acc@1  85.16 ( 82.59)   Acc@5
92.97 ( 95.48)
Test: [ 30/196] Time  0.279 ( 0.498)    Loss 8.0295e-01 (6.4392e-01)    Acc@1  79.30 ( 83.58)   Acc@5
94.92 ( 95.70)
Test: [ 40/196] Time  0.050 ( 0.489)    Loss 6.4413e-01 (6.9066e-01)    Acc@1  82.81 ( 82.16)   Acc@5
96.88 ( 95.57)
Test: [ 50/196] Time  0.050 ( 0.490)    Loss 4.8661e-01 (6.8918e-01)    Acc@1  86.33 ( 81.78)   Acc@5
97.66 ( 95.80)
Test: [ 60/196] Time  0.050 ( 0.496)    Loss 8.7274e-01 (7.0430e-01)    Acc@1  77.34 ( 81.37)   Acc@5
94.92 ( 95.86)
Test: [ 70/196] Time  0.051 ( 0.497)    Loss 7.7110e-01 (6.9134e-01)    Acc@1  80.08 ( 81.71)   Acc@5
95.31 ( 96.00)
Test: [ 80/196] Time  0.049 ( 0.499)    Loss 1.3679e+00 (7.1449e-01)    Acc@1  63.28 ( 81.25)   Acc@5
87.89 ( 95.73)
Test: [ 90/196] Time  0.050 ( 0.497)    Loss 1.9769e+00 (7.6551e-01)    Acc@1  54.69 ( 80.10)   Acc@5
84.38 ( 95.18)
Test: [100/196] Time  0.053 ( 0.498)    Loss 1.2061e+00 (8.1943e-01)    Acc@1  67.58 ( 78.93)   Acc@5
91.41 ( 94.58)
Test: [110/196] Time  0.050 ( 0.496)    Loss 8.6113e-01 (8.4266e-01)    Acc@1  76.56 ( 78.43)   Acc@5
94.14 ( 94.35)
Test: [120/196] Time  0.051 ( 0.498)    Loss 1.2805e+00 (8.6212e-01)    Acc@1  70.31 ( 78.14)   Acc@5
88.67 ( 94.05)
Test: [130/196] Time  0.050 ( 0.495)    Loss 6.3223e-01 (8.9568e-01)    Acc@1  83.20 ( 77.31)   Acc@5
97.27 ( 93.74)
Test: [140/196] Time  0.050 ( 0.494)    Loss 9.7670e-01 (9.1388e-01)    Acc@1  75.39 ( 76.99)   Acc@5
92.58 ( 93.53)
Test: [150/196] Time  0.050 ( 0.494)    Loss 1.0779e+00 (9.3261e-01)    Acc@1  77.73 ( 76.66)   Acc@5
89.45 ( 93.26)
Test: [160/196] Time  0.050 ( 0.494)    Loss 6.7641e-01 (9.4572e-01)    Acc@1  85.94 ( 76.39)   Acc@5
95.70 ( 93.11)
Test: [170/196] Time  0.050 ( 0.493)    Loss 6.4245e-01 (9.6438e-01)    Acc@1  83.20 ( 75.96)   Acc@5
97.27 ( 92.89)
Test: [180/196] Time  0.050 ( 0.492)    Loss 1.3427e+00 (9.8147e-01)    Acc@1  66.41 ( 75.60)   Acc@5
91.80 ( 92.69)
Test: [190/196] Time  0.051 ( 0.493)    Loss 1.2744e+00 (9.8115e-01)    Acc@1  66.41 ( 75.53)   Acc@5
94.53 ( 92.72)
 * Acc@1 75.674 Acc@5 92.778
```

同时dataparallel.csv可以看到训练的时间（参考）：

```
...
2024-08-13 08:36:33,2058.7685174942017
2024-08-13 09:10:52,2003.9772493839264
2024-08-13 09:44:17,2054.34983420372
2024-08-13 10:18:32,2003.627269268036
2024-08-13 10:51:56,1993.6664521694183
2024-08-13 11:25:10,2023.8221940994263
2024-08-13 11:58:54,2009.3610639572144
2024-08-13 12:32:24,2041.4226913452148
2024-08-13 13:06:26,1945.2222220897675
2024-08-13 13:38:51,2009.658884048462
2024-08-13 14:12:21,1953.2794036865234
2024-08-13 14:44:55,2023.0560326576233
```

同时目录下会出现 model_best.pth.tar 和 checkpoint.pth.tar，记录模型权重。

参考文档

- GitHub pytorch-distributed
- ImageNet 官网
- ImageNet Linux 下载解压指南
- ResNet

ViT

最近更新时间：2024-08-29 10:19:52

本指导适用于在 TencentOS Server 3上使用 DataParallel（DP）训练框架运行 ViT (Vision Transformer) 模型，以 Docker 方式启动。

前置环境条件

请确保已经按照 ResNet 文档内进行操作，**运行模型**之前的所有步骤已经完成，并已经准备好了运行模型的所有必要环境。

运行模型

DP 运行的文件为 dataparallel.py，其中有以下几处需要修改。

- 代码第118行，可以根据自己的 GPU 数量更改训练时使用的 GPU 数。例如我们这里拥有8块L40 GPU，我们将所有 GPU 用于训练模型。

```
#原有代码
gpus = [0, 1, 2, 3]

#更改为
gpus = [0, 1, 2, 3, 4, 5, 6, 7]
```

- 代码第375行会报错，将 view() 方法更改为 reshape() 方法即可。

```
#原有代码
correct_k = correct[:k].view(-1).float().sum(0, keepdim=True)

#更改为
correct_k = correct[:k].reshape(-1).float().sum(0, keepdim=True)
```

以上修改完之后，即可运行模型。这里我们测试的是 vit_base_16 模型通过 ImageNet-1K 数据集训练。

```
python dataparallel.py --data datasets/imagenet --arch vit_b_16 --epochs 300 --batch-size 2048 --
learning-rate 0.001
```

由于当前情况下 L40 batch_size 为原论文中4096时会显存溢出，所以这里采用2048的 batch_size 大小。
此时会使用 vit_base_16 模型架构，从0开始训练300个 epochs，数据集在 datasets/imagenet 文件夹下，batchsize 为2048，学习率lr为0.001，优化方法为SGD 使用 momentum=0.9，权重衰退 weight decay=1e-4。部分参数设置来源于ViT原论文：ViT，从而尽可能的复现出论文里的结果。

> **说明：**
> 如需要更改 momentum 参数和权重衰减参数，只需加上对应参数 `--momentum`，`--weight-decay` 并设置想要的值即可。
> 我们发现 batchsize 为2048时训练结果不一定特别理想，原文 batchsize 为4096，可以根据需要对相应参数进行修改以获得更好结果。

此时会开始训练模型，由于我们设置 DP 汇总梯度的 GPU 为 gpus[0]，（代码第138行 model = nn.DataParallel(model, device_ids=gpus, output_device=gpus[0])），所以 GPU0 的显存使用量会明显高于其他7块 GPU。
运行一个 epoch 的时间大约为半个小时，300个 epoch 大约需要六天训练完成。训练花费的时间可以在 dataparallel.csv 文件里找到。运行模型过程中会保存当前 epoch 训练结束之后的模型权重以及到当前 epoch 为止在验证集上最好性能的 epoch 的模型权重。
训练时命令行会出现训练时每一个 iteration 和测试时每一个 iteration 的性能以及每一个 epoch 测试完成之后的总的性能（例如第30个 epoch 的参考如下）：

```
...
Epoch: [30][480/626]    Time  5.109 ( 3.039)    Data  4.707 ( 2.446)    Loss 4.4270e+00 (4.3804e+00)
Acc@1  16.99 ( 17.58)   Acc@5  34.42 ( 36.74)
Epoch: [30][490/626]    Time  5.261 ( 3.041)    Data  4.859 ( 2.446)    Loss 4.3596e+00 (4.3801e+00)
Acc@1  18.26 ( 17.58)   Acc@5  38.23 ( 36.74)
Epoch: [30][500/626]    Time  5.156 ( 3.041)    Data  4.753 ( 2.446)    Loss 4.4136e+00 (4.3801e+00)
Acc@1  17.77 ( 17.58)   Acc@5  36.23 ( 36.73)
```

