# 计算加速套件 TACO Kit

# TACO Infer AI 推理加速引擎

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2024 腾讯云版权所有

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

【商标声明】

腾讯云

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

【服务声明】

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

【联系我们】

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

# 文档目录

TACO Infer AI 推理加速引擎
- TACO Infer 概述
- TACO Infer 安装
- TACO Infer 接口
- 使用 TACO Infer 优化部署 Pytorch 模型
  - Pytorch 模型优化
  - Pytorch 模型推理部署
- 使用 TACO Infer 优化部署 TensorFlow 模型
  - TensorFlow 模型优化
  - TensorFlow 模型推理部署

版权所有：腾讯云计算（北京）有限责任公司

# TACO Infer AI 推理加速引擎
# TACO Infer 概述

最近更新时间：2024-08-08 16:56:41

## 背景信息

长期以来，AI 算法用于生产环境大规模模型的推理部署，都存在着多维度的考虑因素：

- 从企业的角度，需考虑如何选择硬件/基础设施来部署业务从而获得最佳投入产出比。例如，如何选择一款加速芯片，满足稳定运行业务模型、业务运行过程中硬件利用率、业务部署是否足够灵活、部署方案的可迁移性等。
- 从系统、测试和运维工程师的角度，面对开发社区算法的快速迭代，需考虑如何以不同框架训练的模型高效且稳定地部署来自上游业务算法，如何整合、利用社区及硬件厂商提供的加速库及软件，对接不同模型、不同硬件，如何将加速软件和自身业务的定制化需求有机结合。而面对不同业务细分场景，需考虑如何标准化一套接口高效地优化和部署业务模型等。

TACO Infer 在上述背景下，整合腾讯云及硬件厂商研发资源，面向 AI 负载倾力打造的异构加速引擎。帮助用户快速接入不同硬件，实现通用、易用的优化、加速、部署，避免一款硬件/加速芯片、一套方案的窘境，从而全面提高生产力。

## 功能介绍

TACO Infer 是一款是腾讯云异构计算团队推出的 AI 推理套件，旨在为客户提供简洁，高效的推理优化一键式解决方案。TACO Infer 从整体设计上看，像是一个大编译器，通过对模型计算图的精细探查，编译最优的执行方案。用户无需自行整合整合利用社区、厂商加速技术，TACO Infer 帮助您简洁、无侵入业务代码地一站式解决 AI 模型在生产环境中应用的问题，方便客户聚焦于自身业务的商业成功。TACO Infer 具备以下功能特点：

### 产品特性

- 简洁：TACO 的部署非常简单，客户不需要做过多的代码改动即可嵌入到业务代码中。
- 高效：TACO 采用自研方案，相比于业界竞品性能提升明显，充分释放硬件算力。
- 灵活：TACO 支持在私有化和公有云部署，具备灵活的产品形态。
- 稳定：TACO 在集团内、外部都积累大量落地案例，线上服务高效稳定。

### 软硬件兼容能力

- TACO-Infer 兼容 TensorRT，提供了多种精度量化方案，允许客户在推理速度和准确度之间权衡。
- TACO-Infer 兼容 OpenVINO，支持 CPU 侧推理部署，及 CPU/GPU 协同优化方案。
- TACO-Infer 支持 Tensorflow 及 Pytorch 开发的模型，能够最大限度客户减少业务的迁移成本。
- TACO-Infer 支持 cuBLAS 和手工调优的 kernel，能够根据不同的性能场景自动选择不同的优化方案。

