

# 计算加速套件 TACO Kit

## 实践教程



腾讯云

## 【 版权声明 】

©2013–2025 腾讯云版权所有

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

## 【 商标声明 】



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

## 【 服务声明 】

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

## 【 联系我们 】

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

# 文档目录

## 实践教程

### AIGC 场景

TACO Train 加速 Stable Diffusion 模型训练

TACO Infer 优化 Stable Diffusion 系列模型

TACO Infer 部署 Stable Diffusion web UI

# 实践教程

## AIGC 场景

# TACO Train 加速 Stable Diffusion 模型训练

最近更新时间：2025-05-21 17:28:22

## 操作场景

本文将演示如何使用 GPU 云服务器，训练 AI 绘画模型，结合 TACO Train 的加速能力助力您获得 4 倍以上的性能提升。

## 操作步骤

### 购买高性能计算实例

购买实例，其中实例、存储及镜像请参见以下信息选择，其余配置请参见 [通过购买页创建实例](#) 按需选择。

- **实例**：选择 **GPU 型 HCCPNV4h**、**GPU 计算型 GT4**。
- **镜像**：建议选择**公共镜像**，支持自动安装 GPU 驱动。若选择 HCC 机型，公共镜像当中已安装 RDMA 网卡驱动。
  - 操作系统请使用 CentOS 7.6、Ubuntu 18.04 或 TencentOS 2.4（TK4）。
  - 若您选择**公共镜像**，则请勾选**后台自动安装 GPU 驱动**，实例将在系统启动后预装对应版本驱动。如下图所示：



## 安装docker和NVIDIA docker

1. 参见 [使用标准登录方式登录 Linux 实例](#)，登录实例。

## 2. 执行以下命令，安装 docker。

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

若您无法通过该命令安装，请尝试多次执行命令，或参见 Docker 官方文档 [Install Docker Engine](#) 进行安装。

## 3. 执行以下命令，安装 nvidia-docker2。

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

若您无法通过该命令安装，请尝试多次执行命令，或参见 NVIDIA 官方文档 [Installation Guide & mdash](#) 进行安装。

## 启动训练环境

```
#!/bin/bash  
  
docker run \  
    -itd \  
    --gpus all \  
    --privileged --cap-add=IPC_LOCK \  
    --ulimit memlock=-1 --ulimit stack=67108864 \  
    --net=host \  
    --ipc=host \  
    --name=sd \  
    ccr.ccs.tencentyun.com/qcloud/taco-train:torch20-cu117-bm-0.7.2  
  
docker exec -it sd bash
```

该镜像包含的软件版本信息如下：

- OS: Ubuntu 20.04.5 LTS
- python: 3.8.10
- CUDA toolkits: V11.7.99
- cuDNN: 8.5.0
- pytorch: 2.0.0+cu117
- DeepSpeed: 0.8.2
- Transformers: 4.27.1

- xformers: 0.0.17+6967620.d20230323
- diffusers: 0.15.0.dev0 (main branch untill March/24/2023)

## 开始测试

```
cd /workspace/text_to_image
bash run.sh
```

测试代码来自 [stable diffusers 官方](#) 的 [examples](#)。

### ❗ 说明:

1. 原始的测试代码打印了单步的耗时，波动较大，这里对每步的训练耗时做了平均，方便性能对比。
2. 第一次运行训练脚本会下载预训练模型，耗时5分钟左右。
3. [模型](#) 和 [数据集](#) 信息来自 [huggingface 官网](#)。

训练过程中的输出如下:

```
[2023-03-24 11:17:56,637] torch._inductor.compile_fx: [INFO] Step 3: torchinductor done compiling BACKWARDS graph 2
[2023-03-24 11:17:56,642] torch._inductor.compile_fx: [INFO] Step 3: torchinductor compiling BACKWARDS graph 1
[2023-03-24 11:17:56,678] torch._inductor.compile_fx: [INFO] Step 3: torchinductor done compiling BACKWARDS graph 1
[2023-03-24 11:17:56,685] torch._inductor.compile_fx: [INFO] Step 3: torchinductor done compiling BACKWARDS graph 3
[2023-03-24 11:17:56,690] torch._inductor.compile_fx: [INFO] Step 3: torchinductor compiling BACKWARDS graph 2
[2023-03-24 11:17:56,766] torch._inductor.compile_fx: [INFO] Step 3: torchinductor done compiling BACKWARDS graph 2
[2023-03-24 11:17:56,771] torch._inductor.compile_fx: [INFO] Step 3: torchinductor compiling BACKWARDS graph 1
[2023-03-24 11:17:56,803] torch._inductor.compile_fx: [INFO] Step 3: torchinductor done compiling BACKWARDS graph 1
[2023-03-24 11:17:56,991] torch._inductor.compile_fx: [INFO] Step 3: torchinductor done compiling BACKWARDS graph 2
[2023-03-24 11:17:56,996] torch._inductor.compile_fx: [INFO] Step 3: torchinductor compiling BACKWARDS graph 1
[2023-03-24 11:17:57,029] torch._inductor.compile_fx: [INFO] Step 3: torchinductor done compiling BACKWARDS graph 1
Steps: 100% | 100/100 [03:12<00:00, 1.02s/it, avg_train_time=1.083 data_time=0.00673]
{'requires_safety_checker'} was not found in config. Values will be initialized to default values.
'text_config_dict' is provided which will be used to initialize 'CLIPTextConfig'. The value 'text_config["id2label"]' will be overridden.
{'prediction_type'} was not found in config. Values will be initialized to default values.
/usr/local/lib/python3.8/dist-packages/transformers/models/clip/feature_extraction_clip.py:28: FutureWarning: The class CLIPFeatureExtractor is deprecated and will be removed in version 5 of Transformers.
Please use CLIPImageProcessor instead.
warnings.warn(
Configuration saved in sd-pokemon-model/model_index.json
Configuration saved in sd-pokemon-model/user/config.json
```