```
Epoch: [30][510/626]    Time  4.843 ( 3.040)    Data  4.441 ( 2.445)    Loss 4.3692e+00 (4.3798e+00)
Acc@1  19.19 ( 17.59)   Acc@5  36.91 ( 36.74)
Epoch: [30][520/626]    Time  5.095 ( 3.040)    Data  4.664 ( 2.444)    Loss 4.3212e+00 (4.3796e+00)
Acc@1  18.16 ( 17.59)   Acc@5  38.09 ( 36.74)
Epoch: [30][530/626]    Time  5.023 ( 3.040)    Data  4.621 ( 2.443)    Loss 4.3655e+00 (4.3794e+00)
Acc@1  18.07 ( 17.60)   Acc@5  37.30 ( 36.76)
Epoch: [30][540/626]    Time  5.103 ( 3.040)    Data  4.700 ( 2.442)    Loss 4.4637e+00 (4.3803e+00)
Acc@1  17.77 ( 17.59)   Acc@5  35.60 ( 36.74)
Epoch: [30][550/626]    Time  5.359 ( 3.041)    Data  4.955 ( 2.443)    Loss 4.3498e+00 (4.3800e+00)
Acc@1  18.02 ( 17.59)   Acc@5  36.28 ( 36.75)
Epoch: [30][560/626]    Time  5.073 ( 3.041)    Data  4.670 ( 2.442)    Loss 4.4005e+00 (4.3800e+00)
Acc@1  17.58 ( 17.59)   Acc@5  36.77 ( 36.75)
Epoch: [30][570/626]    Time  5.122 ( 3.042)    Data  4.717 ( 2.442)    Loss 4.4096e+00 (4.3798e+00)
Acc@1  17.82 ( 17.59)   Acc@5  35.94 ( 36.75)
Epoch: [30][580/626]    Time  5.111 ( 3.042)    Data  4.707 ( 2.442)    Loss 4.3924e+00 (4.3795e+00)
Acc@1  17.43 ( 17.59)   Acc@5  35.55 ( 36.75)
Epoch: [30][590/626]    Time  5.182 ( 3.043)    Data  4.780 ( 2.442)    Loss 4.3718e+00 (4.3795e+00)
Acc@1  17.77 ( 17.59)   Acc@5  36.62 ( 36.75)
Epoch: [30][600/626]    Time  5.250 ( 3.043)    Data  4.845 ( 2.442)    Loss 4.3822e+00 (4.3792e+00)
Acc@1  16.65 ( 17.60)   Acc@5  37.16 ( 36.75)
Epoch: [30][610/626]    Time  4.912 ( 3.043)    Data  4.508 ( 2.441)    Loss 4.2889e+00 (4.3789e+00)
Acc@1  17.87 ( 17.60)   Acc@5  38.92 ( 36.77)
Epoch: [30][620/626]    Time  4.951 ( 3.043)    Data  4.546 ( 2.441)    Loss 4.3697e+00 (4.3789e+00)
Acc@1  17.63 ( 17.60)   Acc@5  36.33 ( 36.76)
Test: [ 0/25]   Time  9.065 ( 9.065)    Loss 3.7750e+00 (3.7750e+00)    Acc@1  24.66 ( 24.66)   Acc@5
49.32 ( 49.32)
Test: [10/25]   Time  6.520 ( 4.251)    Loss 4.6754e+00 (4.0579e+00)    Acc@1  13.82 ( 20.16)   Acc@5
32.23 ( 42.12)
Test: [20/25]   Time  3.998 ( 3.994)    Loss 4.8062e+00 (4.3301e+00)    Acc@1  10.45 ( 17.44)   Acc@5
25.49 ( 37.14)
 * Acc@1 18.318 Acc@5 38.298
```

同时 dataparallel.csv 可以看到训练的时间（参考）：

```
...
2024-08-15 14:03:14,1982.8778982162476
2024-08-15 14:36:18,2001.0861496925354
2024-08-15 15:09:41,1995.106449842453
2024-08-15 15:42:57,2005.7094027996063
2024-08-15 16:16:24,1995.0537593364716
2024-08-15 16:49:41,2051.8484938144684
2024-08-15 17:23:54,2018.2189569473267
2024-08-15 17:57:34,1993.9278218746185
2024-08-15 18:30:49,2039.668051958084
2024-08-15 19:04:51,2022.5196571350098
2024-08-15 19:38:35,2007.2742729187012
2024-08-15 20:12:04,1991.4569637775421
2024-08-15 20:45:17,1996.3973467350006
2024-08-15 21:18:35,2011.3201851844788
```

同时目录下会出现 model_best.pth.tar和checkpoint.pth.tar，记录模型权重。

参考文档

- GitHub pytorch-distributed
- ViT

Distributed DataParallel（DDP）ResNet

最近更新时间：2024-08-29 10:19:52

本指导适用于在 TencentOS Server 3上使用 Distributed DataParallel（DDP）训练框架运行 ResNet 模型，以 Docker 方式启动。

环境准备

1. 从 GitHub 下载 DDP 的 Demo 开源仓库到本地。

```
git clone https://github.com/tczhangzhi/pytorch-distributed.git
cd pytorch-distributed
```

2. 由于 DDP 是基于 Pytorch 开发的，我们使用 Pytorch 的基础镜像即可包含大部分运行环境。这里我们拉取 Nvidia 的基础镜像来启动容器。

```
docker run -it --gpus all --name=DDP --ipc=host -v $PWD:/workspace nvcr.io/nvidia/pytorch:23.06-py3
/bin/bash
```

此时会从 nvcr 拉取 docker 镜像，请确保网络环境较好，直到镜像里所有层下载成功。成功后会直接进入容器内部。

数据集下载

我们使用 DDP 训练框架运行 ResNet-101 模型，官方预训练 ResNet-101 模型最常用的数据集是 ImageNet-1K，所以这里我们使用 ImageNet-1K 训练模型。

1. 首先创建数据集的存放地址，放在 /workspace/datasets/imagenet 文件夹下，首先创建文件夹。

```
mkdir -p datasets/imagenet
cd datasets/imagenet
```

2. 在该目录下直接下载 ImageNet-1K 数据集，包括训练集，验证集和标签映射文件三个文件。

下载训练集：

```
wget https://image-net.org/data/ILSVRC/2012/ILSVRC2012_img_train.tar --no-check-certificate
```

下载验证集：

```
wget https://image-net.org/data/ILSVRC/2012/ILSVRC2012_img_val.tar --no-check-certificate
```

下载标签映射文件：

```
wget https://image-net.org/data/ILSVRC/2012/ILSVRC2012_devkit_t12.tar.gz --no-check-certificate
```

> ⚠ **注意：**
> 训练集包含1000个类，共计1281167张图片，大小有138G；验证集包含1000个类，共计50000张图片，大小有6G。请确保机器拥有较大的容量下载数据集。此外请确保网络环境较好，ImageNet 训练集较大需要较长时间下载。

训练集解压

1. 将训练集解压到 train 目录下。

```
mkdir train && tar -xvf ILSVRC2012_img_train.tar -C train && for x in `ls train/*tar`; do
fn=train/`basename $x .tar`; mkdir $fn; tar -xvf $x -C $fn; rm -f $fn.tar; done
```

2. 进入 train 目录下。

```
cd train
```

3. 查看该目录下的文件夹数量，若解压成功，正常情况下应该返回1000（代表1000个文件夹，数据集的1000个类）。

```
ls -lR|grep "^d"|wc -l
```

4. 查看该目录下的所有图片的数量，若解压成功，正常情况下应该返回1281167（代表一共有1281167张训练图像）。

```
ls -lR|grep "^-"|wc -l
```

验证集解压

1. 将验证集解压到 train 目录下。

```
mkdir val && tar xvf ILSVRC2012_img_val.tar -C ./val
```

2. 解压完成后，所有图像都在 val 文件夹里了。但是此时 val 下全是图像，没有被分到1000个文件夹里，需要将所有验证集图像分类到1000个类里去。
3. 解压标签映射文件 ILSVRC2012_devkit_t12.tar.gz，里面内容为每张图像对应的类别的映射关系。

```
tar -xzf ILSVRC2012_devkit_t12.tar.gz
```

4. 在 /imagenet 目录下创建 unzip.py 文件，写入以下代码：

