TensorRT基础

一、准备知识

NVIDIA® TensorRT™是一个用于高性能深度学习的推理框架。它可以与TensorFlow、PyTorch和MXNet等训练框架相辅相成地工作。

1.1 环境配置

A. CUDA Driver

- 使用CUDA前，要求GPU驱动与`cuda`的版本要匹配，匹配关系如下：

> 参考：https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html#cuda-major-component-versions__table-cuda-toolkit-driver-versions

Table 3. CUDA Toolkit and Corresponding Driver Versions

CUDA Toolkit	Toolkit Driver Version	
	Linux x86_64 Driver Version	Windows x86_64 Driver Version
CUDA 12.0 Update 1	>=525.85.12	>=528.33
CUDA 12.0 GA	>=525.60.13	>=527.41
CUDA 11.8 GA	>=520.61.05	>=520.06
CUDA 11.7 Update 1	>=515.48.07	>=516.31
CUDA 11.7 GA	>=515.43.04	>=516.01
CUDA 11.6 Update 2	>=510.47.03	>=511.65
CUDA 11.6 Update 1	>=510.47.03	>=511.65
CUDA 11.6 GA	>=510.39.01	>=511.23
CUDA 11.5 Update 2	>=495.29.05	>=496.13
CUDA 11.5 Update 1	>=495.29.05	>=496.13
CUDA 11.5 GA	>=495.29.05	>=496.04
CUDA 11.4 Update 4	>=470.82.01	>=472.50
CUDA 11.4 Update 3	>=470.82.01	>=472.50
CUDA 11.4 Update 2	>=470.57.02	>=471.41
CUDA 11.4 Update 1	>=470.57.02	>=471.41
CUDA 11.4.0 GA	>=470.42.01	>=471.11
CUDA 11.3.1 Update 1	>=465.19.01	>=465.89
CUDA 11.3.0 GA	>=465.19.01	>=465.89
CUDA 11.2.2 Update 2	>=460.32.03	>=461.33

- 检查机器建议的驱动

```
$ ubuntu-drivers devices

// 比如我的机器输出如下

(base) enpei@enpei-ubutnu-desktop:~$ ubuntu-drivers devices
== /sys/devices/pci0000:00/0000:00:01.0/0000:01:00.0 ==
modalias : pci:v000010DEd00001C03sv000010DEsd000011D7bc03sc00i00
vendor   : NVIDIA Corporation
model    : GP106 [GeForce GTX 1060 6GB]
driver   : nvidia-driver-525 - distro non-free recommended
driver   : nvidia-driver-510 - distro non-free
driver   : nvidia-driver-390 - distro non-free
driver   : nvidia-driver-520 - third-party non-free
driver   : nvidia-driver-515-server - distro non-free
driver   : nvidia-driver-470 - distro non-free
driver   : nvidia-driver-418-server - distro non-free
driver   : nvidia-driver-470-server - distro non-free
driver   : nvidia-driver-525-server - distro non-free
driver   : nvidia-driver-515 - distro non-free
driver   : nvidia-driver-450-server - distro non-free
driver   : xserver-xorg-video-nouveau - distro free builtin
```

上面信息提示了，当前我使用的GPU是[GeForce GTX 1060 6GB]，他推荐的（recommended）驱动是`nvidia-driver-525`。