除此之外，TACO-Infer 还可运行在虚拟机、物理机、容器等各种环境。

版权所有：腾讯云计算（北京）有限责任公司

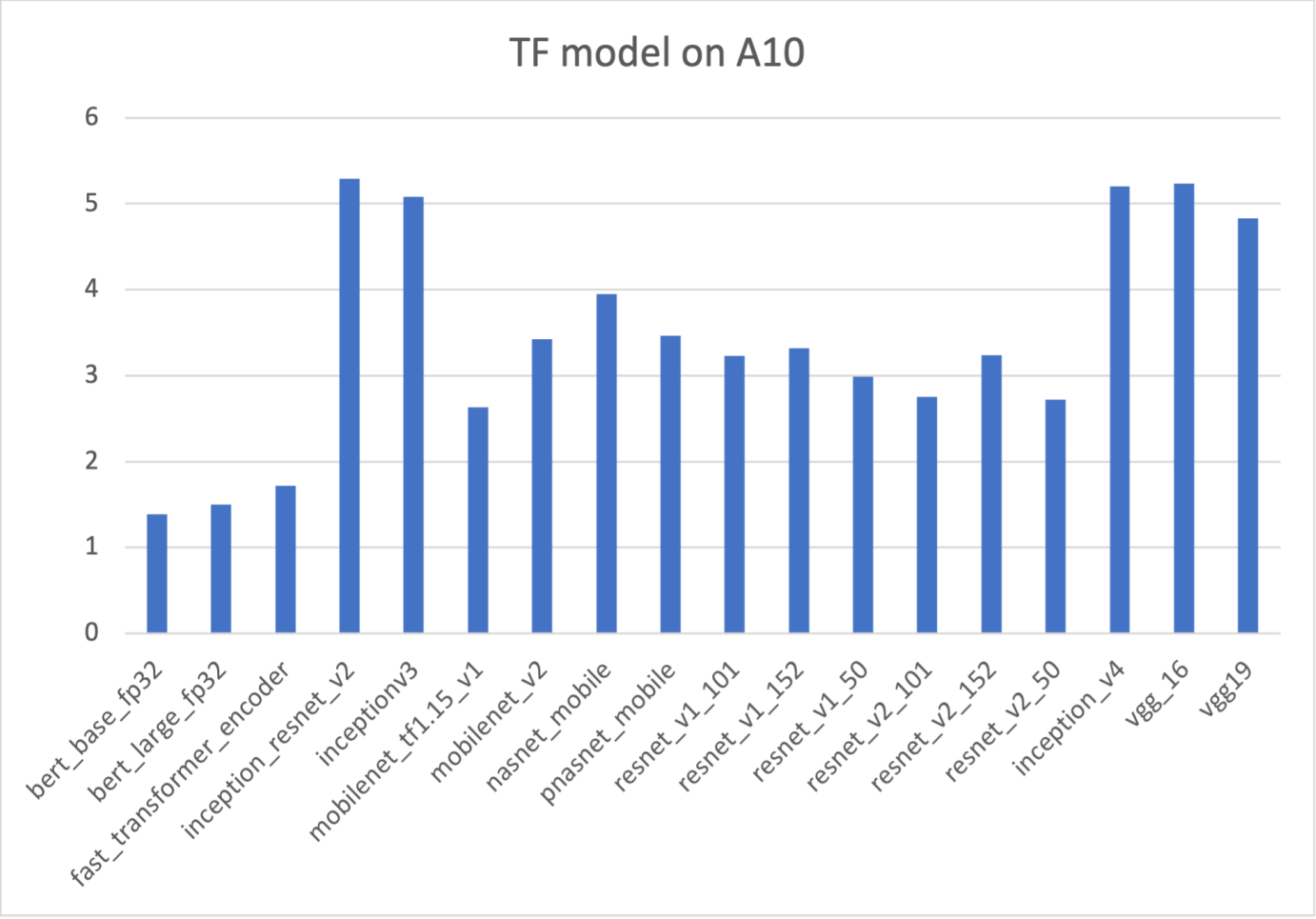
## 集成硬件厂商定向开源的加速方案

- 厂商加速方案集成。
- 编译优化。
- 图优化和算子优化。

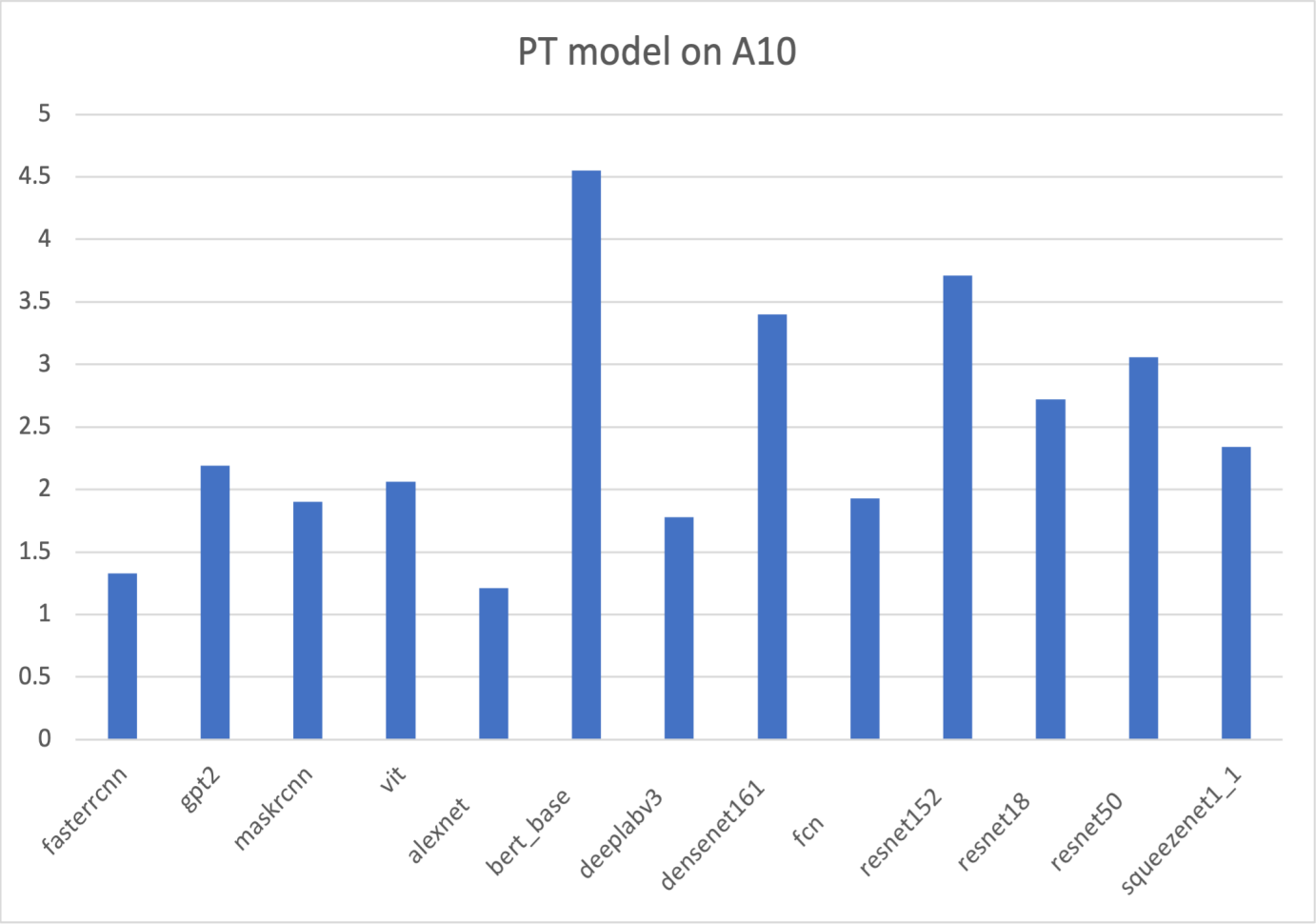
## 加速效果

经过 TACO Infer 优化后的模型运行时间与优化前模型运行时间对比，加速效果显著。以下为搭载单卡 NVIDIA A10 的 PNV4 机型 在 batch_size=1 配置下不同深度学习框架的加速比：

- Tensorflow 2.6




- Pytorch 1.12




## 使用说明

TACO Infer 支持虚拟机、物理机、容器等各种环境，目前已支持 TensorFlow 模型的 CPU 优化部署。TACO Infer 的推理优化部署有以下几个步骤：

1. **硬件选型和环境配置：**

   关于计算环境的选型和配置推荐请参见 TACO Infer 使用环境要求，软件安装请参见 安装 TACO Infer 操作步骤。

2. **使用 TACO Infer 生成优化模型：**

   您需要使用 Python Wheel 包生成优化的模型，具体操作请参见 TensorFlow 模型优化，优化模型将会存放于您指定的路径，与输入模型保持同一格式，加速效果您可通过 调用优化接口 查看。

3. **部署 TACO Infer 优化模型：**

   在完成模型的优化后，您需要将模型部署在生产环境，TACO Infer 提供 SDK 包部署，部署流程请参见 TensorFlow 模型推理部署。按照流程部署编译完成后，运行即可加载优化后的模型进行推理计算。

# TACO Infer 安装

最近更新时间：2024-09-24 11:10:21

## 操作场景

本文介绍如何在服务器上安装 TACO Infer。TACO Infer 的安装包包括 Python Wheel 包和推理 SDK 包两个部分。Python Wheel 包用于在具有目标加速芯片的机器环境中对模型进行优化，SDK 则用于 C++开发模型推理部署。

## 使用环境要求

| 具体项 | 说明 |
| --- | --- |
| 操作系统 | TACO Infer 针对 CentOS 7.9进行了全面的测试，建议您优先选用 CentOS 7.9进行 TACO Infer 的使用和部署。目前仅支持 Linux 操作系统，如使用其他版本 Linux 操作系统遇到问题，欢迎联系我们获取支持。 |
| 计算设备 | TACO Infer 目前开放测试的版本支持 Intel/AMD 系列的 CPU 和 NVIDIA 系列的 GPU 优化，后续会支持更多优化目标硬件，请您关注产品动态。推荐适配的腾讯云机型如下：<br>• CPU机型<br>  ○ 云服务器 CVM：标准型 S6，SA3 等。更多实例信息请参见 云服务器实例规格 。<br>  ○ 裸金属服务器：高 IO 型 BMI5，标准型 BMSA2 等。更多实例信息请参见 裸金属服务器实例规格 。<br>• GPU机型<br>  ○ 云服务器 CVM：计算型 GT4, 计算型 PNV4, 计算型 GN10Xp 等。更多实例信息请参见 GPU计算型实例 。<br>  ○ 高性能计算集群：GPU 型 HCCPNV4h，GPU 型 HCCG5v 等。更多实例信息请参见 高性能计算集群实例规格 。 |
| ABI 版本 | TACO Infer 支持 CXX11 ABI。 |
| Python 版本 | TACO Infer 支持 Python 3.6 及以上版本，建议您安装支持版本的 Python环境。 |
| 框架版本 | TACO Infer支持TensorFlow、Pytorch、Onnx模型优化。我们主要支持的框架版本有：<br>• Tensorflow：1.14, 1.15, 2.4, 2.6<br>• Pytorch：1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.12<br>• Onnx：1.10<br>如需要使用其他版本，欢迎联系我们获取支持。 |