```
from scipy import io
import os
import shutil

def move_valimg(val_dir='./val', devkit_dir='./ILSVRC2012_devkit_t12'):
    """
    move valimg to correspongding folders.
    val_id(start from 1) -> ILSVRC_ID(start from 1) -> WIND
    organize like:
    /val
       /n01440764
           images
       /n01443537
           images
        .....
    """
    # load synset, val ground truth and val images list
    synset = io.loadmat(os.path.join(devkit_dir, 'data', 'meta.mat'))

    ground_truth = open(os.path.join(devkit_dir, 'data', 'ILSVRC2012_validation_ground_truth.txt'))
    lines = ground_truth.readlines()
    labels = [int(line[:-1]) for line in lines]

    root, _, filenames = next(os.walk(val_dir))
    for filename in filenames:
        # val image name -> ILSVRC ID -> WIND
        val_id = int(filename.split('.')[0].split('_')[-1])
        ILSVRC_ID = labels[val_id-1]
        WIND = synset['synsets'][ILSVRC_ID-1][0][1][0]
        print("val_id:%d, ILSVRC_ID:%d, WIND:%s" % (val_id, ILSVRC_ID, WIND))

        # move val images
```

```
        output_dir = os.path.join(root, WIND)
        if os.path.isdir(output_dir):
            pass
        else:
            os.mkdir(output_dir)
        shutil.move(os.path.join(root, filename), os.path.join(output_dir, filename))

if __name__ == '__main__':
    move_valimg()
```

5. 执行该文件。

```
python unzip.py
```

这样验证集就被分到了1000个文件夹之下。

运行模型

DDP 运行的文件为 distributed.py，其中有以下几处需要修改。

- 代码第74行，将 --local_rank 参数的 default 改为 int(os.environ["LOCAL_RANK"]，以便启用最新的 torchrun 方法启动。

```
#原有代码
default=-1,

#更改为
default=int(os.environ["LOCAL_RANK"]),
```

- 代码第375行会报错，将 view() 方法更改为 reshape() 方法即可。

```
#原有代码
correct_k = correct[:k].view(-1).float().sum(0, keepdim=True)

#更改为
correct_k = correct[:k].reshape(-1).float().sum(0, keepdim=True)
```

以上修改完之后，即可运行模型。这里我们测试的是 ResNet-101 模型通过 ImageNet-1K 数据集训练。由于我们有8张L40 gpu，所以 --nproc_per_node 设置为8。

```
torchrun --nproc_per_node=8 distributed.py --data datasets/imagenet --arch resnet101 --epochs 120 --
batch-size 256
```

此时会使用 ResNet-101 模型架构，从0开始训练120个 epochs，数据集在 datasets/imagenet 文件夹下，batchsize为256，学习率 lr 为0.1，优化方法为 SGD 使用 momentum=0.9，权重衰退 weight decay=1e-4。参数设置来源于 ResNet 原论文：ResNet，从而尽可能的复现出论文里的结果。

> **说明：**
> 如需要更改 batchsize，学习率，momentum 参数和权重衰减参数，只需加上对应参数 `--batch-size`，`--learning-rate`，`--momentum`，`--weight-decay` 并设置想要的值即可。

此时会开始训练模型，由于使用的是 DDP 训练模式，所以各个 gpu 之间的显存占用相差不大。运行模型过程中会保存当前 epoch 训练结束之后的模型权重以及到当前 epoch 为止在验证集上最好性能的 epoch 的模型权重。
训练时命令行会出现训练时每一个 iteration 和测试时每一个 iteration 的性能以及每一个 epoch 测试完成之后的总的性能（例如最后一个 epoch 的参考如下，top1 Acc 复现的结果还是比较接近论文中的结果）：

```
...
Test: [180/196] Time  0.021 ( 0.066)   Loss 9.2615e-01 (9.0108e-01)   Acc@1  78.91 ( 77.86)   Acc@5
92.58 ( 93.87)
```

```
Test: [180/196]    Time  0.021 ( 0.066)    Loss 9.2615e-01 (9.0108e-01)    Acc@1  78.91 ( 77.86)
Acc@5  92.58 ( 93.87)
Test: [180/196] Time  0.021 ( 0.066)    Loss 9.2615e-01 (9.0108e-01)    Acc@1  78.91 ( 77.86)   Acc@5
92.58 ( 93.87)
Test: [180/196]    Time  0.021 ( 0.066)    Loss 9.2615e-01 (9.0108e-01)    Acc@1  78.91 ( 77.86)
Acc@5  92.58 ( 93.87)
Test: [180/196] Time  0.021 ( 0.066)    Loss 9.2615e-01 (9.0108e-01)    Acc@1  78.91 ( 77.86)   Acc@5
92.58 ( 93.87)
Test: [180/196]    Time  0.021 ( 0.066)    Loss 9.2615e-01 (9.0108e-01)    Acc@1  78.91 ( 77.86)
Acc@5  92.58 ( 93.87)
Test: [180/196] Time  0.021 ( 0.066)    Loss 9.2615e-01 (9.0108e-01)    Acc@1  78.91 ( 77.86)   Acc@5
92.58 ( 93.87)
Test: [180/196] Time  0.021 ( 0.066)    Loss 9.2615e-01 (9.0108e-01)    Acc@1  78.91 ( 77.86)   Acc@5
92.58 ( 93.87)
Test: [190/196] Time  0.021 ( 0.066)    Loss 8.5628e-01 (8.9837e-01)    Acc@1  77.34 ( 77.88)   Acc@5
94.14 ( 93.88)
Test: [190/196]    Time  0.021 ( 0.066)    Loss 8.5628e-01 (8.9837e-01)    Acc@1  77.34 ( 77.88)
Acc@5  94.14 ( 93.88)
Test: [190/196] Time  0.021 ( 0.066)    Loss 8.5628e-01 (8.9837e-01)    Acc@1  77.34 ( 77.88)   Acc@5
94.14 ( 93.88)
Test: [190/196]    Time  0.021 ( 0.066)    Loss 8.5628e-01 (8.9837e-01)    Acc@1  77.34 ( 77.88)
Acc@5  94.14 ( 93.88)
Test: [190/196] Time  0.021 ( 0.066)    Loss 8.5628e-01 (8.9837e-01)    Acc@1  77.34 ( 77.88)   Acc@5
94.14 ( 93.88)
Test: [190/196]    Time  0.021 ( 0.066)    Loss 8.5628e-01 (8.9837e-01)    Acc@1  77.34 ( 77.88)
Acc@5  94.14 ( 93.88)
Test: [190/196] Time  0.021 ( 0.066)    Loss 8.5628e-01 (8.9837e-01)    Acc@1  77.34 ( 77.88)   Acc@5
94.14 ( 93.88)
Test: [190/196] Time  0.021 ( 0.066)    Loss 8.5628e-01 (8.9837e-01)    Acc@1  77.34 ( 77.88)   Acc@5
94.14 ( 93.88)
 * Acc@1 77.890 Acc@5 93.886
 * Acc@1 77.890 Acc@5 93.886
 * Acc@1 77.890 Acc@5 93.886
 * Acc@1 77.890 Acc@5 93.886
 * Acc@1 77.890 Acc@5 93.886
 * Acc@1 77.890 Acc@5 93.886
 * Acc@1 77.890 Acc@5 93.886
 * Acc@1 77.890 Acc@5 93.886
```

同时目录下会出现 model_best.pth.tar 和 checkpoint.pth.tar，记录模型权重。可以看到测试时同一个 iteration 有多条记录，这是因为 DDP 在每一张卡上都有一个进程，每一张卡都会打印一条记录。

参考文档

- GitHub pytorch-distributed
- ImageNet 官网
- ImageNet Linux 下载解压指南
- ResNet
- torchrun 教程

ViT

最近更新时间：2024-08-29 10:19:52

本指导适用于在 TencentOS Server 3上使用 Distributed DataParallel（DDP）训练框架运行 ViT（Vision Transformer）模型，以 Docker 方式启动。

前置环境条件

请确保已经按照 ResNet 文档内进行操作，**运行模型**之前的所有步骤已经完成，并已经准备好了运行模型的所有必要环境。

运行模型

DDP 运行的文件为 distributed.py，其中有以下几处需要修改。

- 代码第74行，将 --local_rank 参数的 default 改为 int(os.environ["LOCAL_RANK"]，以便启用最新的 torchrun 方法启动。