- 安装指定版本

```
$ sudo apt install nvidia-driver-525
```

- 重启

```
$ sudo reboot
```

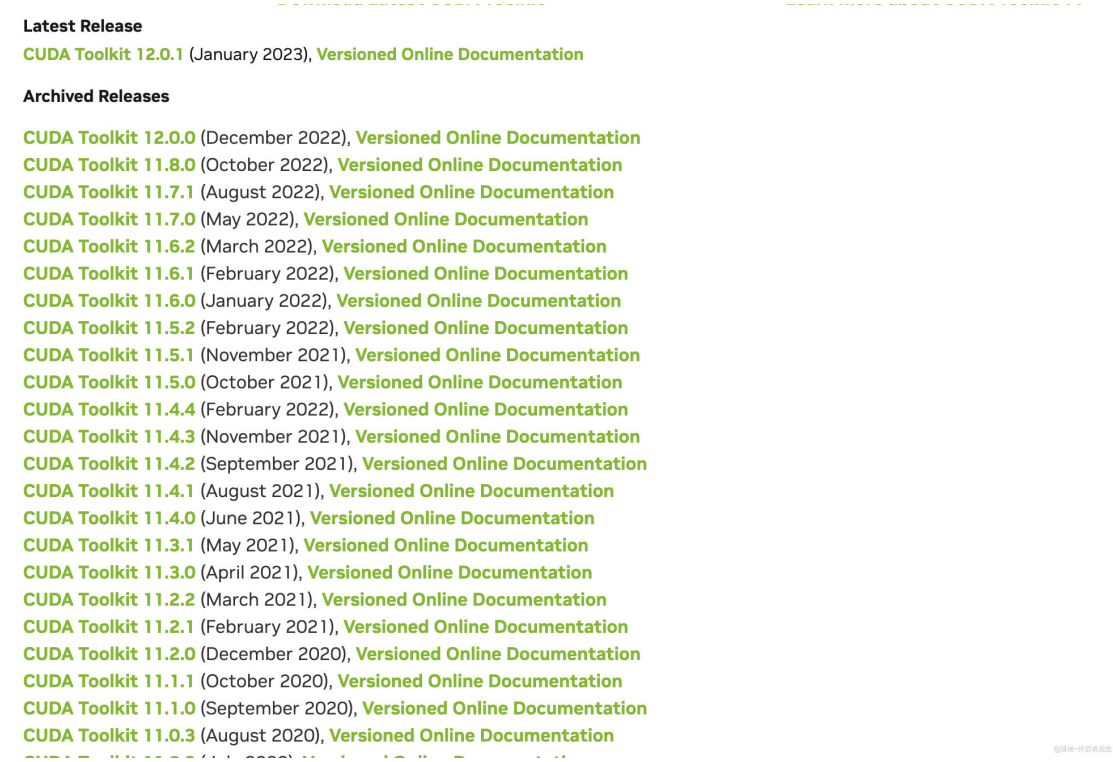
- 检查安装

```
$ nvidia-smi

(base) enpei@enpei-ubutnu-desktop:~$ nvidia-smi
Mon Feb  2 12:23:45 2023
+-----------------------------------------------------------------------------
--+
| NVIDIA-SMI 525.78.01    Driver Version: 525.78.01    CUDA Version: 12.0
  |
|-------------------------------+----------------------+----------------------
--+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr.
ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute
M. |
|                               |                      |               MIG
M. |
|===============================+======================+======================
==|
|   0  NVIDIA GeForce ...  Off  | 00000000:01:00.0  On |
N/A |
| 40%   29C    P8     9W / 120W |    239MiB /  6144MiB |      0%
Default |
|                               |                      |
N/A |
+-------------------------------+----------------------+----------------------
--+

+-----------------------------------------------------------------------------
--+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name                  GPU
Memory |
|        ID   ID                                                   Usage
|
|=============================================================================
==|
|    0   N/A  N/A      1079      G   /usr/lib/xorg/Xorg
102MiB |
|    0   N/A  N/A      1387      G   /usr/bin/gnome-shell
133MiB |
+-----------------------------------------------------------------------------
--+
```

可以看到当前安装的驱动版本是`525.78.01`，需要注意`CUDA Version: 12.0`指当前驱动支持的最高版本。

B. CUDA

- 选择对应版本：https://developer.nvidia.com/cuda-toolkit-archive



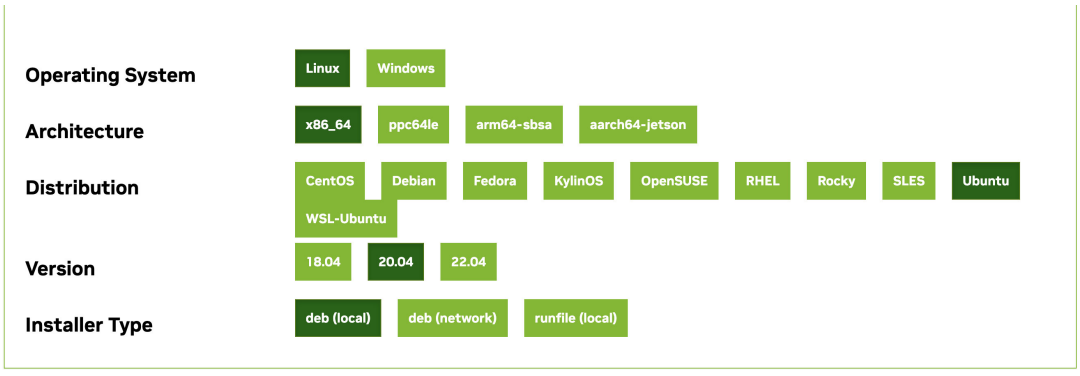
Latest Release

CUDA Toolkit 12.0.1 (January 2023), Versioned Online Documentation

Archived Releases

CUDA Toolkit 12.0.0 (December 2022), Versioned Online Documentation
CUDA Toolkit 11.8.0 (October 2022), Versioned Online Documentation
CUDA Toolkit 11.7.1 (August 2022), Versioned Online Documentation
CUDA Toolkit 11.7.0 (May 2022), Versioned Online Documentation
CUDA Toolkit 11.6.2 (March 2022), Versioned Online Documentation
CUDA Toolkit 11.6.1 (February 2022), Versioned Online Documentation
CUDA Toolkit 11.6.0 (January 2022), Versioned Online Documentation
CUDA Toolkit 11.5.2 (February 2022), Versioned Online Documentation
CUDA Toolkit 11.5.1 (November 2021), Versioned Online Documentation
CUDA Toolkit 11.5.0 (October 2021), Versioned Online Documentation
CUDA Toolkit 11.4.4 (February 2022), Versioned Online Documentation
CUDA Toolkit 11.4.3 (November 2021), Versioned Online Documentation
CUDA Toolkit 11.4.2 (September 2021), Versioned Online Documentation
CUDA Toolkit 11.4.1 (August 2021), Versioned Online Documentation
CUDA Toolkit 11.4.0 (June 2021), Versioned Online Documentation
CUDA Toolkit 11.3.1 (May 2021), Versioned Online Documentation
CUDA Toolkit 11.3.0 (April 2021), Versioned Online Documentation
CUDA Toolkit 11.2.2 (March 2021), Versioned Online Documentation
CUDA Toolkit 11.2.1 (February 2021), Versioned Online Documentation
CUDA Toolkit 11.2.0 (December 2020), Versioned Online Documentation
CUDA Toolkit 11.1.1 (October 2020), Versioned Online Documentation
CUDA Toolkit 11.1.0 (September 2020), Versioned Online Documentation
CUDA Toolkit 11.0.3 (August 2020), Versioned Online Documentation

- 根据提示安装，如我选择的11.8 版本的：https://developer.nvidia.com/cuda-11-8-0-download-archive?target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=20.04&target_type=deb_local



Download Installer for Linux Ubuntu 20.04 x86_64

The base installer is available for download below.