训练过程中的GPU/CPU/内存状态如下:

Every 1.0s: nvidia-smi

Fri Mar 24 11:47:22 2023

通达

Mar 24 11:47:23 2023

NVIDIA-SMI 470.82.01 Driver Version: 470.82.01 CUDA Version: 11.4

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 A100-SXM... On 00000000:0E:00.0 Off 0 N/A 54C P0 386W / 400W 40315MiB / 40536MiB 99% Default Disabled

1 NVIDIA A100-SXM... On 00000000:13:00.0 Off 0 N/A 54C P0 429W / 400W 36648MiB / 40536MiB 100% Default Disabled

2 NVIDIA A100-SXM... On 00000000:4B:00.0 Off 0 N/A 51C P0 386W / 400W 36648MiB / 40536MiB 100% Default Disabled

3 NVIDIA A100-SXM... On 00000000:51:00.0 Off 0 N/A 55C P0 315W / 400W 36646MiB / 40536MiB 100% Default Disabled

4 NVIDIA A100-SXM... On 00000000:93:00.0 Off 0 N/A 52C P0 356W / 400W 36648MiB / 40536MiB 100% Default Disabled

5 NVIDIA A100-SXM... On 00000000:99:00.0 Off 0 N/A 49C P0 195W / 400W 36648MiB / 40536MiB 100% Default Disabled

6 NVIDIA A100-SXM... On 00000000:C8:00.0 Off 0 N/A 51C P0 379W / 400W 36648MiB / 40536MiB 100% Default Disabled

7 NVIDIA A100-SXM... On 00000000:D0:00.0 Off 0 N/A 52C P0 394W / 400W 36360MiB / 40536MiB 95% Default Disabled

Processes:

GPU GI CI PID Type Process name GPU Memory Usage

0 N/A N/A 143654 C /usr/bin/python3 33225MiB

top - 11:47:24 up 13 days, 23:58, 5 users, load average: 7.90, 3.21, 1.83

Tasks: 1990 total, 34 running, 1049 sleeping, 0 stopped, 0 zombie

%Cpu(s): 5.2 us, 0.4 sy, 0.0 ni, 94.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st