```
#原有代码
default=-1,

#更改为
default=int(os.environ["LOCAL_RANK"]),
```

- 代码第375行会报错，将 view() 方法更改为 reshape() 方法即可。

```
#原有代码
correct_k = correct[:k].view(-1).float().sum(0, keepdim=True)

#更改为
correct_k = correct[:k].reshape(-1).float().sum(0, keepdim=True)
```

以上修改完之后，即可运行模型。这里我们测试的是 vit_large_16 模型通过 ImageNet-1K 数据集训练。由于我们有8张L40 gpu，所以 --nproc_per_node 设置为8。

```
torchrun --nproc_per_node=8 distributed.py --data datasets/imagenet --arch vit_l_16 --epochs 400 --
batch-size 768 --learning-rate 0.001
```

此时会使用 vit_large_16 模型架构，从0开始训练400个 epochs，数据集在 datasets/imagenet 文件夹下，batchsize 为768，学习率 lr 为 0.001，优化方法为SGD 使用 momentum=0.9，权重衰退 weight decay=1e-4。部分参数设置来源于 ViT 原论文：ViT，从而尽可能的复现出论文里的结果。

> **说明：**
> 如需要更改 momentum 参数和权重衰减参数，只需加上对应参数 `--momentum`，`--weight-decay` 并设置想要的值即可。
> 同时测试环境L40如果 batchsize 选择1024则会显存溢出，所以选择768。

此时会开始训练模型，由于使用的是 DDP 训练模式，所以各个 gpu 之间的显存占用相差不大。运行模型过程中会保存当前 epoch 训练结束之后的模型权重以及到当前 epoch 为止在验证集上最好性能的 epoch 的模型权重。
训练时命令行会出现训练时每一个 teration 和测试时每一个 iteration 的性能以及每一个 epoch 测试完成之后的总的性能（例如最后一个 epoch 的参考如下）：

```
...
Test: [40/66]   Time  0.272 ( 0.285)   Loss 3.0449e+00 (2.9997e+00)   Acc@1  36.85 ( 36.49)   Acc@5
59.11 ( 60.83)
Test: [40/66]      Time  0.272 ( 0.285)   Loss 3.0449e+00 (2.9997e+00)   Acc@1  36.85 ( 36.49)
Acc@5  59.11 ( 60.83)
Test: [50/66]   Time  0.275 ( 0.283)   Loss 3.1013e+00 (3.0017e+00)   Acc@1  34.11 ( 36.38)   Acc@5
59.51 ( 60.88)
```

```
Test: [50/66]       Time  0.275 ( 0.283)    Loss 3.1013e+00 (3.0017e+00)    Acc@1  34.11 ( 36.38)
Acc@5  59.51 ( 60.88)
Test: [50/66]   Time  0.275 ( 0.283)    Loss 3.1013e+00 (3.0017e+00)    Acc@1  34.11 ( 36.38)   Acc@5
59.51 ( 60.88)
Test: [50/66]       Time  0.275 ( 0.284)    Loss 3.1013e+00 (3.0017e+00)    Acc@1  34.11 ( 36.38)
Acc@5  59.51 ( 60.88)
Test: [50/66]   Time  0.275 ( 0.284)    Loss 3.1013e+00 (3.0017e+00)    Acc@1  34.11 ( 36.38)   Acc@5
59.51 ( 60.88)
Test: [50/66]   Time  0.275 ( 0.284)    Loss 3.1013e+00 (3.0017e+00)    Acc@1  34.11 ( 36.38)   Acc@5
59.51 ( 60.88)
Test: [50/66]       Time  0.275 ( 0.284)    Loss 3.1013e+00 (3.0017e+00)    Acc@1  34.11 ( 36.38)
Acc@5  59.51 ( 60.88)
Test: [50/66]   Time  0.275 ( 0.283)    Loss 3.1013e+00 (3.0017e+00)    Acc@1  34.11 ( 36.38)   Acc@5
59.51 ( 60.88)
Test: [60/66]   Time  0.273 ( 0.282)    Loss 3.0019e+00 (3.0007e+00)    Acc@1  37.11 ( 36.44)   Acc@5
61.20 ( 60.89)
Test: [60/66]       Time  0.273 ( 0.282)    Loss 3.0019e+00 (3.0007e+00)    Acc@1  37.11 ( 36.44)
Acc@5  61.20 ( 60.89)
Test: [60/66]   Time  0.273 ( 0.282)    Loss 3.0019e+00 (3.0007e+00)    Acc@1  37.11 ( 36.44)   Acc@5
61.20 ( 60.89)
Test: [60/66]       Time  0.273 ( 0.282)    Loss 3.0019e+00 (3.0007e+00)    Acc@1  37.11 ( 36.44)
Acc@5  61.20 ( 60.89)
Test: [60/66]   Time  0.273 ( 0.282)    Loss 3.0019e+00 (3.0007e+00)    Acc@1  37.11 ( 36.44)   Acc@5
61.20 ( 60.89)
Test: [60/66]       Time  0.273 ( 0.283)    Loss 3.0019e+00 (3.0007e+00)    Acc@1  37.11 ( 36.44)
Acc@5  61.20 ( 60.89)
Test: [60/66]   Time  0.273 ( 0.282)    Loss 3.0019e+00 (3.0007e+00)    Acc@1  37.11 ( 36.44)   Acc@5
61.20 ( 60.89)
Test: [60/66]   Time  0.273 ( 0.282)    Loss 3.0019e+00 (3.0007e+00)    Acc@1  37.11 ( 36.44)   Acc@5
61.20 ( 60.89)
* Acc@1 36.486 Acc@5 60.978
* Acc@1 36.486 Acc@5 60.978
* Acc@1 36.486 Acc@5 60.978
* Acc@1 36.486 Acc@5 60.978
* Acc@1 36.486 Acc@5 60.978
* Acc@1 36.486 Acc@5 60.978
* Acc@1 36.486 Acc@5 60.978
* Acc@1 36.486 Acc@5 60.978
```

同时目录下会出现 model_best.pth.tar 和 checkpoint.pth.tar，记录模型权重。可以看到测试时同一个 iteration 有多条记录，这是因为 DDP 在每一张卡上都有一个进程，每一张卡都会打印一条记录。

参考文档

- GitHub pytorch-distributed
- ViT
- torchrun 教程

DeepSpeed ResNet

最近更新时间：2024-08-28 17:44:21

本指导适用于在 TencentOS Server 3上使用 DeepSpeed 训练框架运行 ResNet 模型的官方 Demo，以 Docker 方式启动。

环境准备

1. 从 GitHub 下载 DeepSpeed 的 Demo 开源仓库到本地。

```
git clone https://github.com/microsoft/DeepSpeedExamples.git
cd DeepSpeedExamples/
```

2. 由于 DeepSpeed 镜像已经长期没有更新，我们使用 Pytorch 的基础镜像即可包含大部分运行环境。这里我们拉取 Nvidia 的基础镜像来启动容器。

```
docker run -it --gpus all --name=deepspeed -e HF_ENDPOINT="https://hf-mirror.com" --ipc=host -v
$PWD:/workspace nvcr.io/nvidia/pytorch:23.06-py3 /bin/bash
```

此时会从 nvcr 拉取 docker 镜像，请确保网络环境较好，直到镜像里所有层下载成功。成功后会直接进入容器内部。

> ⚠ **注意：**
> 使用 -e HF_ENDPOINT="https://hf-mirror.com" 是因为其中有些 examples 会从 Hugging Face 导入某些模型或者参数，我们需要保证与 Hugging Face 的通信是正常的。

数据集下载

我们使用 DeepSpeed 训练框架运行 ResNet-18 模型，官方预训练 ResNet-18 模型最常用的数据集是 ImageNet-1K，所以这里我们使用 ImageNet-1K 训练模型。

1. 首先创建数据集的存放地址，放在 /workspace/datasets/imagenet 文件夹下，首先创建文件夹。

```
mkdir -p datasets/imagenet
cd datasets/imagenet
```

2. 在该目录下直接下载 ImageNet-1K 数据集，包括训练集，验证集和标签映射文件三个文件。

下载训练集：