Base Installer

Installation Instructions:

```
$ wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2004/x86_64/cuda-ubuntu2004.pin
$ sudo mv cuda-ubuntu2004.pin /etc/apt/preferences.d/cuda-repository-pin-600
$ wget https://developer.download.nvidia.com/compute/cuda/11.8.0/local_installers/cuda-repo-ubuntu2004-11-8-local_11.8.0-520.61.05-1_
amd64.deb
$ sudo dpkg -i cuda-repo-ubuntu2004-11-8-local_11.8.0-520.61.05-1_amd64.deb
$ sudo cp /var/cuda-repo-ubuntu2004-11-8-local/cuda-*-keyring.gpg /usr/share/keyrings/
$ sudo apt-get update
$ sudo apt-get -y install cuda
```

```
wget
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2004/x86_64/cuda-ubuntu2004.pin
sudo mv cuda-ubuntu2004.pin /etc/apt/preferences.d/cuda-repository-pin-600
wget
https://developer.download.nvidia.com/compute/cuda/11.8.0/local_installers/cuda-repo-ubuntu2004-11-8-local_11.8.0-520.61.05-1_amd64.deb
sudo dpkg -i cuda-repo-ubuntu2004-11-8-local_11.8.0-520.61.05-1_amd64.deb
sudo cp /var/cuda-repo-ubuntu2004-11-8-local/cuda-*-keyring.gpg
/usr/share/keyrings/
sudo apt-get update
sudo apt-get -y install cuda
```

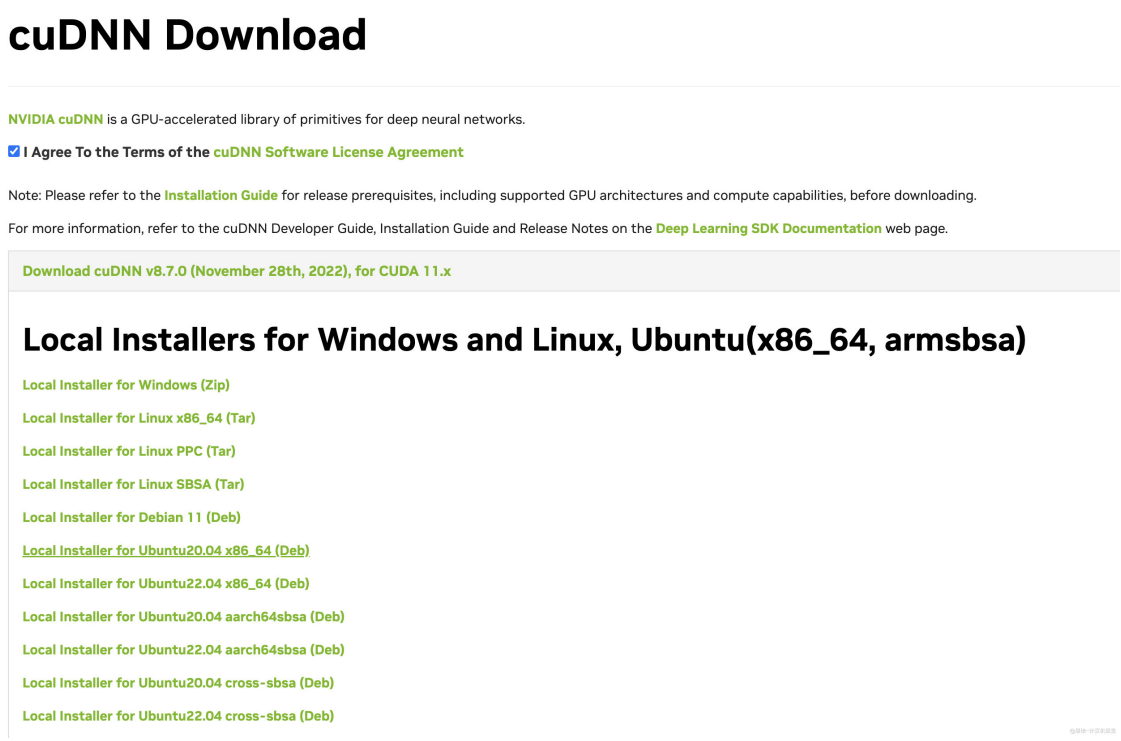
- 安装 `nvcc`

```
sudo apt install nvidia-cuda-toolkit
```

- 重启

C. cuDNN

- 下载安装包：访问：https://developer.nvidia.com/zh-cn/cudnn，选择对应的版本，下载对应的安装包（建议使用Debian包安装）



比如我下载的是：Local Installer for Ubuntu20.04 x86_64 (Deb)，下载后的文件名为 `cudnn-local-repo-ubuntu2004-8.7.0.84_1.0-1_amd64.deb`。