KiB Mem : 10558923+total, 72220230+free, 11514908+used, 21854099+buff/cache

KiB Swap: 0 total, 0 free, 0 used. 92703398+avail Mem

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

34733 root 20 0 2495136 48476 5456 S 110.2 0.0 1248.12 barad\_agent

143654 root 20 0 80.8g 7.6g 1.6g S 94.7 0.8 1:40.54 python3

143655 root 20 0 86.0g 8.0g 1.7g S 94.7 0.8 1:41.40 python3

143656 root 20 0 86.7g 8.7g 1.7g S 94.7 0.9 1:42.05 python3

143657 root 20 0 86.8g 8.9g 1.7g S 94.7 0.9 1:41.74 python3

143658 root 20 0 86.7g 8.8g 1.7g S 94.7 0.9 1:40.99 python3

143662 root 20 0 85.9g 8.0g 1.7g S 94.7 0.8 1:42.73 python3

143664 root 20 0 85.5g 8.3g 1.7g S 94.7 0.8 1:42.05 python3

143659 root 20 0 86.8g 8.9g 1.7g S 94.4 0.9 1:41.14 python3

4077 root 20 0 9704136 109896 26912 S 12.9 0.0 156:10.00 dockerd

146329 root 20 0 53.7g 6.6g 205580 R 5.9 0.7 0:02.92 python3

144194 root 20 0 54.1g 6.3g 219552 R 5.6 0.6 0:04.07 python3

144195 root 20 0 52.0g 5.8g 142864 R 5.6 0.6 0:04.67 python3

144233 root 20 0 54.8g 7.0g 217224 R 5.6 0.7 0:04.15 python3

144387 root 20 0 52.0g 5.8g 130320 R 5.6 0.6 0:03.01 python3

144509 root 20 0 54.8g 7.0g 203144 R 5.6 0.7 0:02.67 python3

144887 root 20 0 54.9g 7.1g 219264 R 5.6 0.7 0:04.93 python3

145024 root 20 0 54.9g 7.1g 219560 R 5.6 0.7 0:04.70 python3

145120 root 20 0 54.0g 6.3g 217856 R 5.6 0.6 0:04.22 python3

145296 root 20 0 54.8g 7.0g 204984 R 5.6 0.7 0:02.79 python3

145526 root 20 0 53.7g 6.6g 204748 R 5.6 0.7 0:02.62 python3

145718 root 20 0 54.9g 7.1g 204928 R 5.6 0.7 0:02.28 python3

145844 root 20 0 54.8g 7.0g 206820 R 5.6 0.7 0:02.40 python3

146483 root 20 0 54.9g 7.1g 204712 R 5.6 0.7 0:02.41 python3

146485 root 20 0 54.0g 6.2g 204344 R 5.6 0.6 0:02.93 python3

144383 root 20 0 54.1g 6.3g 205536 R 5.3 0.6 0:02.25 python3

144525 root 20 0 54.1g 6.3g 204768 R 5.3 0.6 0:02.33 python3

144615 root 20 0 52.0g 5.8g 130000 R 5.3 0.6 0:02.46 python3

Every 1.0s: free -h

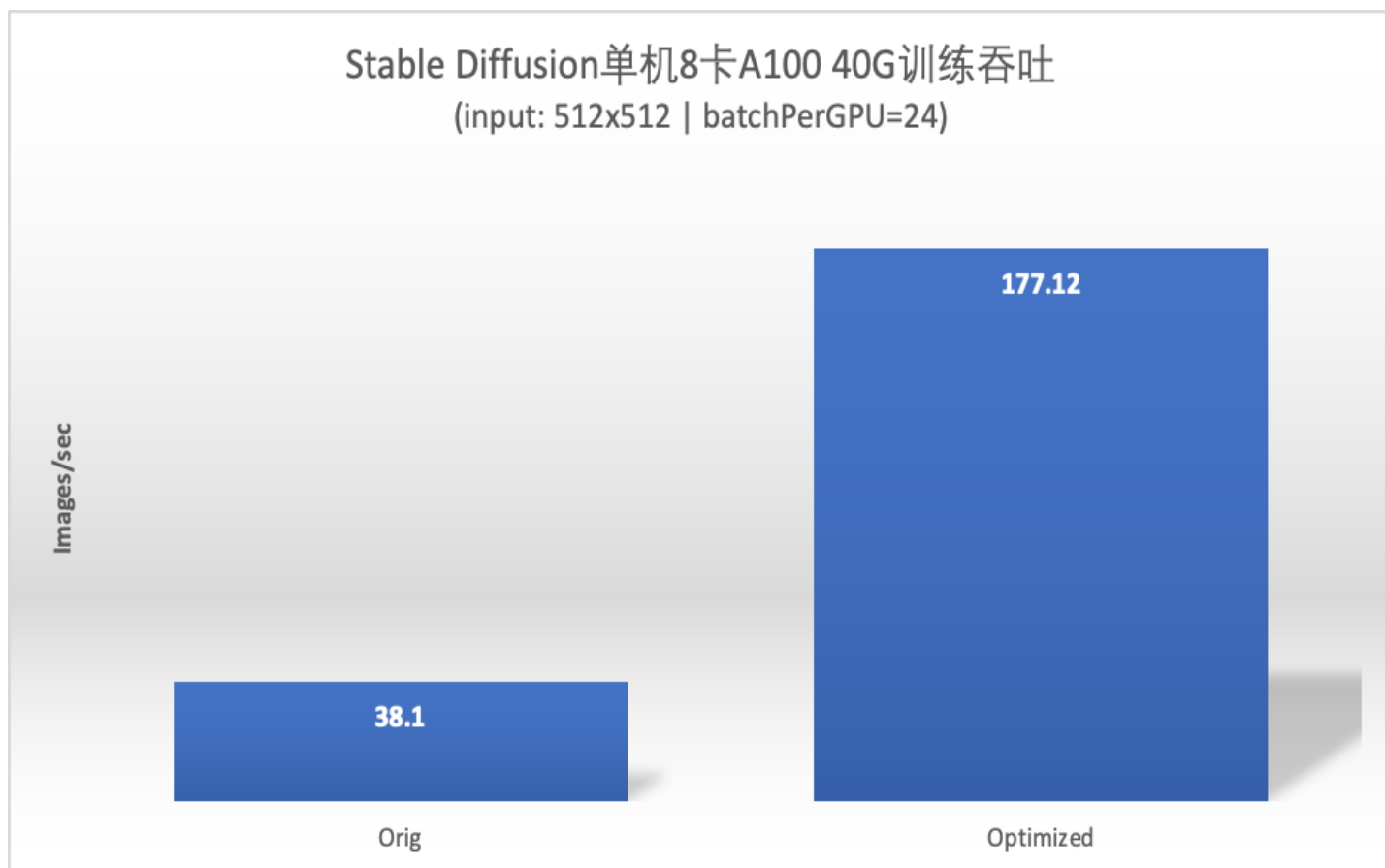
Fri Mar 24 11:47:24 2023

total used free shared buff/cache available

Mem: 1.0T 110G 687G 6.0G 209G 882G

Swap: 0B 0B 0B

性能加速效果



## 总结

本文基于腾讯云高性能计算实例评测运行了官方 Stable diffusion 训练脚本，运行过程中通过性能分析挖掘了若干个训练性能优化方向并加以实施，最终取得了4倍多的性能提升。