## 操作步骤

## 安装深度学习框架

使用 TACO Infer 时，需要您的 Python 运行环境中已安装深度学习框架。以 Tensorflow 1.14 为例：

```
pip install tensorflow==1.14
```

> **注意：**
> TACO Infer 不代替您安装深度学习框架。使用 TACO Infer 时，请确保您的 Python 运行环境中已安装（pip install）相应的深度学习框架(Tensorflow/Pytorch)。TACO Infer会自动适配您所选择的深度学习框架版本。

## 依赖安装

Taco Infer 依赖 libcurl、openssl、libuuid。您可以通过以下命令检查是否已经安装这些组件：

```
yum list installed | grep ${package}
```

若未安装，请通过以下命令安装。

- libcurl

  ```
  yum install libcurl-devel
  ```

- openssl

  ```
  yum install openssl-devel
  ```

- libuuid

  ```
  yum install libuuid-devel
  ```

## 获取 Wheel 包及 SDK 包

填写 TACO Infer 调查问卷，评估通过后可获得 TACO Infer Python Wheel 包和 SDK 安装包。目前支持计算环境和框架请参见 使用环境要求，获取更多资讯可以 联系我们。

## 安装 Wheel 包

通过 pip 命令，即可安装 Taco python 包：

```
pip install ${path/to/wheel_package}
```

## 获取 SDK 包

建议将解压后的库文件拷贝到 `/usr/local/lib` 下，以便 ld 程序能够找到 Taco 动态链接库进行链接。

对于 CPU 设备，SDK包中存在以下文件:

- libtaco_tf.so
- libtidy_ops.so
- libomp-1fdec59b.so
- tidy_vm

对于 GPU 设备，SDK包中存在以下文件:

- libtidy_runtime.so
- libtacaudio_ops.so
- lib_tactorch.so
- libnvinfer.so
- libnvinfer_plugin.so
- libnvonnxparser.so
- libnvparsers.so
- libnvinfer_builder_resource.so.8.5.1

SDK包中所包含的文件会随您选择的深度学习框架版本和硬件设备有所区别，实际包含内容以发布的SDK包为准。

版权所有：腾讯云计算（北京）有限责任公司

# TACO Infer 接口

最近更新时间：2024-09-24 11:10:21

## 优化接口

TACO Infer 为您提供了一套简单易用的模型优化接口。
对于 CPU 计算设备，优化接口为 `optimize_cpu`：

```
def optimize_cpu(
    input_model: Union[str, torch.nn.Module],
    output_model_dir: str,
    test_data: Optional[Dict[str, np.array]],
    optimize_config: OptimizeConfig = OptimizeConfig(),
    model_config: ModelConfig = ModelConfig(),
) -> Dict:
```

对于GPU计算设备，优化接口为 `optimize_gpu`：

```
def optimize_gpu(
    input_model: Union[str, torch.nn.Module],
    output_model_dir: str,
    test_data: Optional[Dict[str, np.array]] = None,
    optimize_config: OptimizeConfig = OptimizeConfig(),
    model_config: ModelConfig = ModelConfig(),
) -> Dict:
```

两种设备的优化接口输入参数完全一致。

## 输入参数说明

| 参数 | 是否必选 | 说明 |
| --- | --- | --- |
| input_model | 必选 | 待优化的模型。对于 TensorFlow，该参数为模型文件所在路径。TACO Infer 支持 TensorFlow frozen pb 和 saved model 两种模型格式。对于 Pytorch，该参数可以是 torchscript 模型文件所在路径，也可以是 torch.nn.Module 对象实例。 |
| output_model_dir | 必选 | 优化后模型的保存目录。 |

| | | |
|---|---|---|
| test_data | 必选 | 优化过程中需要使用到的模型输入的测试数据。TACO Infer 在优化模型的过程中需要使用测试数据对模型的性能，精度等指标进行评估，以指导模型优化过程。对于 TF 模型，该参数为 session run 所需的 feed_dict。<br>需注意，test_data 只接受 numpy array 数据格式。对于Pytorch 模型，该参数为模型 forward 方法或者用户自定义前向函数所需输入，此时 test data 仅接受包含 torch.Tensor 的 tuple，或 torch.Tensor（如果只有一个输入）。<br>构建 test_data 的方式示例如下：<br>`import numpy as np`<br>`def gen_test_data(batch_size = 1):`<br>`INPUT_NAME = "input:0"`<br>`image_size = 299`<br>`input_data = np.random.rand(batch_size, image_size, image_size, 3)`<br>`return {INPUT_NAME: input_data}` |
| optimize_config | 可选 | 优化配置。您可以通过它指导 TACO Infer 提供更高质量的优化：<br>通过 `print(optimize_config)` 可以查看默认（或修改后的）配置。<br>通过 `print(optimize_cfg.help())` 了解有哪些可配置项及如何配置。 |
| model_config | 可选 | 模型配置。例如，对于存在1个以上 signature 的 TF SavedModel，您可以通过配置 model_config。<br>知会 TACO Infer 哪一个需要被优化：<br>通过 `print(model_config)` 可以查看默认（或修改后的）配置。<br>通过 `print(model_cfg.help())` 了解有哪些可配置项及如何配置。示例如下：<br>`print(model_cfg.help())`<br>`How-to-assign-a-"model_config":`<br>`tensorflow.inputs:`<br>`type: <class 'list'>, default value: None`<br>`Input tensor names as a list. Items may use`<br>`node name as prefix, such as "Placeholder:0",`<br>`or saved model signature.` |

| | | |
|---|---|---|
| | | tensorflow.outputs:<br>type: <class 'list'>, default value: None<br>Output tensor names as a list.<br><br>tensorflow.saved_model.signature:<br>type: <class 'str'>, default value: None<br>Tell TACO Inf which signature to use if more than 1 signature.<br><br>Example of updating a config:<br>model_config.parse({"tensorflow.inputs": ['Placeholder:0']}) |

## 输出参数说明

**优化模型后会产生一个 JSON 格式的优化报告，该报告包含了优化模型的硬件，软件以及一些总结信息。输出参数如下所示：**

```
{
    "hardware": {
        "cpu": "AMD EPYC 7K62 48-Core Processor, family '23', model '49'",
        "target device": "AMD EPYC 7K62 48-Core Processor, family '23', model '49'",
        "reference": "https://en.wikichip.org/wiki/intel/cpuid"
    },
    "software": {
        "framework": "tensorflow",
        "framework version": "1.15.0"
    },
    "summary": {
        "working_directory": "/root/taco_test/fast_transformer_encoder",
        "input_model": "./model/fast-transformer-encoder.pb",
        "output_model_dir": "./optimized_model",
        "optimization time": "3min 46s 398ms",
        "model format": "tensorflow frozen pb",
        "status": "satisfactory",
        "baseline latency": "49ms 517us",
        "accelerated latency": "27ms 12us",
        "speedup": "1.83"
    }
```

```
}
```

输出字段说明如下：

- **hardware**：硬件环境信息，包括设备类型、规格等。
- **software**：软件环境信息，包括框架以及框架版本。
- **summary**：模型优化的综合性信息，包括当前工作目录、输入模型路径、输出模型目录、优化时间、模型格式、运行状态、模型优化效果等。

## Config 使用方式

`OptimizeConfig` 和 `ModelConfig` 使用方法是一致的。均支持通过 `parse` 接口设置相关属性的值，通过 " `.` " 获取属性值或者赋值。Config 使用示例如下所示：

```
from taco import ModelConfig


cfg = ModelConfig()

# assign by parse()
cfg.parse({"tensorflow.inputs": ['Placeholder:0']})

print(cfg.tensorflow.inputs)

# assign by .
cfg.tensorflow.saved_model.signature = "predictions"

print(cfg.tensorflow.saved_model.signature)
```

