CHAPTER3
TENSORRT基础入门

主　讲：韩君

单　位：早稻田大学

公众号：自动驾驶之心

主要内容

1. TENSORRT简介
2. TENSORRT的应用场景
3. TENSORRT的模块
4. 导出ONNX以及修改ONNX的方法
5. 初步使用TENSORRT

TensorRT简介

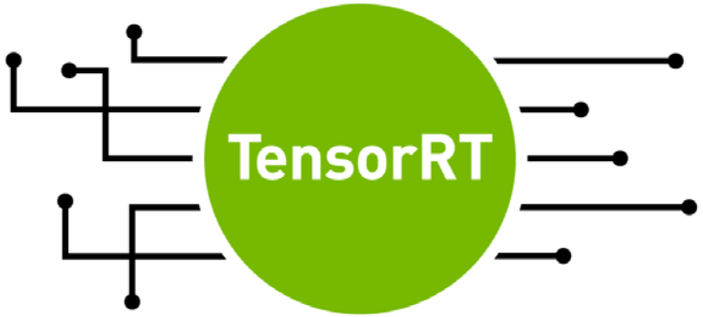
Goal: 理解TensorRT的工作流程，TensorRT的极限，以及其他DNN编译器

什么是TensorRT

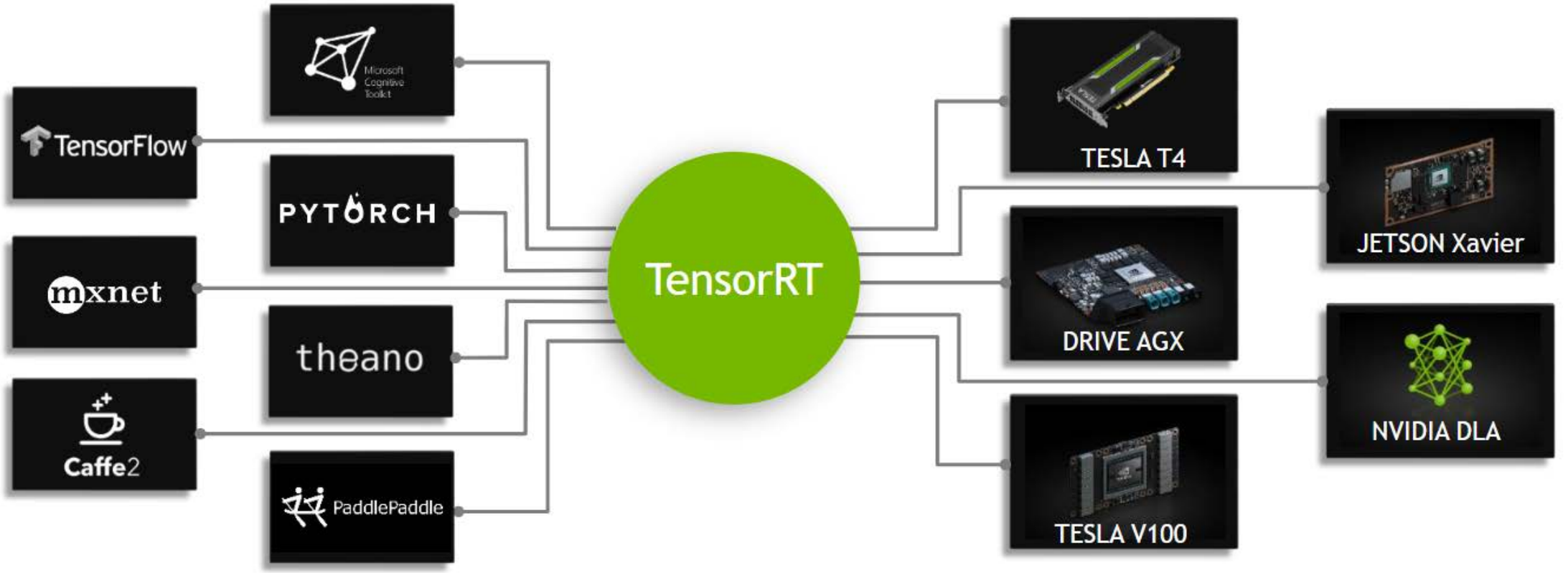
“NVIDIA® TensorRT™, an SDK for high-performance deep learning inference, includes a deep learning inference optimizer and runtime that delivers low latency and high throughput for inference applications.”

可以把它理解为一种针对NVIDIA GPU的一种优化编译器，以及其他开发工具

- 自动优化模型
  - 寻找模型中可以并行处理的地方
  - 针对当前部署的GPU框架，寻找最优的调度和并行策略
- 支持多框架输入
  - ONNX
- Python/C++ API接口
  - 可以方便在自己的程序中调用TensorRT API来实现推理



什么是TensorRT

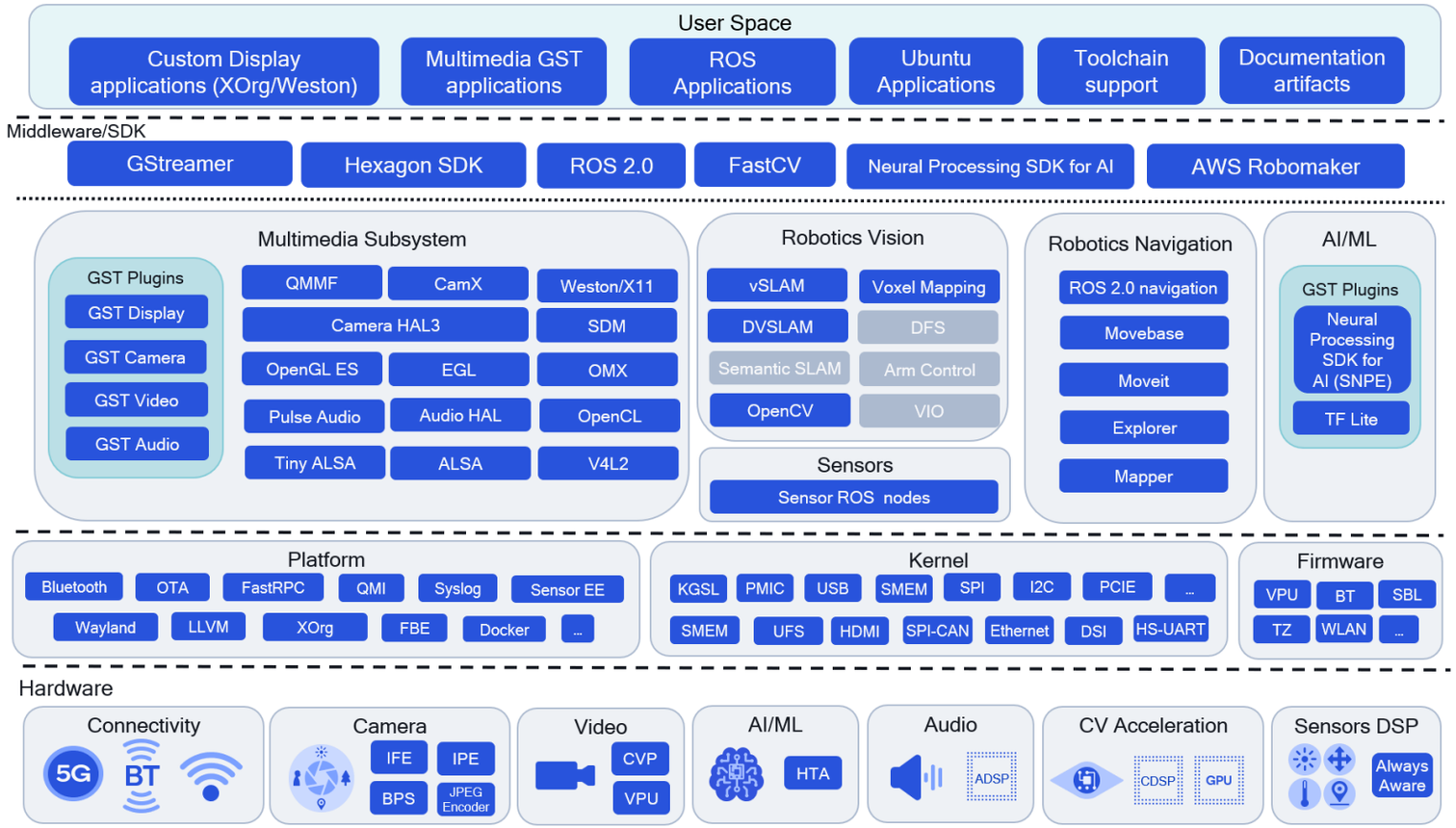


现在的大多数DL framework都可以通过TensorRT生成推理引擎，部署在硬件上

- 比较常见的组合：Pytorch -> ONNX -> TensorRT
- 目前自动驾驶的部署硬件大多都会采用NVIDIA
  - DRIVE series
  - Jetson series