- 安装：

> 参考链接：https://docs.nvidia.com/deeplearning/cudnn/install-guide/index.html

```
# 注意，运行下面的命令前，将下面的 X.Y和v8.x.x.x 替换成自己具体的CUDA 和 cuDNN版本，如我的CUDA 版本是11.8，cuDNN 版本是 8.7.0.84
```

```
sudo dpkg -i cudnn-local-repo-${OS}-8.x.x.x_1.0-1_amd64.deb
# 我的：sudo dpkg -i cudnn-local-repo-ubuntu2004-8.7.0.84_1.0-1_amd64.deb

sudo cp /var/cudnn-local-repo-*/cudnn-local-*-keyring.gpg
/usr/share/keyrings/
sudo apt-get update


sudo apt-get install libcudnn8=8.x.x.x-1+cudaX.Y
# 我的：sudo apt-get install libcudnn8=8.7.0.84-1+cuda11.8


sudo apt-get install libcudnn8-dev=8.x.x.x-1+cudaX.Y
# 我的：sudo apt-get install libcudnn8-dev=8.7.0.84-1+cuda11.8


sudo apt-get install libcudnn8-samples=8.x.x.x-1+cudaX.Y
# 我的：sudo apt-get install libcudnn8-samples=8.7.0.84-1+cuda11.8
```

- 验证

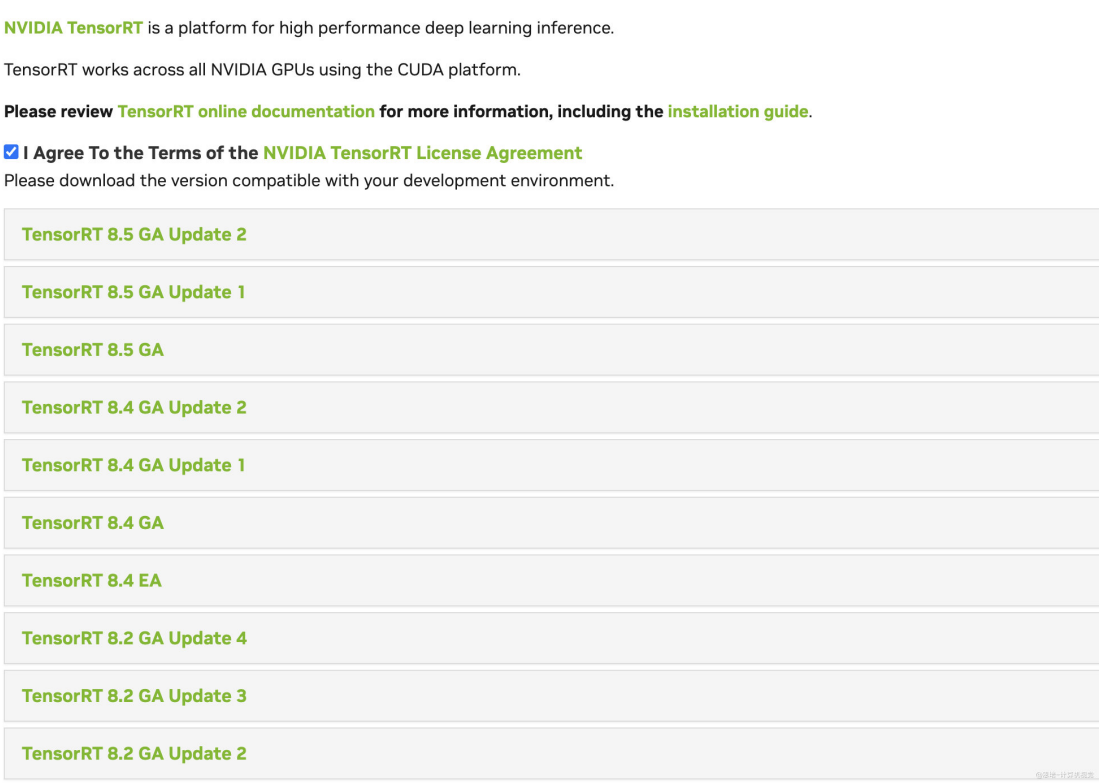
```
# 复制文件
cp -r /usr/src/cudnn_samples_v8/ $HOME
cd  $HOME/cudnn_samples_v8/mnistCUDNN
make clean && make
./mnistCUDNN
```

> 可能报错：test.c:1:10: fatal error: FreeImage.h: No such file or directory
>
> 解决办法：sudo apt-get install libfreeimage3 libfreeimage-dev

D. TensorRT

- 访问：https://developer.nvidia.com/nvidia-tensorrt-8x-download 下载对应版本的TensorRT

NVIDIA TensorRT 8.x Download



比如我选择的是 8.5.3版本，下载完文件名为：`nv-tensorrt-local-repo-ubuntu2004-8.5.3-cuda-11.8_1.0-1_amd64.deb`

- 安装：

> 参考地址：https://docs.nvidia.com/deeplearning/tensorrt/install-guide/index.html#installing-debian

```
# 替换成自己的OS 和 版本信息
os="ubuntuxx04"
tag="8.x.x-cuda-x.x"
sudo dpkg -i nv-tensorrt-local-repo-${os}-${tag}_1.0-1_amd64.deb
# 我的：sudo dpkg -i nv-tensorrt-local-repo-ubuntu2004-8.5.3-cuda-11.8_1.0-1_amd64.deb
sudo cp /var/nv-tensorrt-local-repo-${os}-${tag}/*-keyring.gpg /usr/share/keyrings/
# 我的：sudo cp /var/nv-tensorrt-local-repo-ubuntu2004-8.5.3-cuda-11.8/*-keyring.gpg /usr/share/keyrings/

sudo apt-get update
sudo apt-get install tensorrt
```