```
wget https://image-net.org/data/ILSVRC/2012/ILSVRC2012_img_train.tar --no-check-certificate
```

下载验证集：

```
wget https://image-net.org/data/ILSVRC/2012/ILSVRC2012_img_val.tar --no-check-certificate
```

下载标签映射文件：

```
wget https://image-net.org/data/ILSVRC/2012/ILSVRC2012_devkit_t12.tar.gz --no-check-certificate
```

> ⚠ **注意：**
> 训练集包含1000个类，共计1281167张图片，大小有138G；验证集包含1000个类，共计50000张图片，大小有6G。请确保机器拥有较大的容量下载数据集。此外请确保网络环境较好，ImageNet 训练集较大需要较长时间下载。

训练集解压

1. 将训练集解压到 train 目录下。

```
mkdir train && tar -xvf ILSVRC2012_img_train.tar -C train && for x in `ls train/*tar`; do
fn=train/`basename $x .tar`; mkdir $fn; tar -xvf $x -C $fn; rm -f $fn.tar; done
```

2. 进入 train 目录下。

```
cd train
```

3. 查看该目录下的文件夹数量，若解压成功，正常情况下应该返回1000（代表1000个文件夹，数据集的1000个类）。

```
ls -lR|grep "^d"|wc -l
```

4. 查看该目录下的所有图片的数量，若解压成功，正常情况下应该返回1281167（代表一共有1281167张训练图像）。

```
ls -lR|grep "^-"|wc -l
```

验证集解压

1. 将验证集解压到 train 目录下。

```
mkdir val && tar xvf ILSVRC2012_img_val.tar -C ./val
```

2. 解压完成后，所有图像都在 val 文件夹里了。但是此时 val 下全是图像，没有被分到1000个文件夹里，需要将所有验证集图像分类到1000个类里去。
3. 解压标签映射文件 ILSVRC2012_devkit_t12.tar.gz，里面内容为每张图像对应的类别的映射关系。

```
tar -xzf ILSVRC2012_devkit_t12.tar.gz
```

4. 在 /imagenet 目录下创建 unzip.py 文件，写入以下代码：

```
from scipy import io
import os
import shutil

def move_valimg(val_dir='./val', devkit_dir='./ILSVRC2012_devkit_t12'):
    """
    move valimg to correspongding folders.
    val_id(start from 1) -> ILSVRC_ID(start from 1) -> WIND
    organize like:
    /val
       /n01440764
           images
       /n01443537
           images
        .....
    """
    # load synset, val ground truth and val images list
    synset = io.loadmat(os.path.join(devkit_dir, 'data', 'meta.mat'))

    ground_truth = open(os.path.join(devkit_dir, 'data', 'ILSVRC2012_validation_ground_truth.txt'))
    lines = ground_truth.readlines()
    labels = [int(line[:-1]) for line in lines]

    root, _, filenames = next(os.walk(val_dir))
    for filename in filenames:
        # val image name -> ILSVRC ID -> WIND
        val_id = int(filename.split('.')[0].split('_')[-1])
        ILSVRC_ID = labels[val_id-1]
```

```
        WIND = synset['synsets'][ILSVRC_ID-1][0][1][0]
        print("val_id:%d, ILSVRC_ID:%d, WIND:%s" % (val_id, ILSVRC_ID, WIND))

        # move val images
        output_dir = os.path.join(root, WIND)
        if os.path.isdir(output_dir):
            pass
        else:
            os.mkdir(output_dir)
        shutil.move(os.path.join(root, filename), os.path.join(output_dir, filename))

if __name__ == '__main__':
    move_valimg()
```

5. 执行该文件。

```
python unzip.py
```

这样验证集就被分到了1000个文件夹之下。

运行模型

安装运行模型必要的包

1. 本示例使用 DeepSpeed 运行 ResNet-18 模型，位于 training/imagenet/ 文件夹下。

```
cd training/imagenet/
```

2. 将 pip 换为国内清华源以加快下载速度。

```
#将pip换成清华源
#设为默认，永久有效
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

3. 安装运行 DeepSpeed 必要的包，通过 pip 下载。

```
#安装所有需要的包
pip install datasets fire loguru sh deepspeed transformers openpyxl
```

所有需要的包安装成功之后，即可开始准备运行模型。

运行模型

使用 main.py 文件运行模型，首先需要对 main.py 内部代码进行一些修改。
将32行的 data 变成可添加的参数，在前面加上“--”即可。

```
#修改前
parser.add_argument('data', metavar='DIR', nargs='?', default='imagenet',

#修改后
parser.add_argument('--data', metavar='DIR', nargs='?', default='imagenet',
```

修改好之后进行保存，即可运行模型。

```
deepspeed main.py -a resnet18 --deepspeed --deepspeed_config config/ds_config.json --data
/workspace/datasets/imagenet --multiprocessing_distributed
```

⚠ 注意：

可以在 config/ds_config.json 里更改训练的参数，同时命令行也可以对训练参数进行更改。可更改参数如下：

```
parser.add_argument('--data', metavar='DIR', nargs='?', default='imagenet',
                    help='path to dataset (default: imagenet)')
parser.add_argument('-a', '--arch', metavar='ARCH', default='resnet18',
                    choices=model_names,
                    help='model architecture: ' +
                        ' | '.join(model_names) +
                        ' (default: resnet18)')
parser.add_argument('-j', '--workers', default=4, type=int, metavar='N',
                    help='number of data loading workers (default: 4)')
parser.add_argument('--epochs', default=90, type=int, metavar='N',
                    help='number of total epochs to run')
parser.add_argument('--start-epoch', default=0, type=int, metavar='N',
                    help='manual epoch number (useful on restarts)')
parser.add_argument('-b', '--batch-size', default=256, type=int,
                    metavar='N',
                    help='mini-batch size (default: 256), this is the total '
                         'batch size of all GPUs on the current node when '
                         'using Data Parallel or Distributed Data Parallel')
parser.add_argument('--lr', '--learning-rate', default=0.1, type=float,
                    metavar='LR', help='initial learning rate', dest='lr')
parser.add_argument('--momentum', default=0.9, type=float, metavar='M',
                    help='momentum')
parser.add_argument('--wd', '--weight-decay', default=1e-4, type=float,
                    metavar='W', help='weight decay (default: 1e-4)',
                    dest='weight_decay')
parser.add_argument('-p', '--print-freq', default=10, type=int,
                    metavar='N', help='print frequency (default: 10)')
parser.add_argument('--resume', default='', type=str, metavar='PATH',
                    help='path to latest checkpoint (default: none)')
parser.add_argument('-e', '--evaluate', dest='evaluate', action='store_true',
                    help='evaluate model on validation set')
parser.add_argument('--pretrained', dest='pretrained', action='store_true',
                    help='use pre-trained model')
parser.add_argument('--world-size', default=-1, type=int,
                    help='number of nodes for distributed training')
parser.add_argument('--seed', default=None, type=int,
                    help='seed for initializing training. ')
parser.add_argument('--gpu', default=None, type=int,
                    help='GPU id to use.')
parser.add_argument('--multiprocessing_distributed', action='store_true',
                    help='Use multi-processing distributed training to launch '
                         'N processes per node, which has N GPUs. This is the '
                         'fastest way to use PyTorch for either single node or '
                         'multi node data parallel training')
parser.add_argument('--dummy', action='store_true', help="use fake data to benchmark")
parser.add_argument('--local_rank', type=int, default=-1, help="local rank for distributed training
on gpus")

parser.add_argument('--data', metavar='DIR', nargs='?', default='imagenet',
                    help='path to dataset (default: imagenet)')
parser.add_argument('-a', '--arch', metavar='ARCH', default='resnet18',
                    choices=model_names,
                    help='model architecture: ' +
                        ' | '.join(model_names) +
                        ' (default: resnet18)')
parser.add_argument('-j', '--workers', default=4, type=int, metavar='N',
                    help='number of data loading workers (default: 4)')
```

```
    parser.add_argument('--epochs', default=90, type=int, metavar='N',
                        help='number of total epochs to run')
    parser.add_argument('--start-epoch', default=0, type=int, metavar='N',
                        help='manual epoch number (useful on restarts)')
    parser.add_argument('-b', '--batch-size', default=256, type=int,
                        metavar='N',
                        help='mini-batch size (default: 256), this is the total '
                             'batch size of all GPUs on the current node when '
                             'using Data Parallel or Distributed Data Parallel')
    parser.add_argument('--lr', '--learning-rate', default=0.1, type=float,
                        metavar='LR', help='initial learning rate', dest='lr')
    parser.add_argument('--momentum', default=0.9, type=float, metavar='M',
                        help='momentum')
    parser.add_argument('--wd', '--weight-decay', default=1e-4, type=float,
                        metavar='W', help='weight decay (default: 1e-4)',
                        dest='weight_decay')
    parser.add_argument('-p', '--print-freq', default=10, type=int,
                        metavar='N', help='print frequency (default: 10)')
    parser.add_argument('--resume', default='', type=str, metavar='PATH',
                        help='path to latest checkpoint (default: none)')
    parser.add_argument('-e', '--evaluate', dest='evaluate', action='store_true',
                        help='evaluate model on validation set')
    parser.add_argument('--pretrained', dest='pretrained', action='store_true',
                        help='use pre-trained model')
    parser.add_argument('--world-size', default=-1, type=int,
                        help='number of nodes for distributed training')
    parser.add_argument('--seed', default=None, type=int,
                        help='seed for initializing training. ')
    parser.add_argument('--gpu', default=None, type=int,
                        help='GPU id to use.')
    parser.add_argument('--multiprocessing_distributed', action='store_true',
                        help='Use multi-processing distributed training to launch '
                             'N processes per node, which has N GPUs. This is the '
                             'fastest way to use PyTorch for either single node or '
                             'multi node data parallel training')
    parser.add_argument('--dummy', action='store_true', help="use fake data to benchmark")
    parser.add_argument('--local_rank', type=int, default=-1, help="local rank for distributed training
    on gpus")
```