什么是TensorRT

(扩展)
很多厂商除了NVIDIA，也会有很多其他选择



Qualcomm[1]

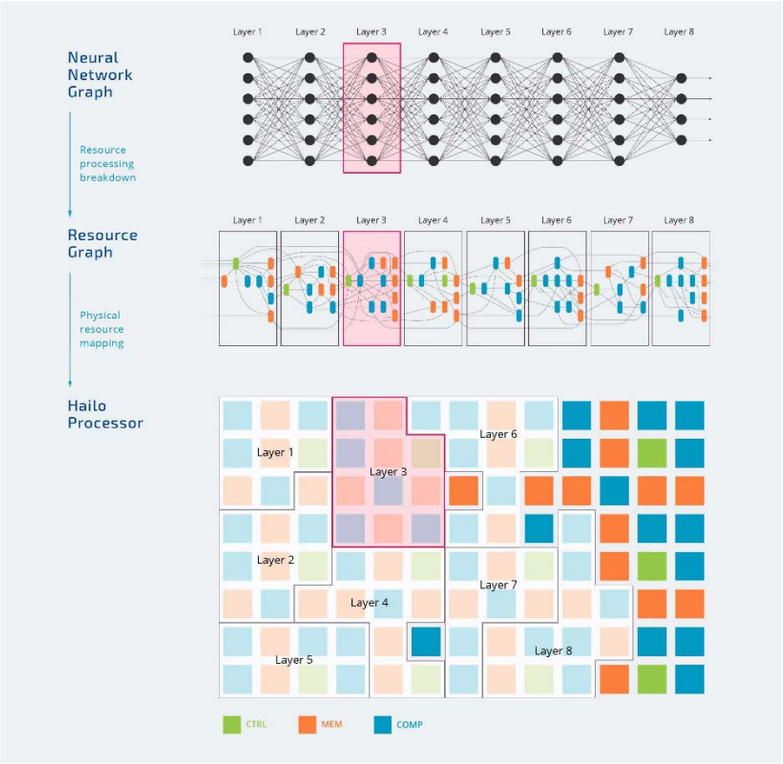
(比较典型的Heterogeneous Architecture，这里以Robotics举例，自动驾驶也差不多)

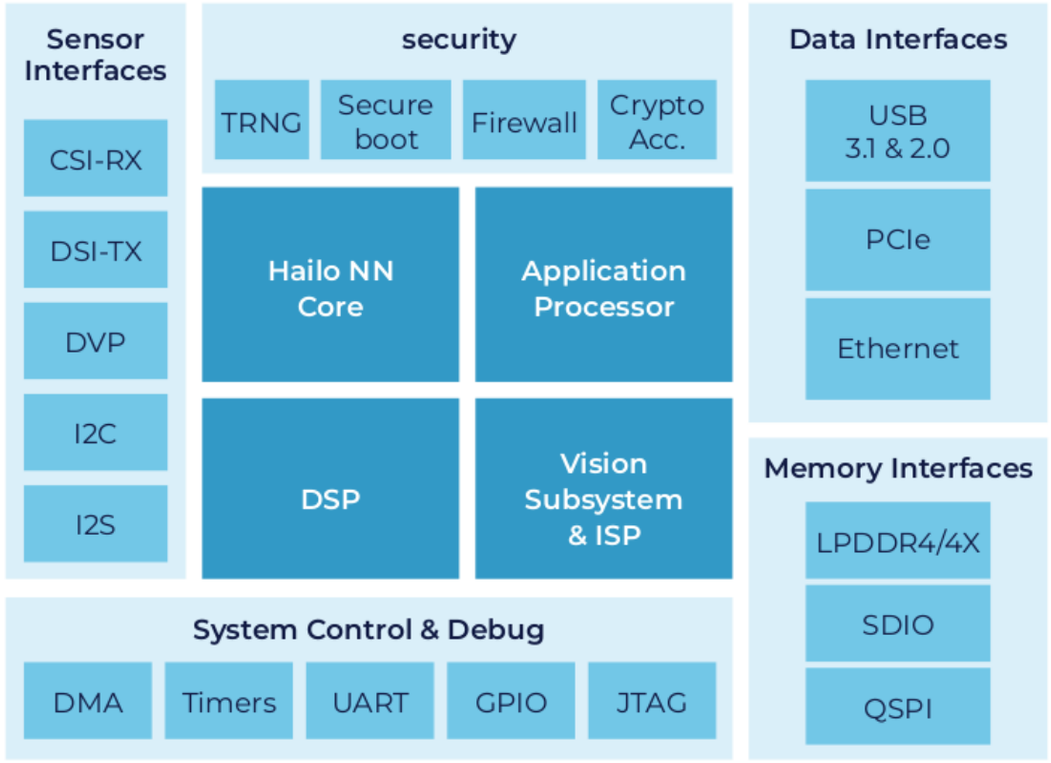
[1]Qualcomm Software Architecture

什么是TensorRT

(*)Dataflow中文叫做“数据流”，是一个很传统的概念。最近针对DNN的硬件设计有偏向Dataflow的趋势。该兴趣的同学可以自学一下。建议参考Hailo, Google TPU, MIT Eyeriss的paper

(扩展)
很多厂商除了NVIDIA，也会有很多其他选择





Hailo[1, 2]

(比较典型的小型Edge Device for DNN，有很多量化和调度的技巧，针对自己的硬件而设计的Dataflow(*) Compiler很精妙)

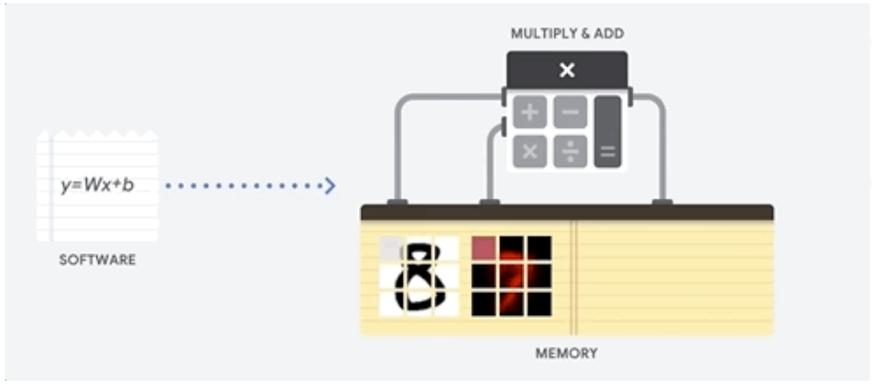
[1]Detail of Hailo AI Edge Accelerator Emerge