# TACO Infer 优化 Stable Diffusion 系列模型

最近更新时间：2024-06-05 10:25:31

## 操作场景

本文将演示如何使用 GPU 云服务器优化 AI 绘画模型，模型范围包括以 Stable Diffusion 1.5为基础的系列模型，您可以使用 Lora 结合模型使用，支持ControlNet。TACO Infer 的加速能力优化后，端到端时延可减少约 30%~50%。

## 操作步骤

### 购买 GPU 云服务器

购买实例，其中实例、存储及镜像请参见以下信息选择，其余配置请参见 [通过购买页创建实例](#) 按需选择。

- **实例**：选择 **计算型 PN4**。
- **系统盘**：配置容量不小于500GB的云硬盘。
- **镜像**：建议选择**公共镜像**。
- 操作系统使用 CentOS 7.9。
- 选择公共镜像后请勾选**后台自动安装 GPU 驱动**，实例将在系统启动后预装对应版本驱动。如下图所示：

镜像 ?

公共镜像自定义镜像共享镜像镜像市场

TencentOS

CentOS

Windows

Ubuntu

64位

CentOS 7.9 64位 ?

☒ 后台自动安装GPU驱动 ?

GPU驱动版本 470.82.01

CUDA版本 11.4.3

cuDNN版本 8.2.4

#### ⚠ 注意：

- 当前优化版本仅支持 A10、A100、V100 GPU 机型，请检查您的实例配置。

- 优化过程会有中间状态文件产生，建议您系统盘容量不小于500GB。

## 安装 docker 和 NVIDIA docker

1. 参见 [使用标准登录方式登录 Linux 实例](#)，登录实例。
2. 执行以下命令，安装 docker。

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

若您无法通过该命令安装，请尝试多次执行命令，或参见 Docker 官方文档 [Install Docker Engine](#) 进行安装。

3. 执行以下命令，安装 nvidia-docker2。

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

若您无法通过该命令安装，请尝试多次执行命令，或参见 NVIDIA 官方文档 [Installation Guide & mdash](#) 进行安装。

## 下载并启动 docker 镜像

```
docker pull xxxxxxxx  
docker run -it --rm --gpus=all --network=host xxxxxxxx
```

演示需要的所有数据和运行环境全部打包在 docker 镜像中，支持优化的镜像获取请 [联系我们](#)，审核通过后会联系您。

### ❗ 说明：

docker 镜像近作为优化模型步骤演示，部署在生产环境中需要您自制镜像，TACO Infer 提供软件安装包。

## 环境准备

1. 进入 `/root/sd_webui_demo/stable-diffusion-webui` 目录中，执行：

```
bash webui.sh
```

脚本会自动构建 python virtualenv 环境，安装 stable-diffusion-webui 所有相关依赖，完成后您可以退出进程。

2. 进入 `/root/sd_webui_demo` 目录中，执行以下命令安装 TACO Infer，下载模型权重：

```
source env.sh
```

❗ 说明：

- `env.sh` 文件中提供示例模型下载，如果您需要下载自己模型，需要修改脚本中下载路径。
- 如果您重新进入容器，可以执行 `source /root/sd_webui_demo/stable-diffusion-webui/venv/bin/activate` 激活虚拟环境。

## （可选）合并LoRA到模型中

1. 进入 `/root/sd_webui_demo/stable-diffusion-v1-5` 目录中，使用 `git clone https://github.com/kohya-ss/sd-scripts` 命令，下载合并 LoRA 与基础模型的脚本库。

2. 进入 `/root/sd_webui_demo/stable-diffusion-v1-5/sd-scripts` 目录，配置环境变量：

```
export PYTHONPATH=/root/sd_webui_demo/stable-diffusion-v1-5/sd-scripts
```

3. 参见以下命令合并 LoRA 与基础模型：

```
python networks/merge_lora.py --sd_model ../v1-5-pruned-emaonly.safetensors --save_to ../lora-v1-5-pruned-emaonly.safetensors --models <LoRA文件目录> --ratios 0.8
```

## 模型导出

1. 执行 `python export_model.py`，用 `diffusers` 加载模型权重，并导出为 `torchscript`，原文件内容如下，注意修改相应的部分：

```
import torch
import functools
from diffusers import StableDiffusionPipeline

def get_unet(device="cuda:0"):
    # change to your model path
    model_path = "../stable-diffusion-v1-5"
    # change if your model merged with LoRA
    # model_path = "../stable-diffusion-v1-5/lora-v1-5-pruned-emaonly.safetensors"
```

```
# pipe = StableDiffusionPipeline.from_ckpt(model_path).to(device)
pipe =
StableDiffusionPipeline.from_pretrained(model_path).to(device)
UNET = pipe.unet
UNET.eval()
UNET.to(memory_format=torch.channels_last) # use channels_last
memory format
UNET.forward = functools.partial(UNET.forward, return_dict=False)
# set return_dict=False as default
return UNET

def get_sample_input(batch_size, latent_height, latent_width,
device="cuda:0"):
    dtype = torch.float32
    text_maxlen = 77
    embedding_dim = 768
    return (
        torch.randn(2*batch_size, 4, latent_height, latent_width,
dtype=dtype, device=device),
        torch.tensor([1.]*batch_size, dtype=dtype, device=device),
        torch.randn(2*batch_size, text_maxlen, embedding_dim,
dtype=dtype, device=device)
    )

model = get_UNET()
# change according to your image (1, h/8, w/8)
test_data = get_sample_input(1, 64, 64)
script_model = torch.jit.trace(model, test_data, strict=False)
script_model.save("origin_model/trace_module.pt")
```