- 验证：

```
dpkg -l | grep TensorRT

# 输出
ii  libnvinfer-bin                                  8.5.3-1+cuda11.8
            amd64        TensorRT binaries
ii  libnvinfer-dev                                  8.5.3-1+cuda11.8
            amd64        TensorRT development libraries and headers
```

```
ii  libnvinfer-plugin-dev                              8.5.3-1+cuda11.8
           amd64        TensorRT plugin libraries
ii  libnvinfer-plugin8                                 8.5.3-1+cuda11.8
           amd64        TensorRT plugin libraries
ii  libnvinfer-samples                                 8.5.3-1+cuda11.8
           all          TensorRT samples
ii  libnvinfer8                                        8.5.3-1+cuda11.8
           amd64        TensorRT runtime libraries
ii  libnvonnxparsers-dev                               8.5.3-1+cuda11.8
           amd64        TensorRT ONNX libraries
ii  libnvonnxparsers8                                  8.5.3-1+cuda11.8
           amd64        TensorRT ONNX libraries
ii  libnvparsers-dev                                   8.5.3-1+cuda11.8
           amd64        TensorRT parsers libraries
ii  libnvparsers8                                      8.5.3-1+cuda11.8
           amd64        TensorRT parsers libraries
ii  tensorrt                                           8.5.3.1-1+cuda11.8
           amd64        Meta package for TensorRT
```

> 如果遇到`unmet dependencies`的问题, 一般是cuda cudnn没有安装好。TensorRT的`INCLUDE`路径是`/usr/include/x86_64-linux-gnu/`, `LIB`路径是`/usr/lib/x86_64-linux-gnu/`,Sample code在`/usr/src/tensorrt/samples`, `trtexec`在`/usr/src/tensorrt/bin`下。

1.2 TensorRT介绍

首先要明确TensorRT的定位，是一个推理框架：

TensorRT（Tensor Runtime）是由NVIDIA开发的深度学习推理库，旨在优化和加速深度学习模型的推理过程。TensorRT主要用于在NVIDIA GPU上执行深度学习推理任务，通过利用GPU的并行计算能力，加速神经网络推断的速度，从而提高模型的实时性能。TensorRT可以通过层融合、混合精度、量化等技术显著提高深度神经网络的推理性能。

TensorRT是英伟达官方针对自己的硬件设备面向AI工作者推出的一种部署方案。

在训练了神经网络之后，TensorRT可以对网络进行压缩、优化以及运行时部署，并且没有框架的开销。TensorRT通过combines layers，kernel优化选择，以及根据指定的精度执行归一化和转换成最优的matrix math方法，改善网络的延迟、吞吐量以及效率。

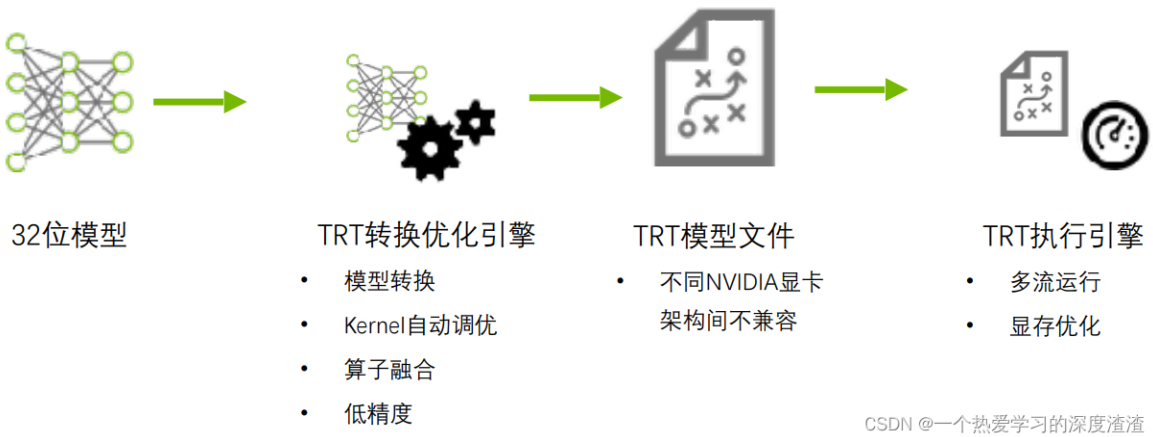
TensorRT通过结合抽象出特定硬件细节的高级API和优化推理的实现来解决这些问题，以实现高吞吐量、低延迟和低设备内存占用。