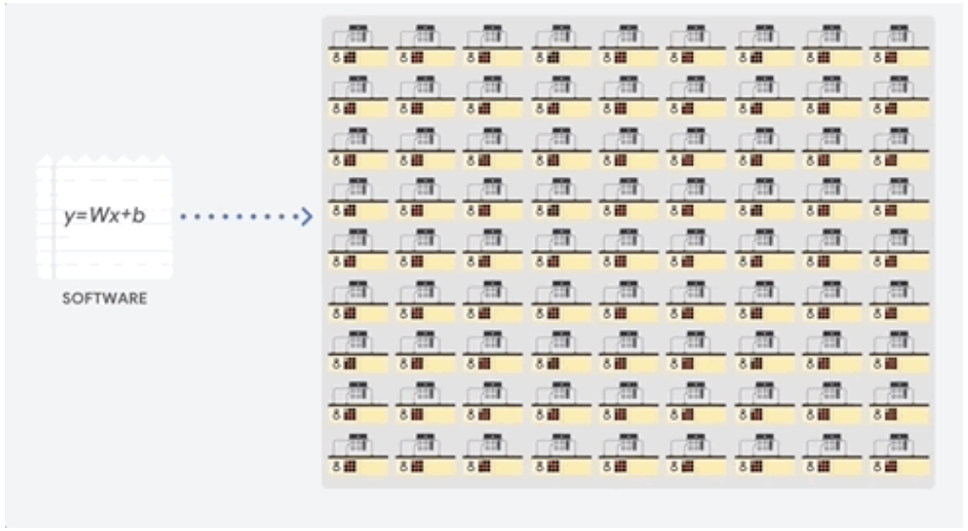
[2]Hailo-15 quad-core AI Vision processor delivers up to 20 TOPS for Smart Cameras

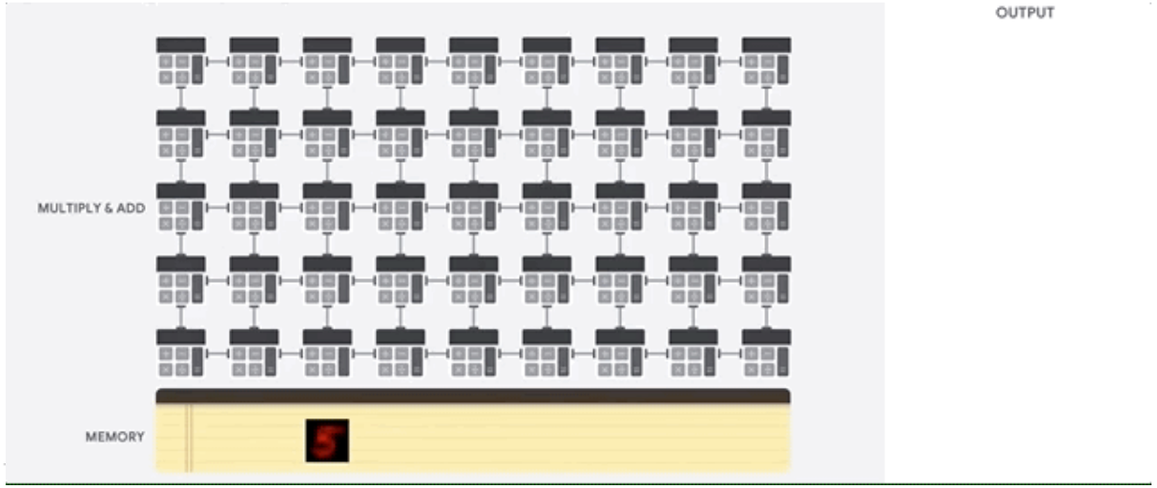
什么是TensorRT

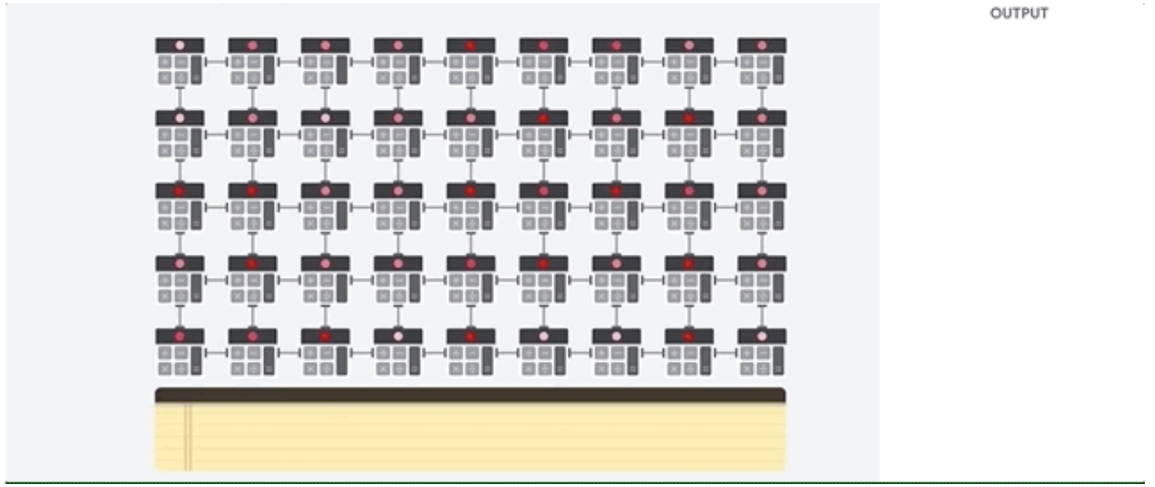
(扩展)
很多厂商除了NVIDIA，也会有很多其他选择

(*)Google的TPU也是属于dataflow architecture的一个例子。TPU内部有一个on-chip memory叫做HBM(high-bindwidth memory)，当读取数据的时候，HBM里的数据会以queue的形式放到MXU(Matrix Multiplication Unit)进行数据的流动以及计算









Google TPU(*)

左上CPU, 左下GPU, 右上右下TPU

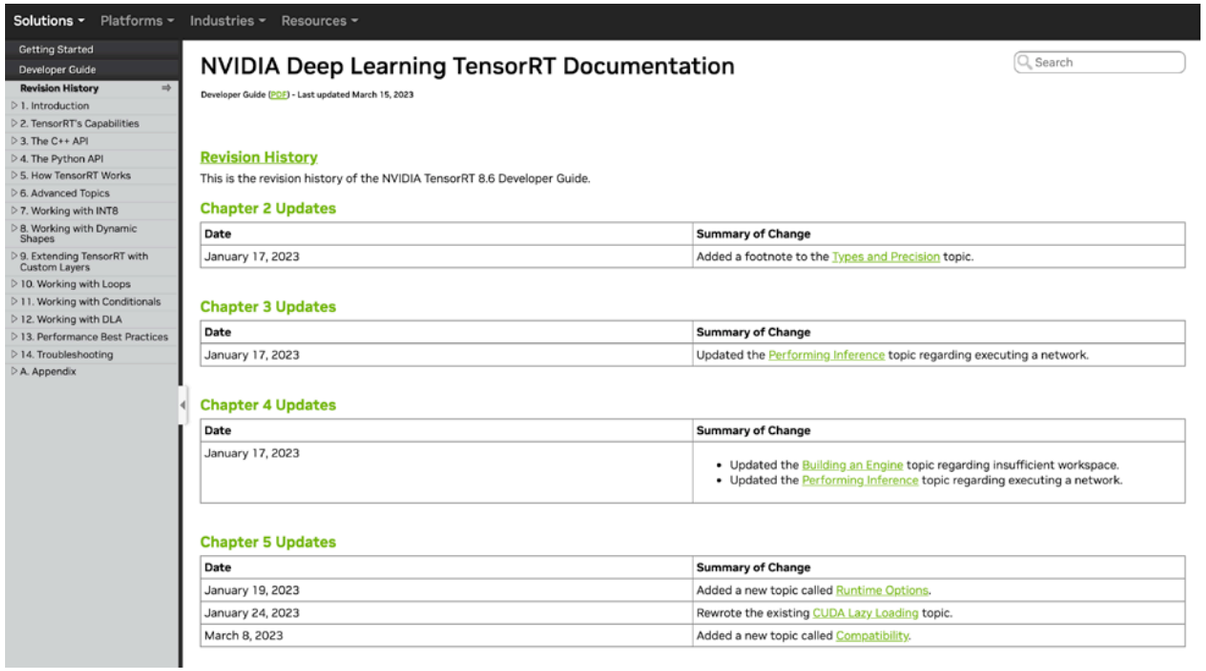
[1]Introduction to Cloud TPU

什么是TensorRT

(*)TOPS：全称是Tera Operations per second。是衡量一个硬件的计算力的常用指标。这个会在之后仔细讲。现阶段有个认识就好

回归正题，有很多部署硬件但为什么NVIDIA和TensorRT一直都很火

- 整体硬件设计和编译技术很成熟
  - BUG算比较少的那种
  - 对量化的支持比较全面以及完善
  - SDK很充足
- 可以参考的文档比较多
  - 官方给的资源还是相当丰富的
  - 给的SAMPLE也很适合学习
- Community很大
  - 出现问题的时候有的问的