# 使用 TACO Infer 优化部署 Pytorch 模型

## Pytorch 模型优化

最近更新时间：2024-09-24 11:10:21

### 模型准备

TACO Infer 支持对 Pytorch TorchScript 和 torch.nn.Module 两种模型格式进行优化。通常在生产环境中，性能最优的方式是导出 TorchScript 模型后进行部署。TorchScript 模型也是 TACO Infer 支持最完善的模型格式，推荐您优先使用 TorchScript 模型格式。在优化前，您需要准备好导出后的 TorchScript 模型。模型导出示例代码如下：

```
import pathlib

import torch
from torchvision.models.resnet import Bottleneck, ResNet

from taco.utils.network import wget_url


class ImageClassifier(ResNet):
    def __init__(self):
        super(ImageClassifier, self).__init__(Bottleneck, [3, 8, 36, 3])
        self.ckpt_url = "https://taco-1251783334.cos.ap-
shanghai.myqcloud.com/model/pytorch/checkpoints/resnet152/resnet152-
b121ed2d.pth"


def gen_model(work_dir: str) -> torch.nn.Module:
    model = ImageClassifier()
    model_path = pathlib.Path(work_dir) / "model.pth"
    wget_url(model.ckpt_url, model_path, disable_bar=False)
    model.load_state_dict(torch.load(model_path))
    return model


model = gen_model(".")
script_model = torch.jit.script(model)
scirpt_model_path = "./model.pt"
torch.jit.save(script_model, scirpt_model_path)
```

## 导入TACO Infer

使用 TACO Infer 优化模型首先需要导入 python 模块:

```
from taco import optimize_gpu, OptimizeConfig, ModelConfig
```

## 调用优化接口

配置好输入模型，输出模型目录，测试数据，优化配置，模型配置之后，调用 optimize_gpu 即可对模型进行优化。关于优化接口参数的详细信息，请参见 TACO Infer 接口 。

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

模型优化结束后，会产出一个保存在您指定的目录中的优化后的模型，以及一个包括硬件信息，软件信息及优化过程相关指标的优化报告。优化报告的详细信息如以下样例所示:

```
{
    "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": "0.2.10",
        "framework": "pytorch",
        "framework version": "1.12.0+cu113",
        "torch device": "NVIDIA A10"
    },
    "summary": {
        "working directory": "/root/resnet152",
        "input model": "ImageClassifier",
        "output model folder": "optimized_dir",
        "input model format": "torch.nn.Module memory object",
        "status": "satisfactory",
        "baseline latency": "20ms 102us",
```

```
        "accelerated latency": "5ms 418us",
        "speedup": "3.71",
        "optimization time": "1min 2s 542ms",
        "env": "{}"
    }
}
```

关于优化报告的详细字段说明，请参见 TACO Infer 接口 。

完整的优化示例代码如下：

```
import torch

from taco.optimizer.optimize import optimize_gpu
from taco import ModelConfig
from taco import OptimizeConfig


def gen_test_data(batch_size: int = 1) -> torch.Tensor:
    IMAGE_SIZE=224
    return torch.rand(batch_size, 3, IMAGE_SIZE, IMAGE_SIZE)


# the path of torchscript model exported in previous chapters
scirpt_model_path = "./model.pt"
test_data = gen_test_data(batch_size=1)
optim_cfg = OptimizeConfig()
model_cfg = ModelConfig()

report = optimize_gpu(input_model=scirpt_model_path,
                      output_model_dir="optimized_dir",
                      test_data=test_data,
                      optimize_config=optim_cfg,
                      model_config=model_cfg)
```

优化完成后，在配置好的模型输出目录，可以看到产出的优化模型：

```
[root@60abf692a8a1 /root/resnet152]
#ll optimized_dir/
total 454M
drwxr-xr-x 3 root root 4.0K Jan  5 15:30 ../
drwxr-xr-x 2 root root 4.0K Jan  5 15:30 ./
```

```
-rw-r--r-- 1 root root 454M Jan  5 15:30
optimized_recursive_script_module.pt
```

可以看到，TACO 优化模型后会产出 TorchScript 格式的模型文件供您进行部署。

## 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。加载模型运行的代码如下所示：

```
import torch
import taco


def gen_test_data(batch_size: int = 1) -> torch.Tensor:
    IMAGE_SIZE=224
    return torch.rand(batch_size, 3, IMAGE_SIZE, IMAGE_SIZE)


optimized_model =
torch.jit.load("optimized_dir/optimized_recursive_script_module.pt")
test_data = gen_test_data(batch_size=1).cuda()

with torch.no_grad():
    output = optimized_model(test_data)
    print(output.shape)
```

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。输出日志如下：

```
[root@a2c4f3d901e6 /root/resnet152]
#python infer.py
torch.Size([1, 1000])
```

可以看到，模型正常运行并且输出了计算结果。

# Pytorch 模型推理部署

最近更新时间：2024-09-24 11:10:21

使用TACO Infer产出优化模型，并且验证模型的性能和正确性符合预期之后，接下来就可以将模型部署在实际生产环境中了。

## 环境准备

- 服务器准备：参见 TACO Infer 安装，选购 GPU 机型。
- ABI 版本：对于 Pytorch 深度学习框架，TACO Infer 对官方发布的 CXX11 ABI 和 Pre-CXX11 ABI 两个版本的 libtorch 库均进行了支持。
- SDK 包安装：在开发部署模型之前，您需要 联系我们 获得 TACO Kit SDK 安装包，然后解压，可以看到安装包中包含多个动态链接库：

```
[root@60abf692a8a1 (Taco Dev) /root/resnet152]
#ll 1.12.0-lib/
total 1.4G
-rwxr-xr-x 1 root root 3.3M Jan  5 16:10 libnvparsers.so*
-r-xr-xr-x 1 root root  11M Jan  5 16:10 libtacaudio_ops.so*
-r-xr-xr-x 1 root root  19M Jan  5 16:10 libtidy_runtime.so*
-rwxr-xr-x 1 root root 357M Jan  5 16:10
libnvinfer_builder_resource.so.8.5.1*
-rwxr-xr-x 1 root root 465M Jan  5 16:10 libnvinfer.so.8*
-rwxr-xr-x 1 root root 3.3M Jan  5 16:10 libnvparsers.so.8*
lrwxrwxrwx 1 root root   12 Jan  5 16:10 lib_tactorch.so ->
_tactorch.so*
-rwxr-xr-x 1 root root 465M Jan  5 16:10 libnvinfer.so*
-rwxr-xr-x 1 root root  42M Jan  5 16:10 libnvinfer_plugin.so*
-rwxr-xr-x 1 root root  42M Jan  5 16:10 libnvinfer_plugin.so.8*
-r-xr-xr-x 1 root root  14M Jan  5 16:10 _tactorch.so*
-rwxr-xr-x 1 root root 2.8M Jan  5 16:10 libnvonnxparser.so.8*
-rwxr-xr-x 1 root root 2.8M Jan  5 16:10 libnvonnxparser.so*
drwxr-xr-x 2 root root 4.0K Jan  5 16:10 ./
drwxr-xr-x 5 root root 4.0K Jan  5 16:39 ../
```