- 以下是TensorRT的一些主要特点和功能：

  1. **精简优化：** TensorRT通过采用精简和优化技术，以减少模型推理过程中的计算和内存开销，从而提高性能。
  2. **混合精度支持：** TensorRT支持混合精度推理，利用低精度（如半精度浮点数）来执行计算，从而在减少存储和带宽需求的同时提高推理速度。
  3. **动态Tensor支持：** TensorRT 7引入了动态Tensor支持，使得可以在推理过程中处理变化的形状和大小，提高了模型的灵活性。
  4. **支持多种深度学习框架：** TensorRT兼容主流的深度学习框架，包括TensorFlow、PyTorch和ONNX等，可以通过将模型转换为TensorRT格式来进行推理加速。
  5. **优化器和量化：** TensorRT提供了优化器，可以在模型转换过程中执行各种优化操作，还支持量化技术，通过减少模型参数的比特数来降低计算和存储开销。
  6. **支持动态批处理：** TensorRT支持动态批处理，允许在推理过程中使用不同大小的输入批次，提高了模型的适应性。
  7. **集成TensorFlow和TensorFlow Lite：** TensorRT可以与TensorFlow和TensorFlow Lite集成，使得在NVIDIA GPU上执行TensorFlow模型变得更为高效。
  8. **插件支持：** TensorRT提供了插件支持，可以通过插件自定义层的实现，以适应一些特殊的网络结构。

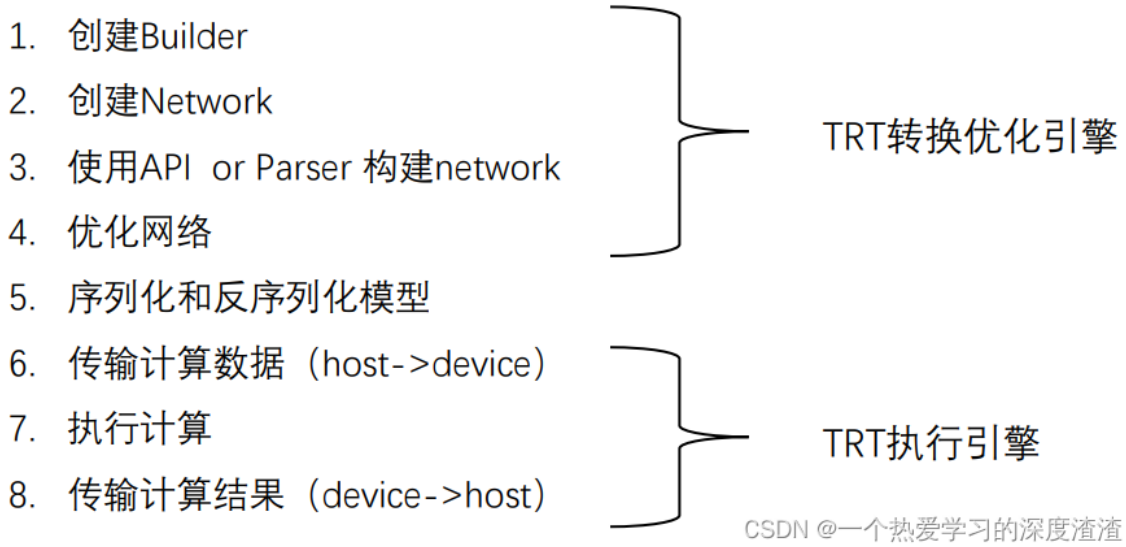
  TensorRT可以在边缘设备、嵌入式系统和数据中心等场景中使用，以加速深度学习模型的推理任务。在需要实时性能和低延迟的应用中，TensorRT是一个强大的工具，可用于优化和部署深度学习模型。

TensorRT分两个阶段运行



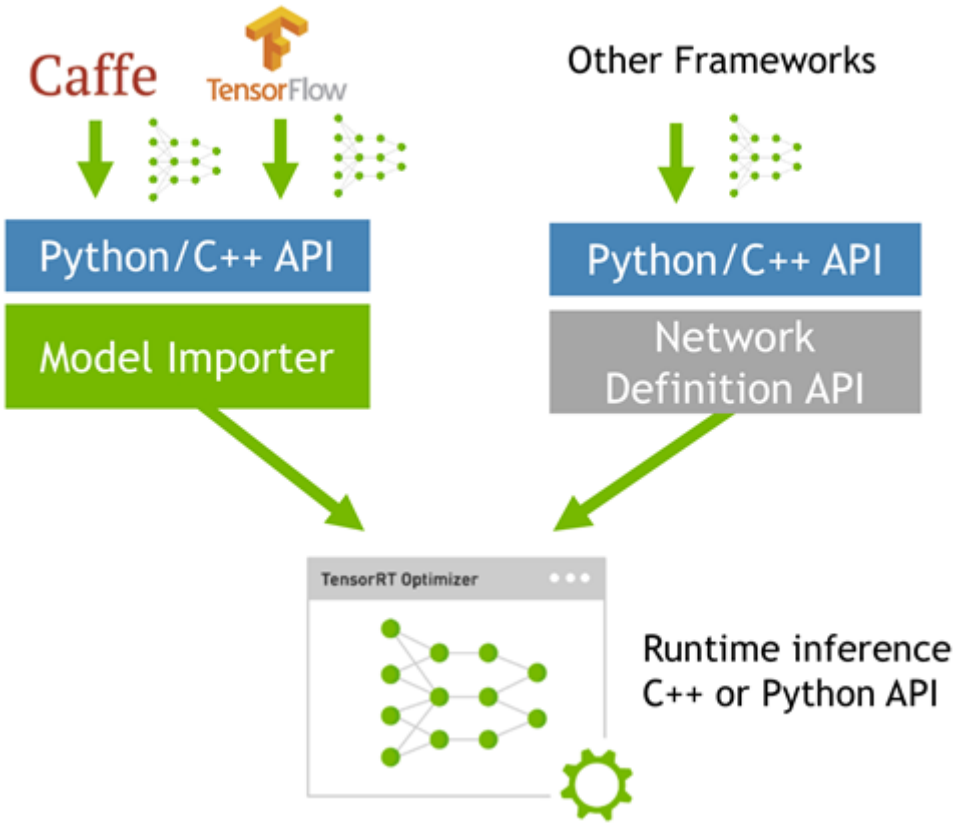
- 构建（`Build`）阶段：你向TensorRT提供一个模型定义，TensorRT为目标GPU优化这个模型。这个过程可以离线运行。