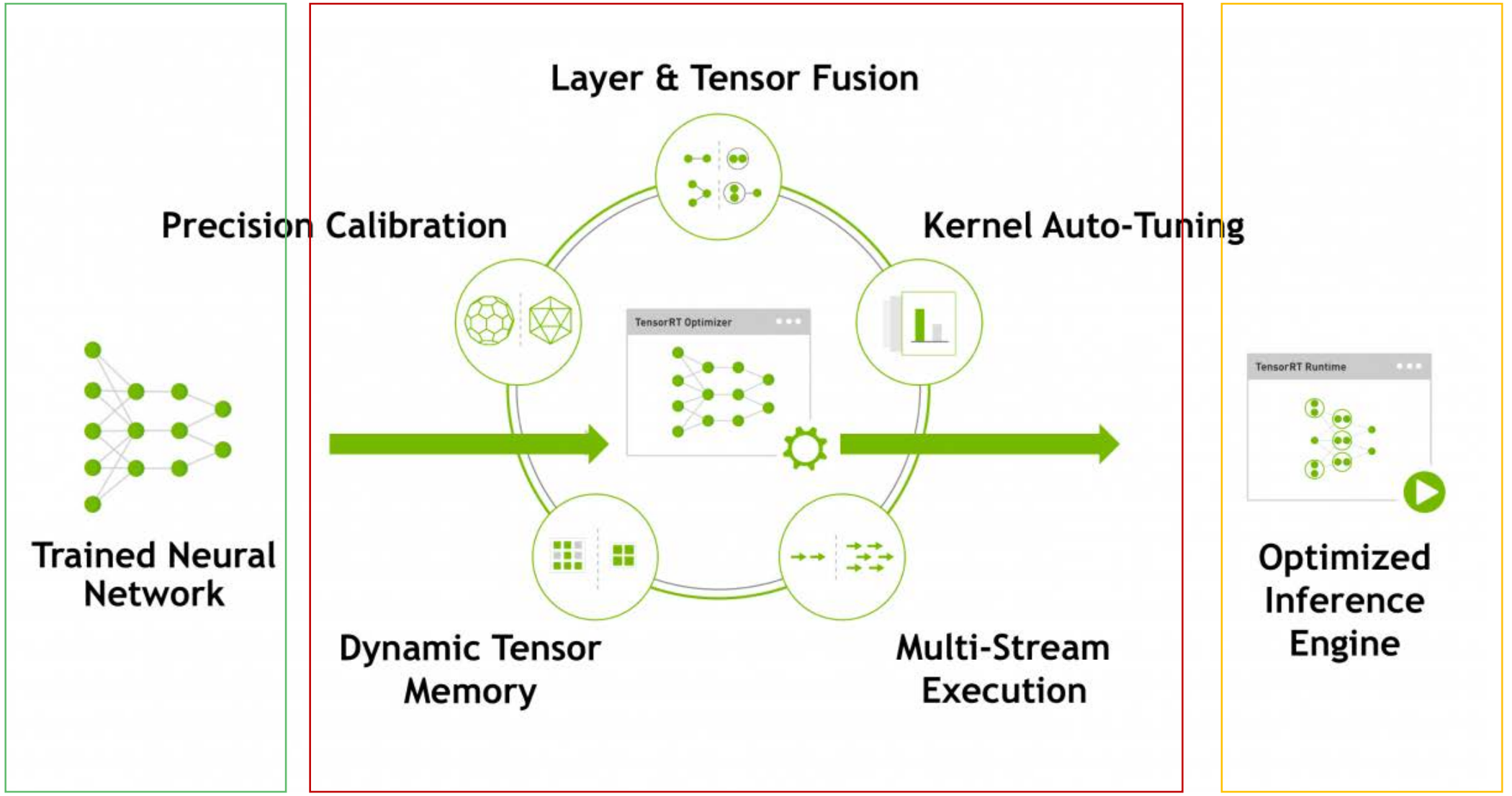


But ...?

- 价格偏贵，当面向量产的时候价格是很需要重视的
- 过大的TOPS(*)是否真的就很好?

[1]NVIDIA Deep Learning TensorRT Documentation

TensorRT的工作流介绍



训练好的模型转为TensorRT可识别的模型

TensorRT逐个Layer进行分析并尝试各种优化策略
(量化、层融合、调度等等)

生成推理引擎，可以从C++/Python程序中调用

[1]NVIDIA TensorRT

TensorRT的一些限制

针对不支持的算子

- 看看不同TensorRT版本中是否有做了支持
  - PyTorch: Facebook
  - ONNX: Microsoft
  - TensorRT: NVIDIA
- 修改PyTorch的算子选择，让它使用一些TensorRT支持的算子
- 自己写插件，内部实现自定义算子以及自定义cuda加速核函数
- 不使用onnx，自己创建parser直接调用TensorRT API逐层创建网络
  - 比如darknet模型

不同TensorRT版本的优化策略是不一样的

- 比如说，对Transformer的优化和TensorRT 7和8跑出来的性能是不一样的

有时你预期TensorRT的优化和实际的优化是不一样的

- 比如说，你期望它使用Tensor core，但kernel autotuning之后TensorRT觉得使用Tensor core反而会效率降低，结果给你分配CUDA core使用。因为内部需要做一些额外的处理

天生并行性差的layer，TensorRT也没有办法

- 1x1 conv这种layer，再怎么优化也没有7x7 conv并行效果好

TensorRT的一些限制

重点！！

TensorRT虽然很方便，但不要过分的依赖TensorRT给你的结果。你需要结合部署的硬件特性，做一些Benchmark和Profiling，学习使用一些Profile tools去分析模型的计算瓶颈在哪里，以及与分析你的预测优化策略和实际优化策略产生分歧的原因在哪里。

TensorRT的应用场景

Goal: 理解TensorRT在自动驾驶中的意义,自动驾驶在模型部署所关注的内容

TensorRT的优化意义

模型训练 (Training)