#### 注意:

如果是单文件格式（ckpt 或 safetensors）或您使用合并 LoRA 后的模型，请修改 `pipe = StableDiffusionPipeline.from_pretrained(model_path).to(device)` 为 `pipe = StableDiffusionPipeline.from_ckpt(model_path).to(device)`。

## 2. 进入 `origin_model` 目录下可以查看导出的模型:

```
[root@vm-0-46-centos demo]# ll origin_model/
total 3358720
```

```
-rw-r--r-- 1 500 500 3439324538 Mar 13 16:56 trace_module.pt
```

## 模型优化

1. 执行 `python demo.py` 命令即可启动 TACO Infer 对导出的 UNetModel 进行性能优化，注意参见注释修改 `demo.py` 文件：

### A10、A100优化代码

```
import torch

from taco import optimize_gpu, ModelConfig, OptimizeConfig

def gen_test_data(batch_size=1):
    # change according to your image (1, h/8, w/8)
    x = torch.randn(batch_size, 4, 64, 64)
    timesteps = torch.tensor([1]*batch_size)
    # change according to your model hidden size
    context = torch.randn(batch_size, 77, 768)
    return (x, timesteps, context)

input_model = "origin_model/trace_module.pt"
output_model_dir = "./optimized_model"
optimize_config = OptimizeConfig()

optimize_config.pytorch.trt.enable_fp16 = True
optimize_config.pytorch.profilng.subproc = False
optimize_config.profilng.num_run = 50
optimize_config.profilng.num_warmup = 10

model_config = ModelConfig()
model_config.pytorch.large_model=True

# change according to your image (h/8, w/8)
# change according to your model hidden size
model_config.pytorch.inputs_shape_range = [
    {"min": (1, 4, 64, 64), "opt": (2, 4, 64, 64), "max": (32, 4,
64, 64)},
    {"min": (1,), "opt": (2,), "max": (32,)},
```

```
        {"min": (1, 77, 768), "opt": (2, 77, 768), "max": (32, 77, 768)},
    ],

    report = optimize_gpu(
        input_model,
        output_model_dir,
        test_data=gen_test_data(batch_size=2),
        optimize_config = optimize_config,
        model_config = model_config
    )
```