- 运行时（`Runtime`）阶段：你使用优化后的模型来运行推理。



构建阶段后，我们可以将优化后的模型保存为模型文件，模型文件可以用于后续加载，以省略模型构建和优化的过程。

其中构建网络分为两种方式。



一种是API构建，也就是网络的每一层都重新用代码构建，相对来说比较复杂；

一种是用Parser来构建，也就是特定的网络有其特定的框架有对应的加载接口，只需要简单几行代码就可以构建网络结构；

二、构建阶段

构建阶段的最高级别接口是 `Builder`。`Builder`负责优化一个模型，并产生`Engine`。通过如下接口创建一个`Builder`。

```
nvinfer1::IBuilder* builder = nvinfer1::createInferBuilder(logger);
```

要生成一个可以进行推理的`Engine`，一般需要以下三个步骤：

- 创建一个网络定义
- 填写`Builder`构建配置参数，告诉构建器应该如何优化模型
- 调用`Builder`生成`Engine`

2.1 创建网络定义

`NetworkDefinition`接口被用来定义模型。如下所示：

```
// bit shift，移位：y左移N位，相当于 y * 2^N
// kEXPLICIT_BATCH（显性Batch）为0，1U << 0 = 1
// static_cast：强制类型转换
const auto explicitBatch = 1U << static_cast<uint32_t>
(nvinfer1::NetworkDefinitionCreationFlag::kEXPLICIT_BATCH);
nvinfer1::INetworkDefinition* network = builder->createNetworkV2(explicitBatch);
```

接口`createNetworkV2`接受配置参数，参数用按位标记的方式传入。比如上面激活`explicitBatch`，是通过`1U << static_cast<uint32_t>(nvinfer1::NetworkDefinitionCreationFlag::kEXPLICIT_BATCH);`将explicitBatch对应的配置位设置为1实现的。在新版本中，请使用`createNetworkV2`而非其他任何创建`NetworkDefinition`的接口。

将模型转移到TensorRT的最常见的方式是以ONNX格式从框架中导出（将在后续课程进行介绍），并使用TensorRT的ONNX解析器来填充网络定义。同时，也可以使用TensorRT的`Layer`和`Tensor`等接口一步一步地进行定义。通过接口来定义网络的代码示例如下：

- 添加输入层

```
nvinfer1::ITensor* input = network->addInput("data", nvinfer1::DataType::kFLOAT,
nvinfer1::Dims4{1, input_size, 1, 1});
```

- 添加全连接层

```
nvinfer1::IFullyConnectedLayer* fc1 = network->addFullyConnected(*input,
output_size, fc1w, fc1b);
```

- 添加激活层

```
nvinfer1::IActivationLayer* relu1 = network->addActivation(*fc1->getOutput(0),
nvinfer1::ActivationType::kRELU);
```

通过调用`network`的方法，我们可以构建网络的定义。

无论你选择哪种方式，你还必须定义哪些张量是网络的输入和输出。没有被标记为输出的张量被认为是瞬时值，可以被构建者优化掉。输入和输出张量必须被命名，以便在运行时，TensorRT知道如何将输入和输出缓冲区绑定到模型上。示例代码如下：

```
// 设置输出名字
relu1->getOutput(0)->setName("output");
// 标记输出
network->markOutput(*relu1->getOutput(0));
```

TensorRT的网络定义不会复制参数数组（如卷积的权重）。因此，在构建阶段完成之前，你不能释放这些数组的内存。

2.2 配置参数

下面我们来添加相关`Builder`的配置。`createBuilderConfig`接口被用来指定TensorRT应该如何优化模型。如下：

```
nvinfer1::IBuilderConfig* config = builder->createBuilderConfig();
```

在可用的配置选项中，你可以控制TensorRT降低计算精度的能力，控制内存和运行时执行速度之间的权衡，并限制CUDA®内核的选择。由于构建器的运行可能需要几分钟或更长时间，你也可以控制构建器如何搜索内核，以及缓存搜索结果以用于后续运行。在我们的示例代码中，我们仅配置`workspace`（workspace 就是 tensorrt 里面算子可用的内存空间 ）大小和运行时`batch size`，如下：

```
// 配置运行时batch size参数
builder->setMaxBatchSize(1);
// 配置运行时workspace大小
std::cout << "Workspace Size = " << (1 << 28) / 1024.0f / 1024.0f << "MB" <<
std::endl; // 256Mib
config->setMaxWorkspaceSize(1 << 28);
```

2.3 生成Engine

在你有了网络定义和`Builder`配置后，你可以调用`Builder`来创建`Engine`。`Builder`以一种称为`plan`的序列化形式创建`Engine`，它可以立即反序列化，也可以保存到磁盘上供以后使用。需要注意的是，由TensorRT创建的`Engine`是特定于创建它们的TensorRT版本和创建它们的GPU的，当迁移到别的GPU和TensorRT版本时，不能保证模型能够被正确执行。生成`Engine`的示例代码如下：

```
nvinfer1::ICudaEngine* engine = builder->buildEngineWithConfig(*network,
*config);
```