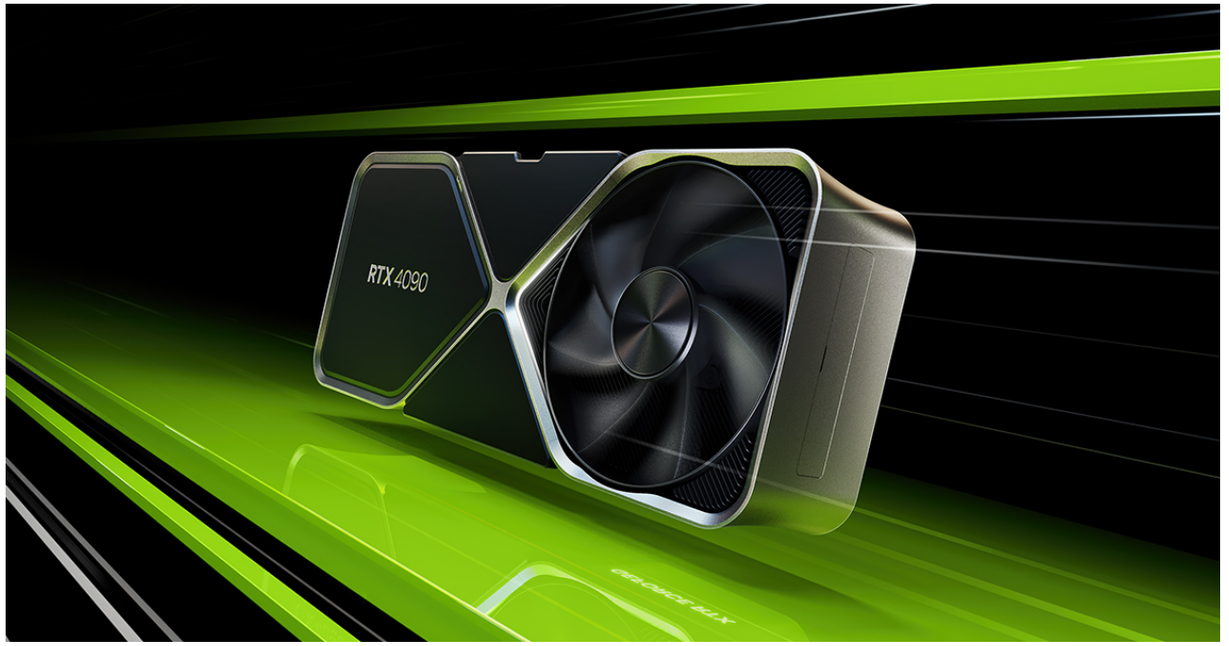
- 精度越高越好
  - 把模型做的深一点，宽一点
  - 使用丰富的Data augmentation
  - 使用各种Training trick

模型部署 (Deploy)

- 以精度不变，或者精度掉点很小的情况下尽量压缩模型
  - 减少计算量
  - 减少memory access
  - 提高计算密度

TensorRT的优化意义

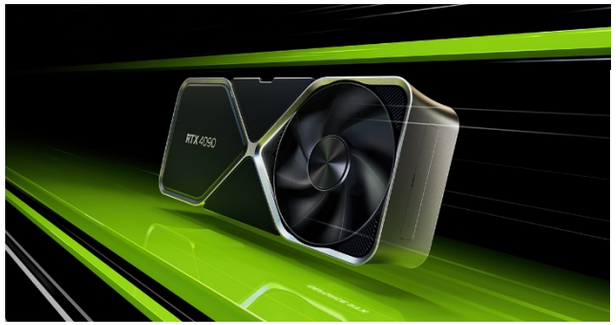
Training environment
(训练时的GPU资源可以很丰富，多个GPU分布式训练也都是完全Okay的)



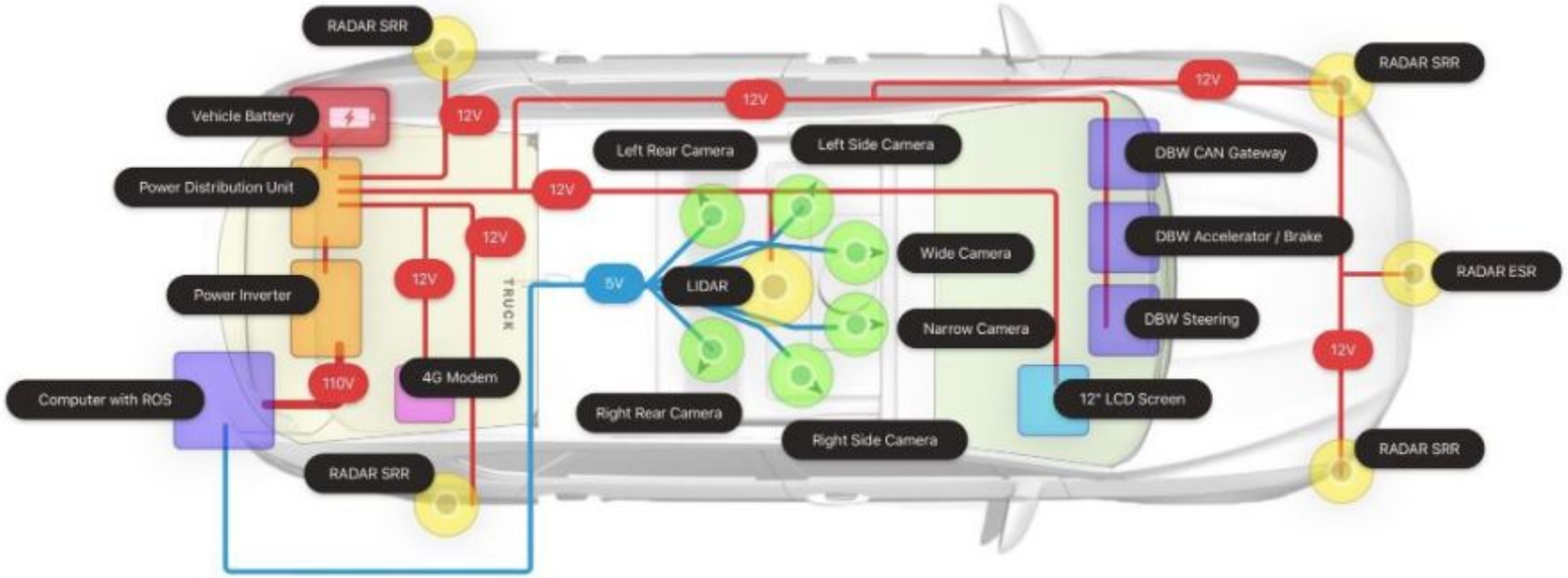
Depoly environment
(我们不可能在一个自动驾驶车上放那么大的服务器，并且太贵。所以选择一个小一点的)

[1]NVIDIA DGX System
[2]GeForce RTX 4090

TensorRT的优化意义



Depoly environment[1, 2]
(我们得考虑GPU的散热，因为经常一不小心就到了80~90°，所以优先考虑水冷)



自动驾驶车中的硬件设备[3]
(硬件设备多，如果是水冷的话对整个汽车的安全性影响很大)

[1]GeForce RTX 4090
[2]bizon z5000
[3]Under the Hood of a Self-Driving Taxi, a Look at Computation and Other Core Self-Driving Car Systems

TensorRT的优化意义

NVIDIA Jetson AGX Xavier[1]

NVIDIA Jetson AGX Orin[2]

Deploy environment:NVIDIA Jetson series
(又要性能高，又要不那么热，那么很多地方就会把
视线集中在Jetson系列上)

[1]Jetson AGX Xavier
[2]Jetson AGX Orin at GTC 2022

自动驾驶中需要关注的实时性

Realtime性(实时性)