## V100 优化代码

docker 镜像内默认优化代码是 A10 的优化代码，如果您在 V100 机型上使用，请使用以下代码进行优化：

```
import os
import torch

from taco import optimize_gpu, ModelConfig, OptimizeConfig

def gen_test_data(batch_size=1):
    dtype = torch.float16
    device = "cuda:0"
    # change according to your image (1, h/8, w/8)
    x = torch.randn(batch_size, 4, 64, 64, dtype=dtype)
    timesteps = torch.tensor([1]*batch_size, dtype=dtype)
    # change according to your model hidden size
    context = torch.randn(batch_size, 77, 768, dtype=dtype)

    return (
        x,
        timesteps,
        context,
        torch.randn(batch_size, 1280, 8, 8, dtype=dtype), #
        control net mid_residual
        torch.randn(batch_size, 320, 64, 64, dtype=dtype), #
        control net down_residual0
        torch.randn(batch_size, 320, 64, 64, dtype=dtype), #
        control net down_residual1
```

```
        torch.randn(batch_size, 320, 64, 64, dtype=dtype), #
control net down_residual2
        torch.randn(batch_size, 320, 32, 32, dtype=dtype), #
control net down_residual3
        torch.randn(batch_size, 640, 32, 32, dtype=dtype), #
control net down_residual4
        torch.randn(batch_size, 640, 32, 32, dtype=dtype), #
control net down_residual5
        torch.randn(batch_size, 640, 16, 16, dtype=dtype), #
control net down_residual6
        torch.randn(batch_size, 1280, 16, 16, dtype=dtype), #
control net down_residual7
        torch.randn(batch_size, 1280, 16, 16, dtype=dtype), #
control net down_residual8
        torch.randn(batch_size, 1280, 8, 8, dtype=dtype), #
control net down_residual9
        torch.randn(batch_size, 1280, 8, 8, dtype=dtype), #
control net down_residual10
        torch.randn(batch_size, 1280, 8, 8, dtype=dtype), #
control net down_residual10
    )

input_model = "origin_model/trace_module.pt"
output_model_dir = "./optimized_model"
optimize_config = OptimizeConfig()

optimize_config.pytorch.tidy.enable = False
optimize_config.pytorch.trt.enable_fp16 = True
optimize_config.pytorch.profilig.subproc = False
optimize_config.profilig.num_run = 50
optimize_config.profilig.num_warmup = 10

os.environ["TORCHTRT_DISABLE_FMHA"] = "1"
os.environ["TORCHTRT_DISABLE_FMHCA"] = "1"
model_config = ModelConfig()
model_config.pytorch.large_model=True

# change according to your image (h/8, w/8)
# change according to your model hidden size
model_config.pytorch.inputs_shape_range = [
    {"min": (1, 4, 64, 64), "opt": (2, 4, 64, 64), "max": (8, 4, 64,
64)},
    {"min": (1,), "opt": (2,), "max": (8,)},
```

```
    {"min": (1, 77, 768), "opt": (2, 77, 768), "max": (8, 77, 768)},
    {"min": (1, 1280, 8, 8), "opt": (2, 1280, 8, 8), "max": (8,
1280, 8, 8)},
    {"min": (1, 320, 64, 64), "opt": (2, 320, 64, 64), "max": (8,
320, 64, 64)},
    {"min": (1, 320, 64, 64), "opt": (2, 320, 64, 64), "max": (8,
320, 64, 64)},
    {"min": (1, 320, 64, 64), "opt": (2, 320, 64, 64), "max": (8,
320, 64, 64)},
    {"min": (1, 320, 32, 32), "opt": (2, 320, 32, 32), "max": (8,
320, 32, 32)},
    {"min": (1, 640, 32, 32), "opt": (2, 640, 32, 32), "max": (8,
640, 32, 32)},
    {"min": (1, 640, 32, 32), "opt": (2, 640, 32, 32), "max": (8,
640, 32, 32)},
    {"min": (1, 640, 16, 16), "opt": (2, 640, 16, 16), "max": (8,
640, 16, 16)},
    {"min": (1, 1280, 16, 16), "opt": (2, 1280, 16, 16), "max": (8,
1280, 16, 16)},
    {"min": (1, 1280, 16, 16), "opt": (2, 1280, 16, 16), "max": (8,
1280, 16, 16)},
    {"min": (1, 1280, 8, 8), "opt": (2, 1280, 8, 8), "max": (8,
1280, 8, 8)},
    {"min": (1, 1280, 8, 8), "opt": (2, 1280, 8, 8), "max": (8,
1280, 8, 8)},
    {"min": (1, 1280, 8, 8), "opt": (2, 1280, 8, 8), "max": (8,
1280, 8, 8)},
]

report = optimize_gpu(
    input_model,
    output_model_dir,
    test_data=gen_test_data(batch_size=2),
    optimize_config = optimize_config,
    model_config = model_config

)
```



**说明:**

优化过程中 “[INFO] ERROR: [Torch-TensorRT TorchScript Conversion Context] – 2: [virtualMemoryBuffer.cpp::resizePhysical::145] Error Code 2: OutOfMemory (no further information)” 提示可忽略。

**2. 优化完成后，会输出性能优化报告，格式如下：**

```
{
  "hardware": {
    "device": "NVIDIA A10, driver: 470.82.01",
    "driver": "470.82.01",
    "num_gpus": "1",
    "cpu": "AMD EPYC 7K83 64-Core Processor, family '25', model
'1'"
  },
  "software": {
    "taco version": "0.2.33",
    "framework": "pytorch",
    "framework version": "1.12.1+cu113",
    "torch device": "NVIDIA A10"
  },
  "summary": {
    "working directory": "/root/sd_webui_demo",
    "input model path": "origin_model/trace_module.pt",
    "output model folder": "./optimized_model",
    "input model format": "torch.jit saved (traced or scripted)
model",
    "status": "satisfactory",
    "baseline latency": "191ms 393us",
    "accelerated latency": "40ms 707us",
    "speedup": "4.70",
    "optimization time": "19min 39s 416ms",
    "env": "{}"
  }
}
```

**在 optimized\_model 目录中查看优化后的模型：**

```
[root@4e302835766c /root/demo]#ll optimized_model/
total 1.8G
drwxr-xr-x 2 500 500 4.0K Mar  3 14:38 ./
```

```
-rw-r--r-- 1 500 500 1.8G Mar 13 16:46
optimized_recursive_script_module.pt
drwxr-xr-x 7 500 500 4.0K Mar 13 16:47 ../
```

## （可选）模型验证

经过以上步骤，得到优化后的模型文件之后，您可以使用 `torch.jit.load` 接口加载该模型，验证其性能和正确性。加载模型运行的示例代码如下所示：

```
import torch
import taco
import os

taco_path = os.path.dirname(taco.__file__)
torch.ops.load_library(os.path.join(taco_path,
"torch_tensorrt/lib/libtorchtrt.so"))
optimized_model = torch.jit.load("optimized_recursive_script_module.pt")

pic = torch.rand(1, 4, 64, 64).cuda() // picture
timesteps = torch.tensor([1]*1) // timesteps
context = torch.randn(1, 77, 768) // text embedding

with torch.no_grad():
    output = optimized_model(pic, timesteps, context)
    print(output)
```