模型运行中途会保存训练好的权重参数，训练输出如下所示（参考）：

```
...
Test: [171/196] Time  0.119 ( 0.034)    Loss 1.3480e+00 (1.2659e+00)    Acc@1  68.75 ( 68.97)   Acc@5
93.75 ( 89.18)
Test: [171/196] Time  0.119 ( 0.034)    Loss 5.9162e-01 (1.2233e+00)    Acc@1  90.62 ( 69.35)   Acc@5
96.88 ( 89.71)
Test: [181/196] Time  0.074 ( 0.032)    Loss 1.3154e+00 (1.2337e+00)    Acc@1  59.38 ( 69.82)   Acc@5
90.62 ( 89.33)
Test: [181/196] Time  0.027 ( 0.033)    Loss 2.0253e+00 (1.2606e+00)    Acc@1  53.12 ( 68.89)   Acc@5
81.25 ( 88.52)
Test: [181/196] Time  0.109 ( 0.033)    Loss 1.0418e+00 (1.2571e+00)    Acc@1  65.62 ( 69.54)   Acc@5
96.88 ( 89.04)
Test: [181/196] Time  0.005 ( 0.033)    Loss 1.4672e+00 (1.3019e+00)    Acc@1  56.25 ( 69.32)   Acc@5
84.38 ( 88.00)
Test: [181/196] Time  0.105 ( 0.033)    Loss 1.2578e+00 (1.2628e+00)    Acc@1  56.25 ( 69.30)   Acc@5
93.75 ( 89.17)
Test: [181/196] Time  0.005 ( 0.033)    Loss 1.7441e+00 (1.2842e+00)    Acc@1  59.38 ( 68.56)   Acc@5
78.12 ( 88.98)
```

```
Test: [181/196] Time  0.005 ( 0.033)    Loss 1.0964e+00 (1.2335e+00)    Acc@1  65.62 ( 69.01)   Acc@5
93.75 ( 89.55)
Test: [171/196] Time  0.092 ( 0.035)    Loss 8.8425e-01 (1.2210e+00)    Acc@1  75.00 ( 70.19)   Acc@5
93.75 ( 88.98)
Test: [191/196] Time  0.005 ( 0.032)    Loss 1.1533e+00 (1.2184e+00)    Acc@1  65.62 ( 70.09)   Acc@5
93.75 ( 89.59)
Test: [191/196] Time  0.005 ( 0.032)    Loss 1.1176e+00 (1.2504e+00)    Acc@1  71.88 ( 69.65)   Acc@5
96.88 ( 89.10)
Test: [191/196] Time  0.010 ( 0.033)    Loss 1.4503e+00 (1.3034e+00)    Acc@1  65.62 ( 69.32)   Acc@5
84.38 ( 87.93)
Test: [191/196] Time  0.062 ( 0.033)    Loss 1.7334e+00 (1.2584e+00)    Acc@1  50.00 ( 69.01)   Acc@5
87.50 ( 88.53)
Test: [191/196] Time  0.005 ( 0.033)    Loss 1.2310e+00 (1.2586e+00)    Acc@1  78.12 ( 69.42)   Acc@5
90.62 ( 89.17)
Test: [191/196] Time  0.109 ( 0.033)    Loss 2.0439e+00 (1.2897e+00)    Acc@1  56.25 ( 68.39)   Acc@5
75.00 ( 88.89)
Test: [181/196] Time  0.054 ( 0.035)    Loss 6.9501e-01 (1.2399e+00)    Acc@1  78.12 ( 69.82)   Acc@5
100.00 ( 88.81)
Test: [191/196] Time  0.118 ( 0.034)    Loss 1.4798e+00 (1.2315e+00)    Acc@1  50.00 ( 68.88)   Acc@5
87.50 ( 89.59)
Test: [191/196] Time  0.078 ( 0.035)    Loss 1.0073e+00 (1.2461e+00)    Acc@1  68.75 ( 69.63)   Acc@5
93.75 ( 88.79)
 *   Acc@1 69.496 Acc@5 89.034
 *   Acc@1 69.496 Acc@5 89.034
 *   Acc@1 69.496 Acc@5 89.034
 *   Acc@1 69.496 Acc@5 89.034
 *   Acc@1 69.496 Acc@5 89.034
 *   Acc@1 69.496 Acc@5 89.034
 *   Acc@1 69.496 Acc@5 89.034
 *   Acc@1 69.496 Acc@5 89.034
```

同时目录下会出现 model_best.pth.tar 记录精度最好的模型权重。可以看到测试时同一个 iteration 有多条记录，这是因为运行时在每一张卡上都有一个进程，每一张卡都会打印一条记录。训练完成之后目录下会出现 Acc_loss_log.xlsx，用于记录精度和 loss 变化。

参考文档

- DeepSpeedExamples GitHub
- DeepSpeedExamples imagenet ResNet
- ImageNet 官网
- ImageNet Linux 下载解压指南
- ResNet

Bert

最近更新时间：2024-08-28 17:44:21

本指导适用于在 TencentOS Server 3上使用 DeepSpeed 训练框架运行 Bert 模型的官方 Demo，以 Docker 方式启动。

环境准备

1. 从 GitHub下载 DeepSpeed 的 Demo 开源仓库到本地。

```
git clone https://github.com/microsoft/DeepSpeedExamples.git
cd DeepSpeedExamples/
```

2. 由于 DeepSpeed 镜像已经长期没有更新，我们使用 Pytorch 的基础镜像即可包含大部分运行环境。这里我们拉取 Nvidia 的基础镜像来启动容器。

```
docker run -it --gpus all --name=deepspeed -e HF_ENDPOINT="https://hf-mirror.com" --ipc=host -v
$PWD:/workspace nvcr.io/nvidia/pytorch:23.06-py3 /bin/bash
```

此时会从 nvcr 拉取 docker 镜像，请确保网络环境较好，直到镜像里所有层下载成功。成功后会直接进入容器内部。

> ⚠ **注意：**
> 使用 -e HF_ENDPOINT="https://hf-mirror.com" 是因为其中有些 examples 会从 Hugging Face 导入某些模型或者参数，我们需要保证与 Hugging Face 的通信是正常的。

安装 DeepSpeed 以及必要的包

1. 将 pip 换为国内清华源以加快下载速度。

```
#将pip换成清华源
#设为默认，永久有效
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

2. 安装运行 DeepSpeed 必要的包，通过 pip 下载。

```
#安装所有需要的包
pip install datasets fire loguru sh deepspeed transformers
```

所有需要的包安装成功之后，即可开始准备运行模型。

运行模型

1. 这里我们运行 HelloDeepSpeed 下的 Bert 模型。