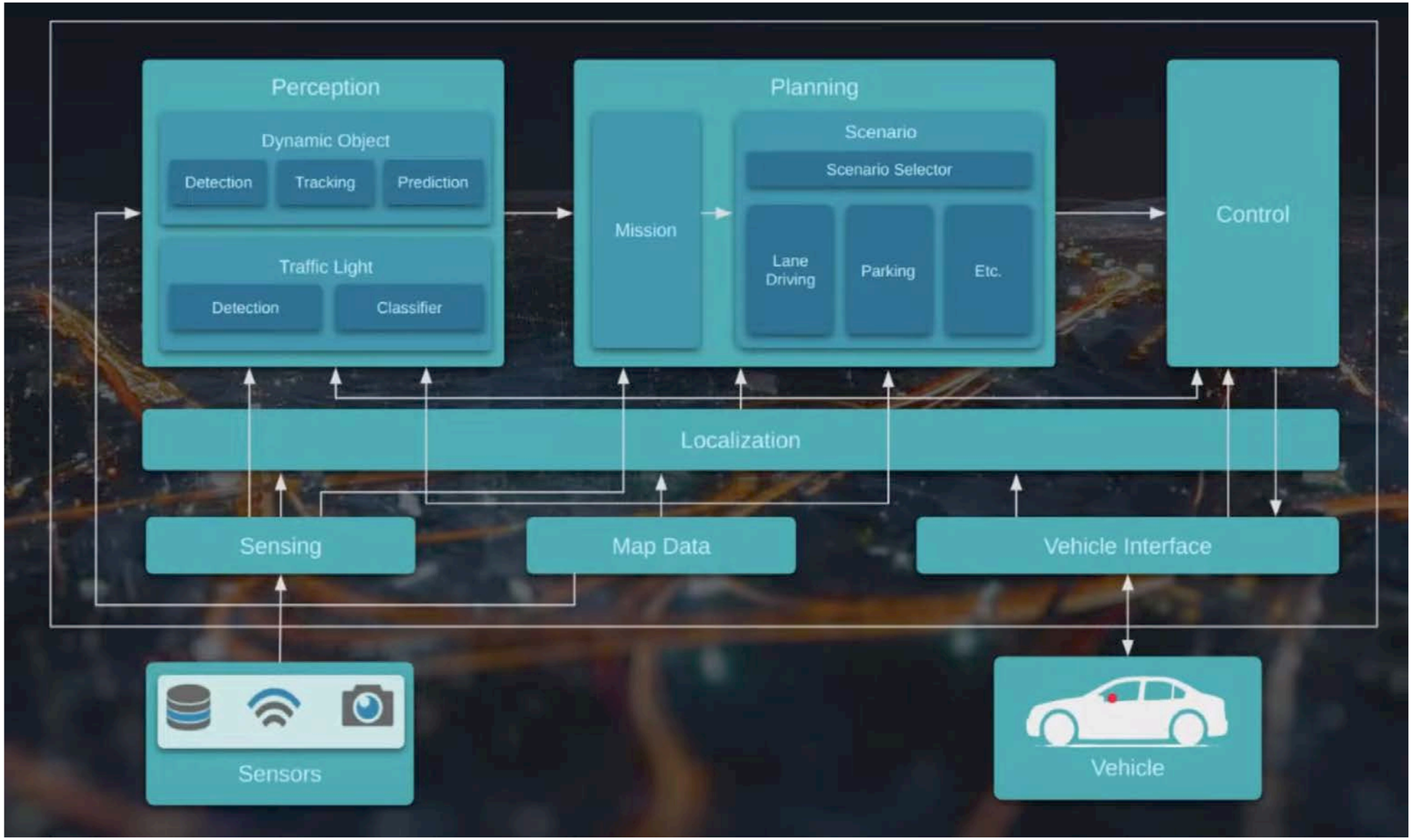
- 我们在部署模型是需要关注的一个很重要的指标
  - 一般来说15 fps是一个底线 (高速公路上需要至少30fps)
  - 100km/h = 28m/s

- 对于单个小型DNN，在计算资源限制的情况下想达到这个指标难度系数不高
  - 模型可以做的很simple

- 但我们又得需要看的很远
  - 比如说150m远的红绿灯，以及红绿灯下面的箭头和时间
  - 比如说100m远的路标

- 不光是Perception，后期的Planning等等也需要时间

自动驾驶中需要关注的实时性



[1]自動運転におけるAIコンピューティング

自动驾驶中需要关注的实时性

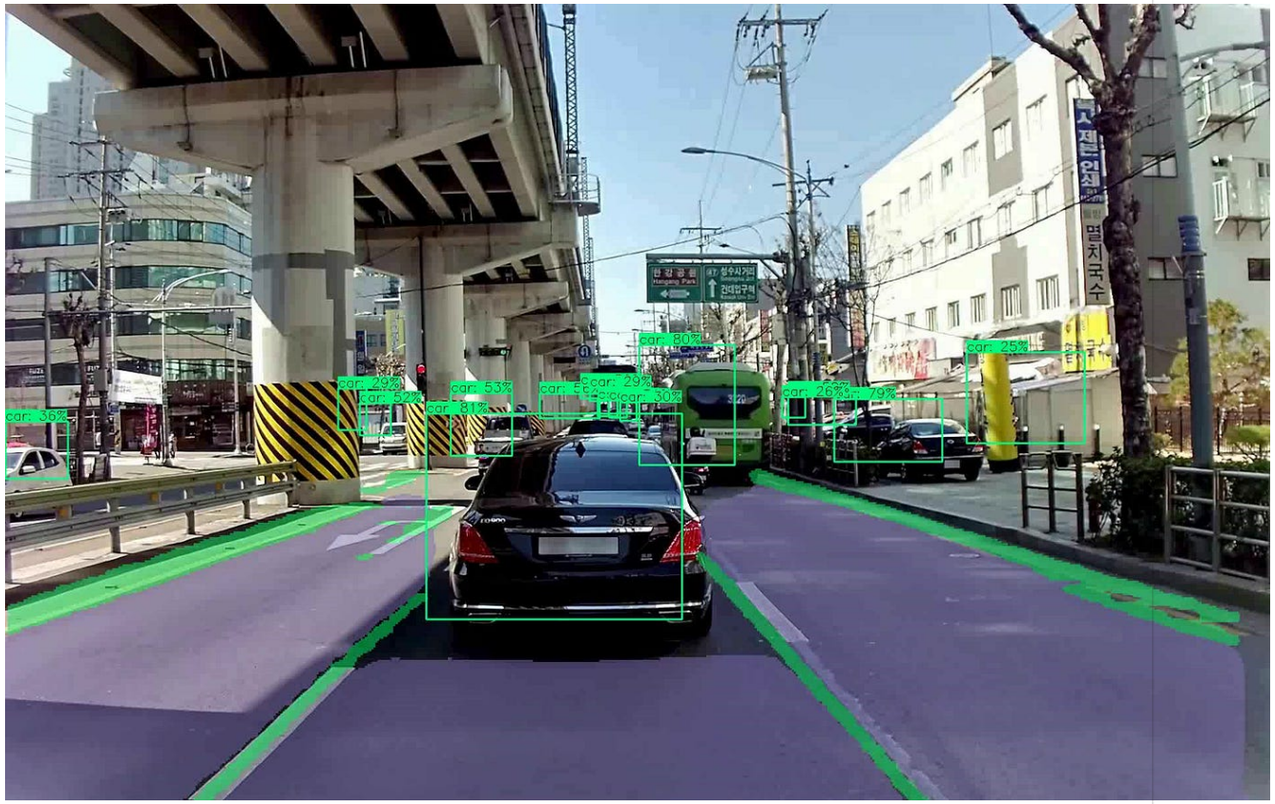
But...