您可以将所有 TACO 库文件拷贝到系统库目录 `/usr/lib` ，以便链接器ld进行链接的时候能够找到它们。
或者您也可以将 TACO 库文件拷贝到其他路径，并在 `LD_LIBRARY_PATH` 环境变量中添加库所在路径。
请确保所有的 TACO 库文件位于同一路径下。

## 推理代码开发

下面以 Torch C++ API 展示优化后模型的加载运行过程：

```
#include <torch/script.h>

#include <iostream>
#include <memory>
#include <chrono>


int main(int argc, const char* argv[]) {
  // set this env var outside of TencentCloud.
  setenv("TACO_TRIAL_RUN", "true", 1);

  torch::jit::Module module;
  try {
    // Deserialize the ScriptModule from a file using
torch::jit::load().
    module =
torch::jit::load("optimized_dir/optimized_recursive_script_module.pt");
  } catch (const c10::Error& e) {
    std::cerr << "error loading the model\n";
    return -1;
  }

  torch::Tensor input = torch::randn({1, 3, 224,
224}).to(torch::Device(torch::kCUDA, 0));

  std::vector<torch::jit::IValue> inputs;
  inputs.push_back(input);

  // warmup
  for (int i=0; i<10; i++) {
    module.forward(inputs);
  }

  auto start_time = std::chrono::high_resolution_clock::now();
  for (int i=0; i<10; i++) {
    module.forward(inputs);
  }
  auto end_time = std::chrono::high_resolution_clock::now();
  auto duration = std::chrono::duration_cast<std::chrono::microseconds>(
      end_time - start_time);
```

```
  std::cout << "Duration of resnet152 is: "
            << double(duration.count()) / 10000 << "ms" << std::endl;

  torch::Tensor output_tensor =
module.forward(inputs).toTensor().to(torch::Device(torch::kCPU));

  auto output_a = output_tensor.accessor<float, 2>();
  for(int i = 0; i < 10; i++) {
    std::cout << output_a[0][i] << std::endl;
  }
}
```

如以上代码所示，您只需要按照标准的 Torch C++ API 加载经过优化的模型即可，和加载普通的 TorchScript 模型没有任何区别。

## 编译链接

编译以上代码的时候，需要链接 Taco 提供的两个动态库：`_tactorch.so` 和 `tidy_runtime.so`，此外还需要链接 Pytorch 的相关动态链接库。
编译脚本示例如下所示：

```
#!/bin/bash

TORCH_LIB_PATH=/root/venv/taco_dev_pt1.12/lib/python3.8/site-
packages/torch/lib

gcc -std=c++14 \
    -I./1.12.0-inc/include \
    -I./1.12.0-inc/include/torch/csrc/api/include \
    -L./1.12.0-lib -l_tactorch -ltidy_runtime \
    -L$TORCH_LIB_PATH -ltorch -ltorch_cpu -lc10 \
    -lstdc++ \
    -o infer infer.cc
```

编译完成之后，即得到一个可运行的二进制文件：

```
[root@60abf692a8a1 /root/resnet152]
#ll

-rw-r--r--  1 root root 1.4K Jan  5 16:39 infer.cc
-rwxr-xr-x  1 root root 495K Jan  5 16:39 infer*
drwxr-xr-x 28 root root 4.0K Jan  5 16:39 ../
```

```
drwxr-xr-x  5 root root 4.0K Jan  5 16:39 ./
```

## 推理计算

部署程序编译完成之后，运行便可以加载优化后的模型并进行推理计算：

```
[root@60abf692a8a1 (Taco Dev) /root/resnet152]
#./infer

Duration of resnet152 is: 5.330ms
0.149882
1.17847
0.698387
-0.326189
-0.0360627
-0.938206
-0.905457
-0.272352
-0.280662
-2.14997
```

可以看到，模型正常加载运行，并输出了推理计算结果。

# 使用 TACO Infer 优化部署 TensorFlow 模型

# TensorFlow 模型优化

最近更新时间：2024-09-24 11:10:21

## 操作场景

本文介绍如何使用 TACO Infer 优化模型。

> **说明：**
> 请确保您已经完成相关环境的安装配置，详情请参见 安装 TACO Infer 。

## 操作步骤

### 准备模型

TACO Infer 支持 TensorFlow frozen pb 和 saved model 两种主要的模型格式。在模型优化前，需在磁盘上导出模型文件。您可以自行训练导出支持格式的模型或下载公开的网络模型。导出模型示例代码如下：

```
import tensorflow as tf


def export_frozen_pb(save_dir, file_name):
  g = tf.Graph()
  with g.as_default():
      px = tf.placeholder(shape=shape, dtype=dtype, name="px")
      py = tf.placeholder(shape=shape, dtype=dtype, name="py")
      c = tf.constant(random_data(positive_constant), name="c")
      x = px + py
      x = x * x
      x = tf.nn.relu(x)
      x = tf.abs(x)
      x = x + c
      y = tf.rsqrt(x)
      z = tf.nn.relu(x)
      tf.io.write_graph(g.as_graph_def(), save_dir, file_name)
```

### 导入 TACO Infer

使用 TACO Infer 优化模型首先需要导入 python 模块，代码如下：

```
from taco import optimize_cpu, OptimizeConfig, ModelConfig
```

## 调用优化接口

1. 配置输入模型、输出模型目录、测试数据、优化配置、模型配置后，调用 optimize_cpu 即可对模型进行优化。关于优化接口参数的详细信息，请参见 接口文档 。优化接口代码如下：

```
report = optimize_cpu(
 input_model,
 output_model_dir,
 test_data = test_data,
 optimize_config = optimize_config,
 model_config = model_config
)
```

2. 优化过程中，可以看到如下类似的优化日志：

```
2022-03-31 16:54:19,581 [INFO] [optimize_model.py:27] Load frozen pb
from /root/taco_test/model/fast-transformer-encoder.pb
2022-03-31 16:54:40,213 [INFO] [optimize_model.py:27] Matched 12
[softmax_pattern] in the graph.
2022-03-31 16:54:52,545 [INFO] [optimize_model.py:27] Matched 0
[conv2_d_pattern] in the graph.
2022-03-31 16:54:57,316 [INFO] [optimize_model.py:27] Matched 72
[mat_mul_pattern] in the graph.
2022-03-31 16:55:05,606 [INFO] [optimize_model.py:27] Matched 0
[avg_pool_pattern] in the graph.
2022-03-31 16:55:10,354 [INFO] [optimize_model.py:27] Matched 0
[max_pool_pattern] in the graph.
2022-03-31 16:55:15,088 [INFO] [optimize_model.py:27] Matched 0
[mat_mul_bias_add_relu_pattern] in the graph.
2022-03-31 16:55:19,952 [INFO] [optimize_model.py:27] Matched 72
[mat_mul_bias_add_pattern] in the graph.
2022-03-31 16:55:28,033 [INFO] [optimize_model.py:27] Matched 0
[conv2_d_bias_add_relu_pattern] in the graph.
2022-03-31 16:55:32,676 [INFO] [optimize_model.py:27] Matched 0
[conv2_d_bias_add_pattern] in the graph.
2022-03-31 16:55:37,246 [INFO] [optimize_model.py:27] Matched 0
[conv2_d_fused_batch_norm_relu_pattern] in the graph.
```

版权所有：腾讯云计算（北京）有限责任公司

```
2022-03-31 16:55:41,781 [INFO] [optimize_model.py:27] Matched 0
[conv2_d_fused_batch_norm_pattern] in the graph.
```