```
cd training/HelloDeepSpeed/
```

2. 这里我们跑 train_bert_ds.py 文件，该文件可以直接调用 GPU 进行训练。该文件运行时还存在一些 bug，需要对第500行和第514行的代码进行一些修改。

```
#第500行
#原代码
handle.write(gitlog.stdout.decode("utf-8"))
#更改为
handle.write(gitlog)

#第514行
#原代码
handle.write(gitdiff.stdout.decode("utf-8"))
```

```
#更改为
handle.write(gitdiff)
```

3. 修改完成之后保存，我们就可以通过容器运行 Bert 的代码了。

```
#运行模型
deepspeed train_bert_ds.py --checkpoint_dir ./experiments
```

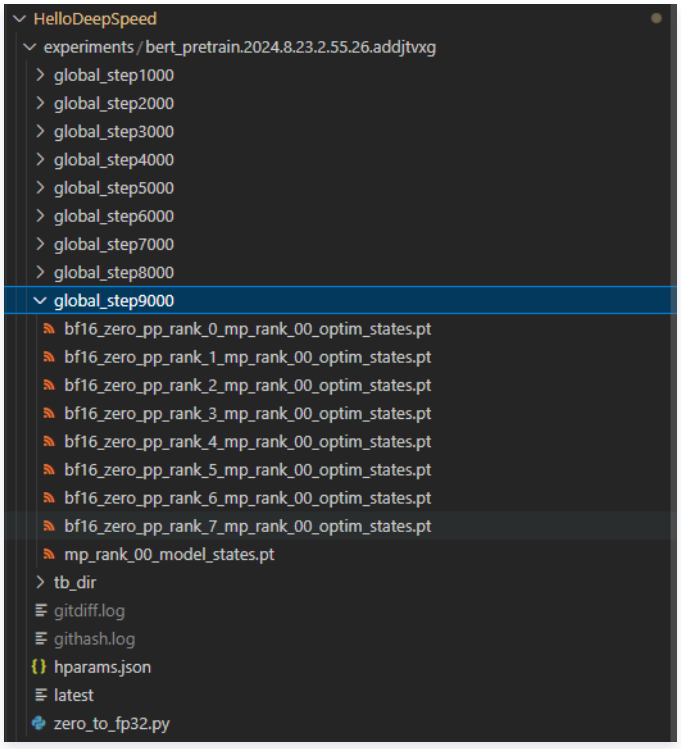
> ⚠ **注意：**
> 除了 checkpoint_dir 参数，可以将训练好的模型放在 experiments 文件夹下；还可以调整其他各种参数，如下所示，具体可参见：DeepSpeedExamples HelloDeepSpeed Bert：
>
> ```
> --mask_prob ${mask_prob} \
> --random_replace_prob ${random_replace_prob} \
> --unmask_replace_prob ${unmask_replace_prob} \
> --max_seq_length ${max_seq_length} \
> --tokenizer ${tokenizer} \
> --num_layers ${num_layers} \
> --num_heads ${num_heads} \
> --ff_dim ${ff_dim} \
> --h_dim ${h_dim} \
> --dropout ${dropout} \
> --batch_size ${batch_size} \
> --num_iterations ${num_iterations} \
> --checkpoint_every ${checkpoint_every} \
> --log_every ${log_every} \
> --local_rank ${local_rank}
> ```

4. 此时会根据拥有的 GPU 数量运行模型，可以看到如下内容：

```
[2024-08-23 10:26:00,572] [INFO] [logging.py:96:log_dist] [Rank 0] step=8300, skipped=0, lr=[0.0001],
mom=[(0.9, 0.999)]
[2024-08-23 10:26:00,585] [INFO] [timer.py:259:stop] epoch=0/micro_step=8300/global_step=8300,
RunningAvgSamplesPerSec=308.66381961861657, CurrSamplesPerSec=312.85119175664465, MemAllocated=0.07GB,
MaxMemAllocated=1.71GB
2024-08-23 10:26:00.586 | INFO     | __main__:log_dist:54 - [Rank 0] Loss: 6.6766
[2024-08-23 10:26:02,736] [INFO] [logging.py:96:log_dist] [Rank 0] step=8310, skipped=0, lr=[0.0001],
mom=[(0.9, 0.999)]
[2024-08-23 10:26:02,743] [INFO] [timer.py:259:stop] epoch=0/micro_step=8310/global_step=8310,
RunningAvgSamplesPerSec=308.66526213278956, CurrSamplesPerSec=314.5054045739528, MemAllocated=0.07GB,
MaxMemAllocated=1.71GB
2024-08-23 10:26:02.744 | INFO     | __main__:log_dist:54 - [Rank 0] Loss: 6.6762
[2024-08-23 10:26:04,874] [INFO] [logging.py:96:log_dist] [Rank 0] step=8320, skipped=0, lr=[0.0001],
mom=[(0.9, 0.999)]
[2024-08-23 10:26:04,902] [INFO] [timer.py:259:stop] epoch=0/micro_step=8320/global_step=8320,
RunningAvgSamplesPerSec=308.66605948195775, CurrSamplesPerSec=310.6948689220451, MemAllocated=0.07GB,
MaxMemAllocated=1.71GB
2024-08-23 10:26:04.904 | INFO     | __main__:log_dist:54 - [Rank 0] Loss: 6.6756
[2024-08-23 10:26:07,056] [INFO] [logging.py:96:log_dist] [Rank 0] step=8330, skipped=0, lr=[0.0001],
mom=[(0.9, 0.999)]
[2024-08-23 10:26:07,060] [INFO] [timer.py:259:stop] epoch=0/micro_step=8330/global_step=8330,
RunningAvgSamplesPerSec=308.66868241767776, CurrSamplesPerSec=307.43401607097644, MemAllocated=0.07GB,
MaxMemAllocated=1.71GB
2024-08-23 10:26:07.061 | INFO     | __main__:log_dist:54 - [Rank 0] Loss: 6.6751
[2024-08-23 10:26:09,219] [INFO] [logging.py:96:log_dist] [Rank 0] step=8340, skipped=0, lr=[0.0001],
mom=[(0.9, 0.999)]
```

```
[2024-08-23 10:26:09,234] [INFO] [timer.py:259:stop] epoch=0/micro_step=8340/global_step=8340,
RunningAvgSamplesPerSec=308.6679433643697, CurrSamplesPerSec=297.1419886451415, MemAllocated=0.07GB,
MaxMemAllocated=1.71GB
```

5. 此时会看到 loss 在一直递减，默认运行10000个 step，运行完成之后，可以在 experiments 文件夹下看到 Bert 运行时保存的 log 和权重。



参考文档

- DeepSpeedExamples GitHub
- DeepSpeedExamples HelloDeepSpeed Bert
- deepspeed.ai bert-pretraining tutorials
- Bert 运行报错指引

Megatron-LM GPT

最近更新时间：2024-08-28 17:44:21

本指导适用于在 TencentOS Server 3上使用 Megatron-LM 训练框架运行 GPT 模型的官方 Demo，以 Docker 方式启动。

环境准备

1. 从 GitHub 下载 Megatron-LM 的开源仓库到本地。

```
git clone https://github.com/NVIDIA/Megatron-LM.git
cd Megatron-LM
```

2. 接下来我们使用 Pytorch 的基础镜像即可包含大部分运行环境，这里我们拉取 Nvidia 的基础镜像来启动容器。