- 多任务
  - 2D/3D 检测
  - 车道线检测
  - 图像分割
  - 深度估计
  - 多目标追踪
  - BEV检测
  - 光流预测
- 多传感器
  - LiDAR
  - Camera
  - Radar
- 高分辨率输入图像
- Multi-camera

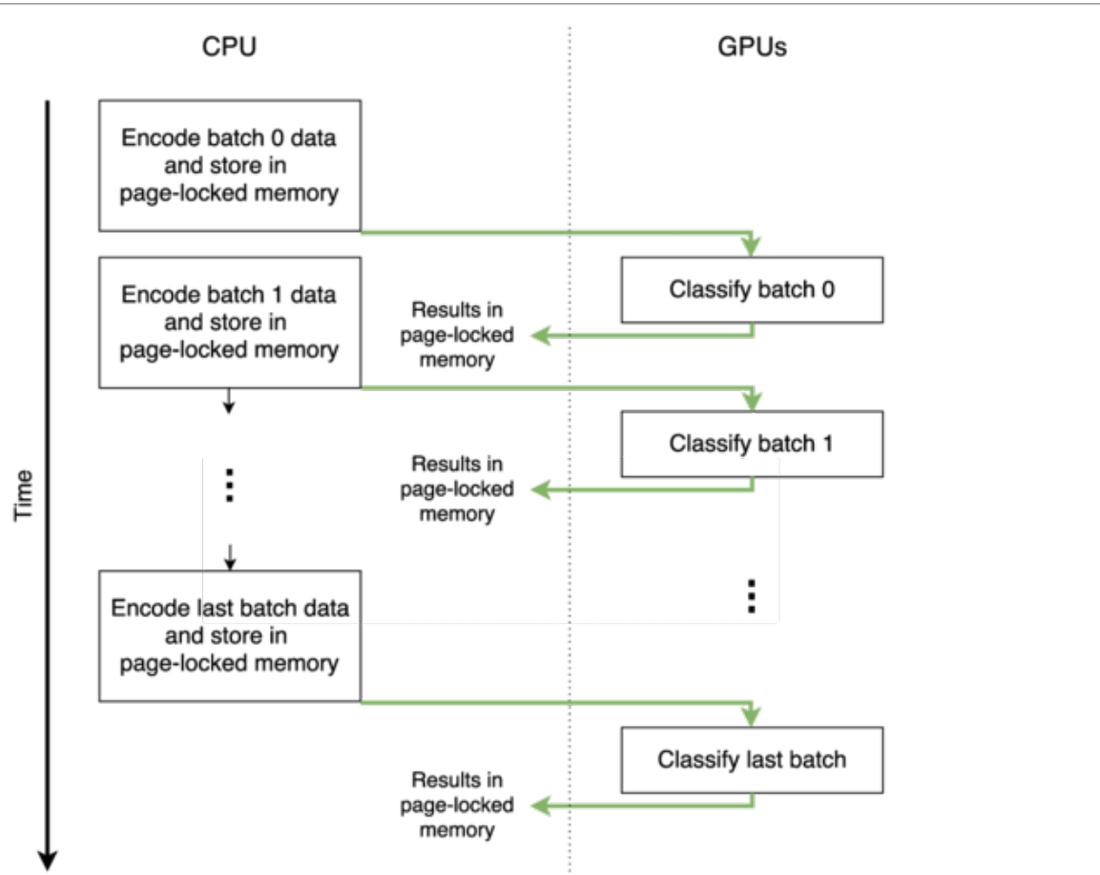
如果把这些所有的事情能够在几短时间内做完并达到>10fps，甚至30fps，是一个非常具有挑战性的事情

自动驾驶中需要关注的实时性

我们怎么做呢？工业界中有很多种方法，比如说：



Multi-task DNN in autonomous driving[1]



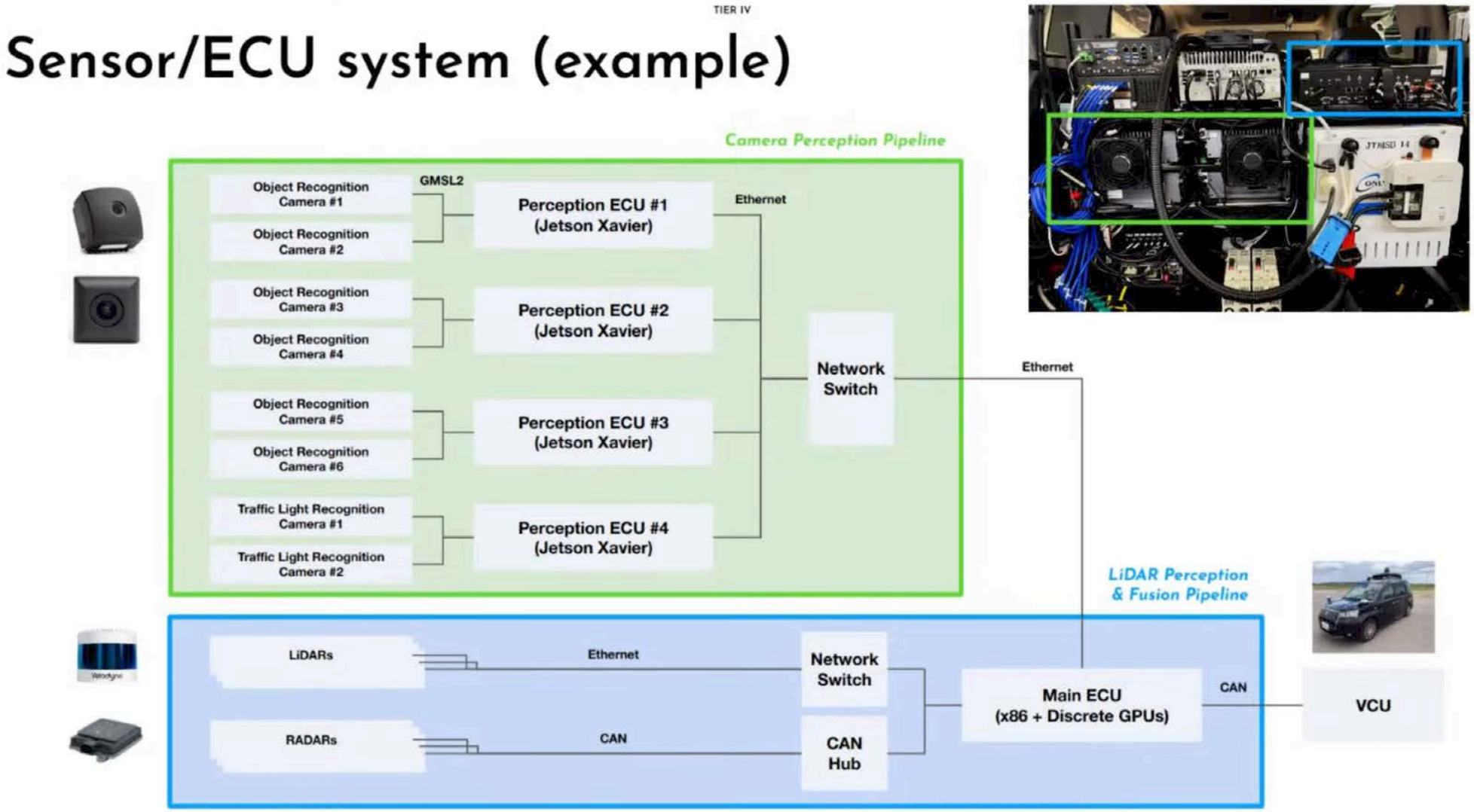
Asynchronous processing in GPU[2]

再比如局部提高分辨率
优先处理局部区域等等

[1]Visual Perception for Self-Driving Cars! Part 5: Multi-Task Learning
[2]Accelerating metagenomic read classification on CUDA-enabled GPUs

自动驾驶中需要关注的电力消耗

Main ECU (CPU + GPU) ~150W
Perception ECU (Object detection * 6) = ~90w
Perception ECU (Traffic light* 2) = ~20w
合计下来把所有的这些东西全部搭上只做最基本的优化的话大概~255w



自动驾驶ECU方案举例
(camera: yolox, LiDAR: centetpoint)

[1]自動運転におけるAIコンピューティング

自动驾驶中需要关注的电力消耗

基本上我们需要的算力越大，消耗的电力越大。自动驾驶所需要控制的电力一般来说需要控制在小于100w。自动驾驶也属于Edge computing的一种，所以对电力消耗需要严格控制