3. 模型优化结束后，会产出一个保存在您指定的目录中的优化后的模型，以及一个包括硬件信息，软件信息及优化过程相关指标的优化报告。关于优化报告的详细字段说明，请参见 输出参数说明 。优化报告的详细信息如以下样例所示：

```
{
 "hardware": {
     "cpu": "AMD EPYC 7K62 48-Core Processor, family '23', model
'49'",
     "target device": "AMD EPYC 7K62 48-Core Processor, family '23',
model '49'",
     "reference": "https://en.wikichip.org/wiki/intel/cpuid"
 },
 "software": {
     "framework": "tensorflow",
     "framework version": "1.15.0"
 },
 "summary": {
     "working_directory": "/root/taco_test/optimize",
     "input_model": "./model/fast-transformer-encoder.pb",
     "output_model_dir": "./optimized_model",
     "optimization time": "3min 43s 902ms",
     "model format": "tensorflow frozen pb",
     "status": "satisfactory",
     "baseline latency": "95ms 380us",
     "accelerated latency": "46ms 892us",
     "speedup": "2.03",
 }
}
```

优化完成后，在配置好的模型输出目录，可查看如下所示产出的优化模型。TACO Infer 优化模型后会产出一个和输入模型保持同一格式的模型供您进行部署。

```
[root@VM-3-46-centos optimized_model]# ll
total 332440
-rw-r--r-- 1 root root 340411615 Mar 31 21:03 fast-transformer-
encoder.pb
```

## 问题调试

如果在使用 TACO Infer 过程中遇到问题，您还可以在进行模型优化前设置以下环境变量，导出详细的日志信息到指定的日志文件中：

```
export TACO_LOG_FILE=path/to/log_file
export TACO_LOG_LEVEL=debug
```

例如，将日志信息导出到当前工作目录下的 `taco_log` 文件中：

```
[root@VM-3-46-centos fast_transformer_encoder]# export
TACO_LOG_FILE=./taco_log
[root@VM-3-46-centos fast_transformer_encoder]# export
TACO_LOG_LEVEL=debug
[root@VM-3-46-centos fast_transformer_encoder]# python optimize_model.py
```

优化完成之后，可以看到当前工作目录下生成了一个名为 `taco_log` 的日志文件和一个名为 `taco_log.trace` 的 trace 文件：

```
[root@VM-3-46-centos fast_transformer_encoder]# ll
-rw-r--r-- 1 root root 86388 Apr 14 14:23 taco_log
-rw-r--r-- 1 root root 41358 Apr 14 14:23 taco_log.trace
drwxr-xr-x 2 root root  4096 Mar 31 18:07 model
drwxr-xr-x 2 root root  4096 Apr 14 14:23 optimized_model
-rw-r--r-- 1 root root   674 Mar 31 18:09 optimize_model.py
```

- `taco_log.trace` 文件：包含一些编码后的 trace 信息。
- `taco_log` 文件：包含明文的详细日志信息，内容如下所示：

```
...
2022-04-14 14:20:17,351 [DEBUG] [optimize_model.py:24]
zen_softmax_rewrite receives 1 input models
2022-04-14 14:20:17,365 [INFO] [optimize_model.py:24] Matched 12
[softmax_pattern] in the graph.
2022-04-14 14:20:17,746 [DEBUG] [optimize_model.py:24] Matched
original nodes: ['layer_0/attention/self/Softmax']
2022-04-14 14:20:17,746 [DEBUG] [optimize_model.py:24] Origin root
node name of graph segment: layer_0/attention/self/Softmax
2022-04-14 14:20:17,749 [DEBUG] [optimize_model.py:24] Matched
original nodes: ['layer_1/attention/self/Softmax']
2022-04-14 14:20:17,749 [DEBUG] [optimize_model.py:24] Origin root
node name of graph segment: layer_1/attention/self/Softmax
```

版权所有：腾讯云计算（北京）有限责任公司

```
2022-04-14 14:20:17,751 [DEBUG] [optimize_model.py:24] Matched
original nodes: ['layer_10/attention/self/Softmax']
2022-04-14 14:20:17,751 [DEBUG] [optimize_model.py:24] Origin root
node name of graph segment: layer_10/attention/self/Softmax
2022-04-14 14:20:17,754 [DEBUG] [optimize_model.py:24] Matched
original nodes: ['layer_11/attention/self/Softmax']
2022-04-14 14:20:17,754 [DEBUG] [optimize_model.py:24] Origin root
node name of graph segment: layer_11/attention/self/Softmax
2022-04-14 14:20:17,757 [DEBUG] [optimize_model.py:24] Matched
original nodes: ['layer_2/attention/self/Softmax']
2022-04-14 14:20:17,757 [DEBUG] [optimize_model.py:24] Origin root
node name of graph segment: layer_2/attention/self/Softmax
2022-04-14 14:20:17,760 [DEBUG] [optimize_model.py:24] Matched
original nodes: ['layer_3/attention/self/Softmax']
2022-04-14 14:20:17,760 [DEBUG] [optimize_model.py:24] Origin root
node name of graph segment: layer_3/attention/self/Softmax
2022-04-14 14:20:17,762 [DEBUG] [optimize_model.py:24] Matched
original nodes: ['layer_4/attention/self/Softmax']
2022-04-14 14:20:17,762 [DEBUG] [optimize_model.py:24] Origin root
node name of graph segment: layer_4/attention/self/Softmax
2022-04-14 14:20:17,765 [DEBUG] [optimize_model.py:24] Matched
original nodes: ['layer_5/attention/self/Softmax']
2022-04-14 14:20:17,765 [DEBUG] [optimize_model.py:24] Origin root
node name of graph segment: layer_5/attention/self/Softmax
2022-04-14 14:20:17,768 [DEBUG] [optimize_model.py:24] Matched
original nodes: ['layer_6/attention/self/Softmax']
2022-04-14 14:20:17,768 [DEBUG] [optimize_model.py:24] Origin root
node name of graph segment: layer_6/attention/self/Softmax
2022-04-14 14:20:17,770 [DEBUG] [optimize_model.py:24] Matched
original nodes: ['layer_7/attention/self/Softmax']
2022-04-14 14:20:17,771 [DEBUG] [optimize_model.py:24] Origin root
node name of graph segment: layer_7/attention/self/Softmax
2022-04-14 14:20:17,773 [DEBUG] [optimize_model.py:24] Matched
original nodes: ['layer_8/attention/self/Softmax']
2022-04-14 14:20:17,773 [DEBUG] [optimize_model.py:24] Origin root
node name of graph segment: layer_8/attention/self/Softmax
2022-04-14 14:20:17,776 [DEBUG] [optimize_model.py:24] Matched
original nodes: ['layer_9/attention/self/Softmax']
2022-04-14 14:20:17,776 [DEBUG] [optimize_model.py:24] Origin root
node name of graph segment: layer_9/attention/self/Softmax
2022-04-14 14:20:21,215 [DEBUG] [optimize_model.py:24] Loading the
TACO native passes.
...
```