```
docker run -it --gpus all --name=megatron -e HF_ENDPOINT="https://hf-mirror.com" --ipc=host -v
$PWD:/workspace nvcr.io/nvidia/pytorch:24.01-py3 /bin/bash
```

此时会从 nvcr 拉取 docker 镜像，请确保网络环境较好，直到镜像里所有层下载成功。成功后会直接进入容器内部。

> ⚠ **注意：**
> 使用 -e HF_ENDPOINT="https://hf-mirror.com" 是因为后续下载数据集从 datasets 库中下载，我们需要保证与 Hugging Face 的通信是正常的。

安装运行模型必要的包

1. 将 pip 换为国内清华源以加快下载速度。

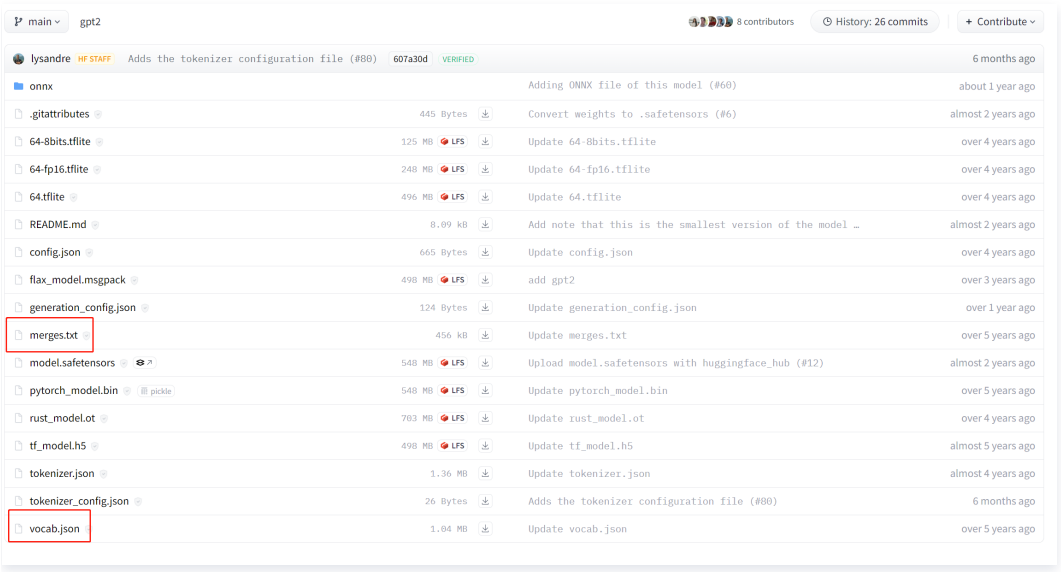
```
#将pip换成清华源
#设为默认，永久有效
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

2. 安装运行 megatron 必要的包，通过 pip 下载。

```
#安装所有需要的包
pip install datasets nltk
```

下载分词器和合并表

在 分词器 合并表下载 网站下载 merges.txt 和 vocab.json 两个文件，放在 workspace/ 文件夹下。



数据集下载以及数据预处理

1. 新建 dataset.py 文件，输入以下代码：

```
from datasets import load_dataset

train_data = load_dataset('codeparrot/codeparrot-clean-train', split='train')
train_data.to_json("codeparrot_data.json", lines=True)
```

2. 运行代码

```
python dataset.py
```

此时会开始下载 CodeParrot 数据集并转化为 json 格式，json 一行包括一个文本样本。

3. 随后将 json 数据处理成二进制格式：

```
python tools/preprocess_data.py \
       --input codeparrot_data.json \
       --output-prefix codeparrot \
       --vocab vocab.json \
       --dataset-impl mmap \
       --tokenizer-type GPT2BPETokenizer \
       --merge-file merges.txt \
       --json-keys content \
       --workers 32 \
       --chunk-size 25 \
       --append-eod
```

完成后会输出 codeparrot_content_document.idx 和 codeparrot_content_document.bin 两个文件用于训练。

训练模型

1. 新建 train.sh 脚本，输入以下代码：

```
GPUS_PER_NODE=8
MASTER_ADDR=localhost
MASTER_PORT=6001
NNODES=1
NODE_RANK=0
WORLD_SIZE=$(($GPUS_PER_NODE*$NNODES))
```

```
DISTRIBUTED_ARGS="--nproc_per_node $GPUS_PER_NODE
                  --nnodes $NNODES
                  --node_rank $NODE_RANK
                  --master_addr $MASTER_ADDR
                  --master_port $MASTER_PORT"
CHECKPOINT_PATH=/workspace/Megatron-LM/experiments/codeparrot-small
VOCAB_FILE=vocab.json
MERGE_FILE=merges.txt
DATA_PATH=codeparrot_content_document
GPT_ARGS="--num-layers 12
          --hidden-size 768
          --num-attention-heads 12
          --seq-length 1024
          --max-position-embeddings 1024
          --micro-batch-size 12
          --global-batch-size 192
          --lr 0.0005
          --train-iters 150000
          --lr-decay-iters 150000
          --lr-decay-style cosine
          --lr-warmup-iters 2000
          --weight-decay .1
          --adam-beta2 .999
          --fp16
          --log-interval 10
          --save-interval 2000
          --eval-interval 200
          --eval-iters 10"
TENSORBOARD_ARGS="--tensorboard-dir experiments/tensorboard"
python3 -m torch.distributed.launch $DISTRIBUTED_ARGS \
        pretrain_gpt.py \
        --tensor-model-parallel-size 1 \
        --pipeline-model-parallel-size 1 \
        $GPT_ARGS \
        --vocab-file $VOCAB_FILE \
        --merge-file $MERGE_FILE \
        --save $CHECKPOINT_PATH \
        --load $CHECKPOINT_PATH \
        --data-path $DATA_PATH \
        $TENSORBOARD_ARGS
```

2. 运行脚本

```
bash train.sh
```

即可开始训练模型，最后得到训练权重。

参考文档

- Megatron-LM GitHub
- Megatron-LM gpt3
- 如何使用 Megatron-LM 训练语言模型

虚拟化与容器使用指南
Docker 容器用户使用指南

最近更新时间：2024-11-11 14:11:01

在 TencentOS Server 上使用 Docker 容器技术

Docker 是一种开源的容器引擎，可以轻松地在不同的操作系统中运行应用程序，并支持快速、高效和可扩展的部署。在本教程中，我们将介绍如何在 TencentOS Server 上使用 Docker 容器技术，并通过示例和实践经验分享任务如何管理和部署容器。

步骤1：安装 Docker

默认情况下，Docker 并不随 TencentOS Server 操作系统一起安装。因此，在使用 Docker 之前，我们需要通过命令行安装。

通过 YUM 安装 Docker

1. 安装 Docker。
   使用以下命令在 TencentOS Server 上安装 Docker：

```
$ sudo yum install -y docker-ce
```

2. 启动 Docker。
   使用以下命令运行 Docker：

```
$ sudo systemctl start docker
```

3. 检查 Docker 状态。
   使用以下命令检查 Docker 是否正在运行：

```
$ sudo systemctl status docker
```

通过安装脚本安装 Docker

1. 获取 Docker 安装脚本。
   使用以下命令获取 Docker 安装脚本：

```
$ curl -fsSL https://get.docker.com -o get-docker.sh
```

2. 运行 Docker 安装脚本。
   使用以下命令运行 Docker 安装脚本：

```
$ sudo sh get-docker.sh
```

3. 启动 Docker。
   使用以下命令运行 Docker：

```
$ sudo systemctl start docker
```

4. 检查 Docker 状态。
使用以下命令检查 Docker 是否正在运行：

```
$ sudo systemctl status docker
```

步骤2：Docker 容器的使用

练习一：运行一个 Hello World 容器

1. 使用以下命令运行一个 Hello World 容器：

```
$ sudo docker run hello-world
```

2. 查看 Docker 容器运行状态。

   使用以下命令查看 Docker 容器的运行状态：

```
$ sudo docker ps
```

练习二：运行 Nginx 容器

1. 使用以下命令下载 Nginx 镜像：

```
$ sudo docker pull nginx
```

2. 下载 Nginx 镜像后，使用以下命令运行 Nginx 容器：

```
$ sudo docker run -d -p 8080:80 nginx
```

   上述命令将在 Docker 环境中启动一个 Nginx 容器，并将容器的标准80端口映射到主机的 `8080` 端口。

3. 检查 Docker 容器运行状态。

   使用以下命令查看 Docker 容器的运行状态：

```
$ sudo docker ps
```

4. 访问 Nginx 服务。

   使用以下命令在本地 web 浏览器中访问 Nginx 服务：

```
http://localhost:8080/
```

   Nginx 欢迎页面应该能在您的本地 web 浏览器中呈现。

常用命令

Docker 提供了许多命令，可以让您轻松地创建、管理和部署容器。以下是一些常用的 Docker 命令：

- `sudo docker run`：运行新容器
- `sudo docker ps`：列出当前正在运行的容器
- `sudo docker images`：列出本地仓库中的当前镜像
- `sudo docker stop`：停止指定容器
- `sudo docker rm`：删除已停止的容器