需要注意的是，由于优化后的模型包含经过高度优化的 TACO Kit 自定义算子，因此运行模型之前，需要执行 `import taco` 加载包含自定义算子的动态链接库。

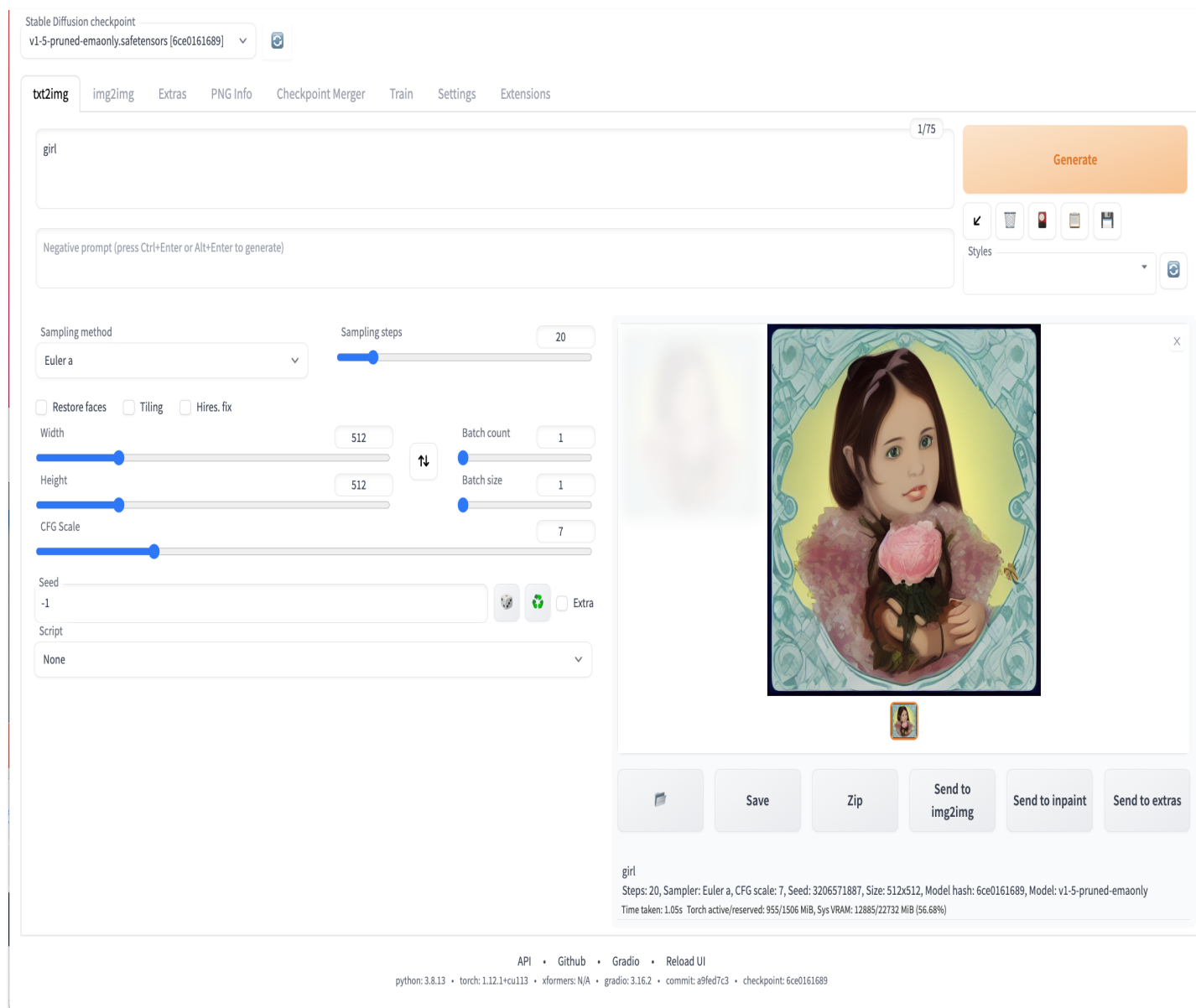
您根据自己的输出模型目录调整好相关参数之后，运行以上代码，即可加载优化后的模型进行推理计算。

## 启动优化后的模型

修改 `webui-user.sh` 脚本后，使用 `bash webui.sh` 命令启动 webUI 服务，将会自动加载 TACO Infer 优化后模型：

```
# edit webui-user.sh
# Uncomment to enable TACO opt UNetModel
export
TACO_OPT_UNET="/root/sd_webui_demo/optimized_model/optimized_recursive_s
cript_module.pt"

bash webui.sh
```



可以看到使用 TACO Infer 优化后，单张 A10 卡生成一张512 × 512图片时间仅为1秒左右。

## 总结

本文基于腾讯云 GPU 云服务器评测优化了 Stable diffusion 模型，通过TACO Infer的优化，在模型耗时主体结构 Unet 上获得了超过 4 倍的性能提升，端到端时延减少一半，助力业务性能大幅提升，吞吐率翻倍。