2.4 保存为模型文件

当有了`engine`后我们可以将其保存为文件，以供后续使用。代码如下：

```
// 序列化
nvinfer1::IHostMemory* engine_data = engine->serialize();
// 保存至文件
std::ofstream engine_file("mlp.engine", std::ios::binary);
engine_file.write((char*)engine_data->data(), engine_data->size());
```

2.5 释放资源

```
// 理论上，前面申请的资源都应该在这里释放，但是这里只是为了演示，所以只释放了部分资源
file.close();               // 关闭文件
delete serialized_engine; // 释放序列化的engine
delete engine;              // 释放engine
delete config;              // 释放config
delete network;             // 释放network
delete builder;             // 释放builder
```

三、运行时阶段

TensorRT运行时的最高层级接口是`Runtime`如下：

```
nvinfer1::IRuntime *runtime = nvinfer1::createInferRuntime(looger);
```

当使用`Runtime`时，你通常会执行以下步骤：

- 反序列化一个计划以创建一个`Engine`。
- 从引擎中创建一个`ExecutionContext`。

然后，重复进行：

- 为Inference填充输入缓冲区。
- 在`ExecutionContext`调用`enqueueV2()`来运行Inference

3.1 反序列化并创建Engine

通过读取模型文件并反序列化，我们可以利用runtime生成`Engine`。如下：

```
nvinfer1::ICudaEngine *engine = runtime-
>deserializeCudaEngine(engine_data.data(), engine_data.size(), nullptr);
```

`Engine`接口代表一个优化的模型。你可以查询`Engine`关于网络的输入和输出张量的信息，如：预期尺寸、数据类型、数据格式等。

3.2 创建一个`ExecutionContext`

有了Engine后我们需要创建`ExecutionContext`以用于后面的推理执行。

```
nvinfer1::IExecutionContext *context = engine->createExecutionContext();
```

从`Engine`创建的`ExecutionContext`接口是调用推理的主要接口。`ExecutionContext`包含与特定调用相关的所有状态，因此你可以有多个与单个引擎相关的上下文，且并行运行它们，在这里我们暂不展开了解，仅做介绍。

3.3 为推理填充输入

我们首先创建CUDA Stream用于推理的执行。

> stream 可以理解为一个任务队列，调用以 async 结尾的 api 时，是把任务加到队列，但执行是异步的，当有多个任务且互相没有依赖时可以创建多个 stream 分别用于不同的任务，任务直接的执行可以被 cuda driver 调度，这样某个任务做 memcpy时 另外一个任务可以执行计算任务，这样可以提高 gpu利用率。

```
cudaStream_t stream = nullptr;
// 创建CUDA Stream用于context推理
cudaStreamCreate(&stream);
```

然后我们同时在CPU和GPU上分配输入输出内存，并将输入数据从CPU拷贝到GPU上。

```
// 输入数据
float* h_in_data = new float[3]{1.4, 3.2, 1.1};
int in_data_size = sizeof(float) * 3;
float* d_in_data = nullptr;
// 输出数据
float* h_out_data = new float[2]{0.0, 0.0};
int out_data_size = sizeof(float) * 2;
float* d_out_data = nullptr;
// 申请GPU上的内存
cudaMalloc(&d_in_data, in_data_size);
cudaMalloc(&d_out_data, out_data_size);
// 拷贝数据
cudaMemcpyAsync(d_in_data, h_in_data, in_data_size, cudaMemcpyHostToDevice,
stream);
// enqueueV2中是把输入输出的内存地址放到bindings这个数组中，需要写代码时确定这些输入输出的顺序（这样容易出错，而且不好定位bug，所以新的接口取消了这样的方式，不过目前很多官方 sample 也在用v2）
float* bindings[] = {d_in_data, d_out_data};
```

3.4 调用enqueueV2来执行推理

将数据从CPU中拷贝到GPU上后，便可以调用`enqueueV2`进行推理。代码如下：

```
// 执行推理
bool success = context->enqueueV2((void**)bindings, stream, nullptr);
// 把数据从GPU拷贝回host
cudaMemcpyAsync(h_out_data, d_out_data, out_data_size, cudaMemcpyDeviceToHost,
stream);
// stream同步，等待stream中的操作完成
cudaStreamSynchronize(stream);
// 输出
std::cout << "输出信息: " << host_output_data[0] << " " << host_output_data[1] <<
std::endl;
```

3.5 释放资源

```
cudaStreamDestroy(stream);
cudaFree(device_input_data_address);
cudaFree(device_output_data_address);
delete[] host_input_data;
delete[] host_output_data;

delete context;
delete engine;
delete runtime;
```

四、编译和运行

利用我们前面cmake课程介绍的添加自定义模块的方法，创建`cmake/FindTensorRT.cmake`文件，我们运行下面的命令以编译示例代码：

```
cmake -S . -B build
cmake --build build
```

然后执行下面命令，build将生成mlp.engine，而runtime将读取mlp.engine并执行：

```
./build/build
./build/runtime
```

最后将看到输出结果：

```
输出信息: 0.970688 0.999697
```