您可以根据日志文件中的信息尝试解决问题，也可以通过 联系我们 将日志文件及 trace 文件发送给 TACO 技术团队寻求协助。

## 模型验证

经过上述步骤得到优化过的模型后，您可以使用 TensorFlow 的 python 接口加载该模型，验证其性能和正确性。加载模型运行的代码如下所示：

> **注意：**
> 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，需要使用 TensorFlow 的 `load_op_library` 接口加载位于 `${Taco Infer python安装目录}/lib` 目录中的两个包含自定义算子的动态链接库。

```
import tensorflow as tf
import numpy as np


def gen_test_data(batch_size = 1):
    MAX_SEQ_LEN = 32
    HIDDEN_DIM = 768
    rng = np.random.RandomState(2022)
    input_data = rng.randn(batch_size, MAX_SEQ_LEN, HIDDEN_DIM)
    return input_data


def run_model(pb_model_path, input_name='', output_name='',
test_data=None):
    load_library()
    graph_def = read_pb_model(pb_model_path)

    tf.import_graph_def(graph_def, name="")
    with tf.Session() as sess:
        output_tensor = sess.graph.get_tensor_by_name(output_name +
':0')
        input_tensor = sess.graph.get_tensor_by_name(input_name + ':0')
        output = sess.run([output_tensor], {input_tensor: test_data})
    print(output)


def load_library():

tf.load_op_library("/root/taco_test/venv/taco_dev/lib64/python3.6/site-
```

```
packages/taco/lib/libtaco_tf.so")

tf.load_op_library("/root/taco_test/venv/taco_dev/lib64/python3.6/site-
packages/taco/lib/libtidy_ops.so")


def read_pb_model(pb_model_path):
    with tf.gfile.GFile(pb_model_path, "rb") as f:
        graph_def = tf.GraphDef()
        graph_def.ParseFromString(f.read())
        return graph_def


if __name__ == '__main__':
    pb_model_path = "./optimized_model/fast-transformer-encoder.pb"
    input_name = "Placeholder"
    output_name = "mul_1"
    test_data = gen_test_data()
    run_model(pb_model_path, input_name=input_name,
output_name=output_name, test_data=test_data)
```

**根据实际的输出模型目录和 Taco Infer python 安装目录调整相关参数后，运行以上代码，即可加载优化后的模型进行推理计算。输出日志如下，可查看模型正常运行并且输出了计算结果。**

```
[root@VM-3-46-centos fast_transformer_encoder]#python run_model.py

...

[array([[[ 0.10604528,  2.0389333 ,  0.7614179 , ...,  1.8888083 ,
          0.9291879 , -0.42675698],
        [-0.42543304,  1.1162308 , -0.03524539, ...,  0.54224205,
          2.1540937 ,  0.760207  ],
        [-0.38016492,  0.4529049 ,  0.647373  , ..., -0.4969776 ,
         -0.04796869,  0.6062175 ],
        ...,
        [ 0.        ,  0.        ,  0.        , ..., -0.        ,
          0.        , -0.        ],
        [ 0.        ,  0.        , -0.        , ...,  0.        ,
         -0.        ,  0.        ],
        [-0.        ,  0.        ,  0.        , ..., -0.        ,
          0.        ,  0.        ]]], dtype=float32)]
```

# TensorFlow 模型推理部署

最近更新时间：2024-09-20 21:56:01

## 操作场景

本文介绍如何使用 TACO Infer 部署模型。在部署前，请确保您已完成 TensorFlow 模型优化，并且验证模型的性能和正确性符合预期后，您即可通过本文将模型部署在实际生产环境中。

## 操作步骤

### 环境准备

- 服务器：参见 TACO Infer 安装，选购 CPU 机型。
- ABI 版本：TACO Infer 支持 CXX11 ABI。如有其他版本需求请通过 联系我们 获取支持。
- SDK 包安装：在开发部署模型之前，请确保您已经安装了 TACO Infer SDK 安装包，详情请参见 获取 Wheel 包及 SDK 包 。解压后可查看安装包中包含三个动态链接库和一个可执行文件：

```
[root@VM-3-46-centos inceptionv3]# ll lib/
total 416180
-rwxr-xr-x 1 root root   1018440 Mar 31 20:25 libomp-1fdec59b.so
-rwxr-xr-x 1 root root  42617800 Mar 31 20:25 libtaco_tf.so
-rwxr-xr-x 1 root root 125954112 Mar 31 20:25 libtidy_ops.so
-rwxr-xr-x 1 root root 256572616 Mar 31 20:25 tidy_vm
```

> **说明：**
> - 建议您将所有 TACO 库文件拷贝到系统库目录 `/usr/lib` 下，以便链接器 ld 进行链接时可定位。或您也可以将 TACO 库文件拷贝到其他路径，并在 `LD_LIBRARY_PATH` 环境变量中添加库所在路径。
> - 请确保所有的 TACO 库文件位于同一路径下。

### 推理代码开发

以下代码以 TensorFlow C++ API 展示优化后模型的加载运行过程，您只需要按照标准的 TensorFlow C++ API 加载经过优化的模型即可，和加载普通的 TF 模型没有任何区别。

```
#include <string>
#include <vector>

#include "tensorflow/core/framework/graph.pb.h"
```

```
#include "tensorflow/core/framework/tensor.h"
#include "tensorflow/core/graph/default_device.h"
#include "tensorflow/core/graph/graph_def_builder.h"
#include "tensorflow/core/lib/core/threadpool.h"
#include "tensorflow/core/lib/strings/stringprintf.h"
#include "tensorflow/core/platform/init_main.h"
#include "tensorflow/core/platform/logging.h"
#include "tensorflow/core/platform/types.h"
#include "tensorflow/core/public/session.h"
#include "tensorflow/core/public/session_options.h"
#include "tensorflow/cc/client/client_session.h"
#include "tensorflow/core/protobuf/meta_graph.pb.h"
#include "tensorflow/c/c_api.h"

using tensorflow::GraphDef;
using tensorflow::int32;
using tensorflow::string;

void LoadFrozenPbModel(
    const std::unique_ptr<tensorflow::Session>& session,
    std::string model_path,
    bool as_text) {
  GraphDef graph_def;
  tensorflow::Status load_graph_status;
  std::cout << "Model Path: " << model_path << std::endl;
  if (as_text) {
    load_graph_status =
        ReadTextProto(tensorflow::Env::Default(), model_path,
&graph_def);
  } else {
    load_graph_status =
        ReadBinaryProto(tensorflow::Env::Default(), model_path,
&graph_def);
  }

  if (!load_graph_status.ok()) {
      std::cout << "Failed to load model: " << model_path
                    << load_graph_status.ToString() << std::endl;
  }

  auto session_status = session->Create(graph_def);
  if (!session_status.ok()) {
```

```
      std::cout << "Failed to create session" <<
session_status.ToString() << std::endl;
  }
}

void RunInference(
    const std::string& model_path,
    const std::vector<std::pair<std::string, tensorflow::Tensor>>&
inputs,
    const std::vector<string>& output_tensor_names,
    const std::vector<string>& target_node_names,
    std::vector<tensorflow::Tensor>& outputs) {  // NOLINT
  // Create session
  tensorflow::SessionOptions options;
  std::unique_ptr<tensorflow::Session>
session(tensorflow::NewSession(options));
  // Load model
  LoadFrozenPbModel(session, model_path, false);

  TF_CHECK_OK(
      session->Run(inputs, output_tensor_names, target_node_names,
&outputs));
}

int main() {
  std::string model_path = "./optimized_model/fast-transformer-
encoder.pb";

  // Create test input data
  std::vector<std::pair<std::string, tensorflow::Tensor>> inputs;
  auto input_tensor = tensorflow::Tensor(
      tensorflow::DT_FLOAT, tensorflow::TensorShape({1, 32, 768}));
  auto flat = input_tensor.flat<float>();
  for (int i = 0; i < 24576; i++) {
    flat(i) = 0.5;
  }
  string input_name = "Placeholder";
  inputs.emplace_back(input_name, input_tensor);

  // Create output tensor
  const std::vector<string> output_tensor_names = {"mul_1:0"};
  const std::vector<string> target_node_names = {};
  std::vector<tensorflow::Tensor> outputs;
```

```
  RunInference(
      model_path, inputs, output_tensor_names, target_node_names,
outputs);

  std::cout << "Output tensor: [" << std::endl;
  for (int i = 0; i < 10; i++) {
      std::cout << outputs[0].flat<float>()(i) << std::endl;
  }
  std::cout << "]" << std::endl;
}
```