# TACO Infer 部署 Stable Diffusion web UI

最近更新时间：2024-04-03 14:11:21

## 操作场景

本文将演示如何使用 GPU 云服务器部署优化过的 AI 绘画模型 web UI 界面，结合 TACO Infer 的加速能力助力您获得 30% 以上的端到端性能提升。

## 操作步骤

### 购买 GPU 云服务器

购买实例，其中实例、存储及镜像请参见以下信息选择，其余配置请参见 [通过购买页创建实例](#) 按需选择。

- **实例：**选择 [计算型 PN4](#)。
- **系统盘：**配置容量不小于 200GB 的云硬盘。
- **镜像：**建议选择公共镜像。
- 操作系统使用 CentOS 7.9。
- 选择公共镜像后请勾选**后台自动安装GPU驱动**，实例将在系统启动后预装对应版本驱动。如下图所示：



#### ⚠ 注意：

当前优化版本仅支持 A10 GPU 机型，请检查您的实例配置。

## 安装docker和NVIDIA docker

1. 参见 [使用标准登录方式登录 Linux 实例](#)，登录实例。
2. 执行以下命令，安装 docker。

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

若您无法通过该命令安装，请尝试多次执行命令，或参见 Docker 官方文档 [Install Docker Engine](#) 进行安装。

#### ❗ 说明：

如果您使用非 root 用户身份安装 Docker，请参考 Docker 官方文档进行 [安装后配置](#)。

3. 执行以下命令，安装 nvidia-docker2。

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

若您无法通过该命令安装，请尝试多次执行命令，或参见 NVIDIA 官方文档 [Installation Guide & mdash](#) 进行安装。

## 下载 docker 镜像

执行以下命令，下载 docker 镜像，演示需要的所有数据和运行环境全部打包在 docker 镜像中。

```
docker pull taco-0.tencentcloudcr.com/taco_serving/sd_v1.5_demo:v4
```

模型下载完成，查看下载的 docker 镜像：

```
[root@VM-8-15-centos ~]# docker image ls
```

| REPOSITORY                                          | TAG | IMAGE ID     | CREATED      | SIZE   |
|-----------------------------------------------------|-----|--------------|--------------|--------|
| taco-0.tencentcloudcr.com/taco_serving/sd_v1.5_demo | v4  | 0f362817aebe | 19 hours ago | 19.7GB |

## 下载优化后的模型

执行以下命令，创建 optimized\_model 目录，下载模型后回到上层目录：

```
mkdir optimized_model && cd optimized_model
```

```
wget https://taco-1251783334.cos.ap-  
shanghai.myqcloud.com/demo/sd/unet/optimized_recursive_script_module.pt  
  
cd ..
```

## 启动 docker 镜像

如果您当前所处路径是/root/，执行以下命令，启动 docker 镜像。或者 -v 参数修改为 optimized\_model 所在目录。

```
docker run -it --rm --gpus=all --network=host -v /root:/data taco-  
0.tencentcloudcr.com/taco_serving/sd_v1.5_demo:v4 --listen
```

## 启动 Stable Diffusion web UI

复制 public URL 到浏览器，即可打开 webUI 界面。或者您可以选择将 local URL 的 0.0.0.0 替换成公网 IP，在浏览器打开页面。

```
Loading ldm.models.autoencoder.AutoencoderKL.  
Loading torch.nn.Identity.  
Loading ldm.modules.encoders.modules.FrozenCLIPEmbedder.  
Applying cross attention optimization (Doggettx).  
Textual inversion embeddings loaded(0):  
Model loaded in 7.7s (calculate hash: 2.9s, create model: 2.9s, apply weights to model: 0.1s, loa  
Running on local URL: http://0.0.0.0:7860  
Running on public URL: https://2916cc8b-319e-4b22.gradio.live  
  
This share link expires in 72 hours. For free permanent hosting and GPU upgrades (NEW!), check ou  
Startup time: 23.9s (import gradio: 4.8s, import ldm: 1.7s, other imports: 2.7s, setup codeformer
```

TACO Infer 优化过的模型单张图片处理时间约 1 秒，开源模型单张图片处理时间约 2 秒。

txt2img
img2img
Extras
PNG Info
Checkpoint Merger
Train
Settings
Extensions

a boy riding a dog

5/75

Generate

↩

🗑

🖨

📄

💾

Styles

↺

Sampling method

Euler a

Sampling steps

20

☐ Restore faces
☐ Tiling
☐ Hires. fix

Width

512

Height

512

Batch count

1

Batch size

1

CFG Scale

7

Seed

-1

🎲

🔄

☐ Extra

Script

None



📁

Save

Zip

Send to img2img

Send to inpaint

Send to extras

a boy riding a dog

Steps: 20, Sampler: Euler a, CFG scale: 7, Seed: 774759739, Size: 512x512, Model hash: 6ce0161689, Model: v1-5-pruned-emaonly

Time taken: 1.08s Torch active/reserved: 955/1506 MiB, Sys VRAM: 6912/22732 MiB (30.41%)

经过 TACO infer 优化，模型的前向推理能力提升 4 倍多，端到端性能提升 50%，由原来的约 2 秒缩短到 1 秒。