## 编译链接

编译以上代码时，需要链接 Taco 提供的 `libtaco_tf.so` 和 `libtidy_ops.so` 两个动态库，及链接 TensorFlow 提供的 `libtensorflow_framework.so` 和 `libtensorflow_cc.so` 两个动态链接库。其中，`libtensorflow_framework.so` 位于 TensorFlow 的 Python 安装目录中。而 `libtensorflow_cc.so` 没有随着TensorFlow Python 安装包一起发行，需要您下载 TensorFlow 源码自行编译。

1. 执行以下命令，完成编译。

```
git clone https://github.com/tensorflow/tensorflow
cd tensorflow
git checkout ${xxx version}
./configure
bazel build --config=opt //tensorflow:libtensorflow_cc.so
```

2. `libtensorflow_cc.so` 编译完成后，即可进行模型部署代码的编译。编译参数如下所示：

```
#!/bin/bash

gcc -std=c++11 \
 -I/root/taco_test/1.15.0/include \
 -L/root/taco_test/venv/taco_dev/lib64/python3.6/site-
packages/tensorflow_core -ltensorflow_framework \
 -L/root/taco_test/lib -ltensorflow_cc -ltaco_tf -ltidy_ops \
 -lstdc++ \
 -o tf_sdk_demo tf_sdk_demo.cc
```

编译完成后，得到一个可运行的二进制文件。如下所示：

```
[root@VM-3-56-ubuntu (Taco Dev)
/home/ubuntu/taco_test/test_taco_sdk_demo]
#ll
-rw-rw-r--  1  500  500 3.4K Mar 29 16:04 taco_sdk_demo.cc
-rwxr-xr-x  1 root root  99K Mar 29 16:04 taco_sdk_demo*
drwxr-xr-x  2 root root 4.0K Mar 29 16:06 ./
drwxrwxr-x 10  500  500 4.0K Mar 29 16:58 ../
```

3. 确认相关的动态链接库已经正确链接：

```
(taco_dev) [root@VM-3-46-centos fast_transformer_encoder]# ldd
tf_sdk_demo
 linux-vdso.so.1 =>  (0x00007fff34bea000)
 libtensorflow_framework.so.1 =>
/root/taco_test/venv/taco_dev/lib64/python3.6/site-
packages/tensorflow_core/libtensorflow_framework.so.1
(0x00007efff59ab000)
 libtensorflow_cc.so.1 => /usr/local/lib/libtensorflow_cc.so.1
(0x00007effee66f000)
 libtaco_tf.so => /usr/local/lib/libtaco_tf.so (0x00007effec488000)
 libtidy_ops.so => /usr/local/lib/libtidy_ops.so (0x00007effea335000)
 libstdc++.so.6 => /lib64/libstdc++.so.6 (0x00007effea02d000)
 libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007effe9e17000)
 libc.so.6 => /lib64/libc.so.6 (0x00007effe9a49000)
 librt.so.1 => /lib64/librt.so.1 (0x00007effe9841000)
 libpthread.so.0 => /lib64/libpthread.so.0 (0x00007effe9625000)
 libdl.so.2 => /lib64/libdl.so.2 (0x00007effe9421000)
 libm.so.6 => /lib64/libm.so.6 (0x00007effe911f000)
 /lib64/ld-linux-x86-64.so.2 (0x00007efff76b7000)
 libomp-1fdec59b.so => /usr/local/lib/libomp-1fdec59b.so
(0x00007effe8e4e000)
 libcurl.so.4 => /lib64/libcurl.so.4 (0x00007effe8be4000)
 libuuid.so.1 => /lib64/libuuid.so.1 (0x00007effe89df000)
 libcrypto.so.10 => /lib64/libcrypto.so.10 (0x00007effe857c000)
 libidn.so.11 => /lib64/libidn.so.11 (0x00007effe8349000)
 libssh2.so.1 => /lib64/libssh2.so.1 (0x00007effe811c000)
 libssl3.so => /lib64/libssl3.so (0x00007effe7eb9000)
 libsmime3.so => /lib64/libsmime3.so (0x00007effe7c91000)
 libnss3.so => /lib64/libnss3.so (0x00007effe7958000)
 libnssutil3.so => /lib64/libnssutil3.so (0x00007effe7728000)
 libplds4.so => /lib64/libplds4.so (0x00007effe7524000)
 libplc4.so => /lib64/libplc4.so (0x00007effe731f000)
```

```
  libnspr4.so => /lib64/libnspr4.so (0x00007effe70e1000)
  libgssapi_krb5.so.2 => /lib64/libgssapi_krb5.so.2
 (0x00007effe6e94000)
  libkrb5.so.3 => /lib64/libkrb5.so.3 (0x00007effe6bab000)
  libk5crypto.so.3 => /lib64/libk5crypto.so.3 (0x00007effe6978000)
  libcom_err.so.2 => /lib64/libcom_err.so.2 (0x00007effe6774000)
  liblber-2.4.so.2 => /lib64/liblber-2.4.so.2 (0x00007effe6565000)
  libldap-2.4.so.2 => /lib64/libldap-2.4.so.2 (0x00007effe6310000)
  libz.so.1 => /lib64/libz.so.1 (0x00007effe60fa000)
  libssl.so.10 => /lib64/libssl.so.10 (0x00007effe5e88000)
  libkrb5support.so.0 => /lib64/libkrb5support.so.0
 (0x00007effe5c78000)
  libkeyutils.so.1 => /lib64/libkeyutils.so.1 (0x00007effe5a74000)
  libresolv.so.2 => /lib64/libresolv.so.2 (0x00007effe585a000)
  libsasl2.so.3 => /lib64/libsasl2.so.3 (0x00007effe563d000)
  libselinux.so.1 => /lib64/libselinux.so.1 (0x00007effe5416000)
  libcrypt.so.1 => /lib64/libcrypt.so.1 (0x00007effe51df000)
  libpcre.so.1 => /lib64/libpcre.so.1 (0x00007effe4f7d000)
  libfreebl3.so => /lib64/libfreebl3.so (0x00007effe4d7a000)
```

## 推理计算

**部署程序编译完成后，运行即可加载优化后的模型并进行推理计算。可查看模型正常加载运行，并输出了推理计算结果。如下所示：**

```
[root@VM-3-56-ubuntu (Taco Dev)
/home/ubuntu/taco_test/test_taco_sdk_demo]
#./taco_sdk_demo

2022-03-31 20:52:21.702558: I
tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports
instructions that this TensorFlow binary was not compiled to use: AVX2
FMA
2022-03-31 20:52:21.703465: I
tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel driver
does not appear to be running on this host (VM-3-46-centos):
/proc/driver/nvidia/version does not exist
Model Path: ./optimized_model/fast-transformer-encoder.pb
...
Output tensor: [
0.692124
-1.30981
1.93331
```

```
-0.0825812
-0.423409
-0.73291
-2.13366
0.758448
-1.25149
0.659645
]
```