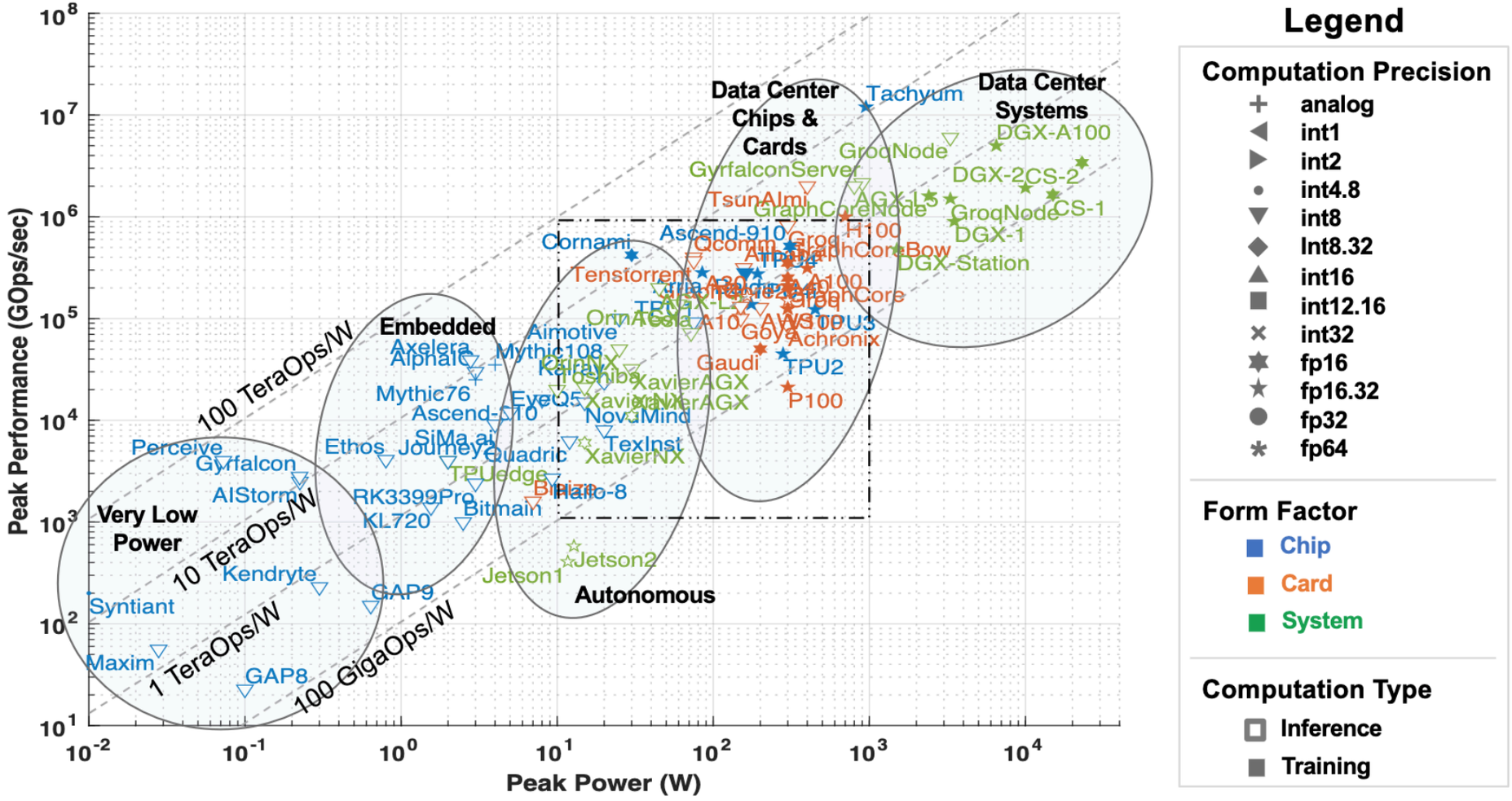


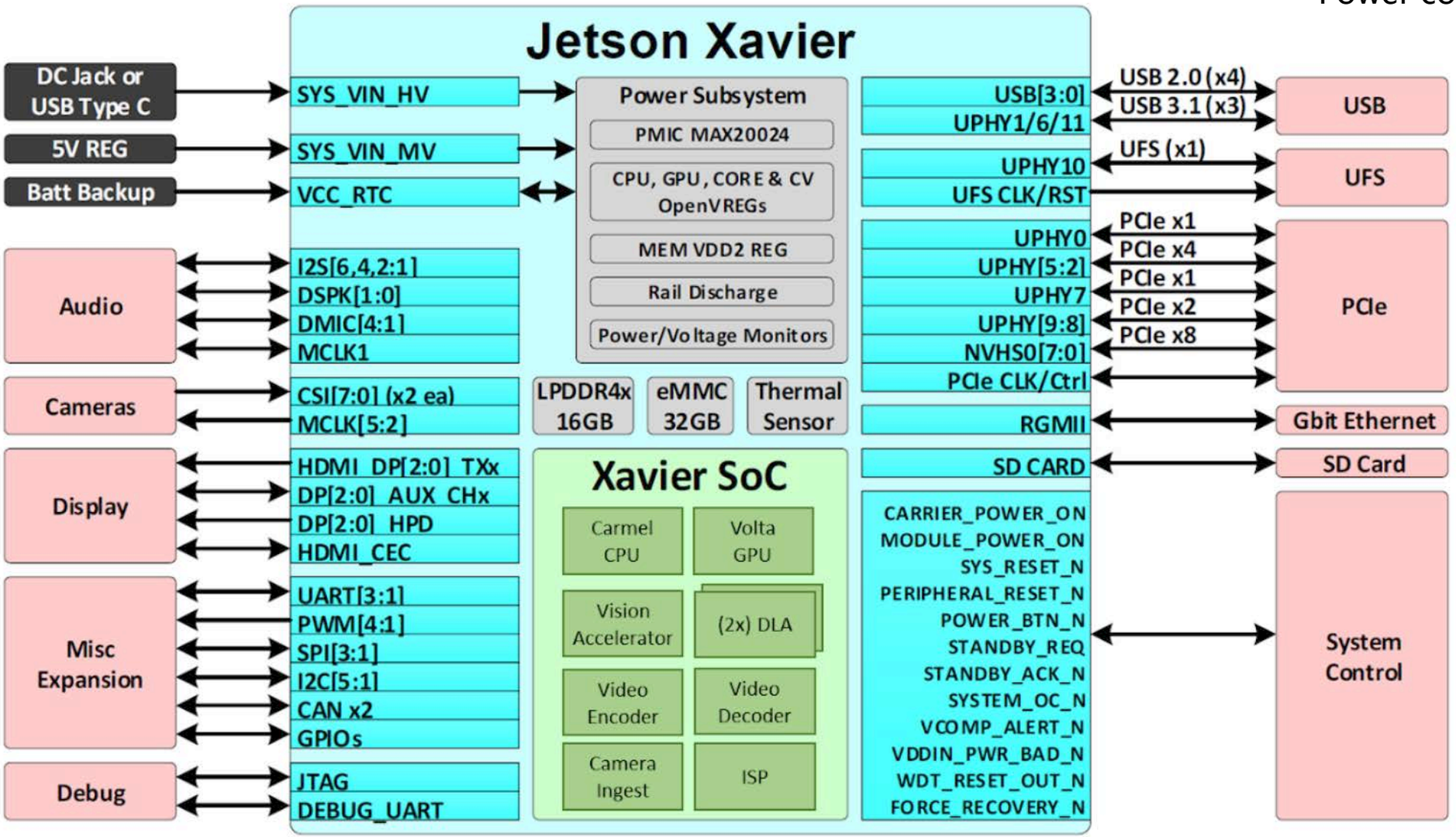
Fig. 2: Peak performance vs. power scatter plot of publicly announced AI accelerators and processors.

[1]AI and ML Accelerator Survey and Trends

自动驾驶中需要关注的电力消耗

NVIDIA Jetson AGX Xavier

Peak performance
GPU: 22TOPS (DENSE INT8)
DLA0: 5TOPS (DENSE INT8)
DLA1: 5 TOPS (DENSE INT8)
Power consumption ~30W

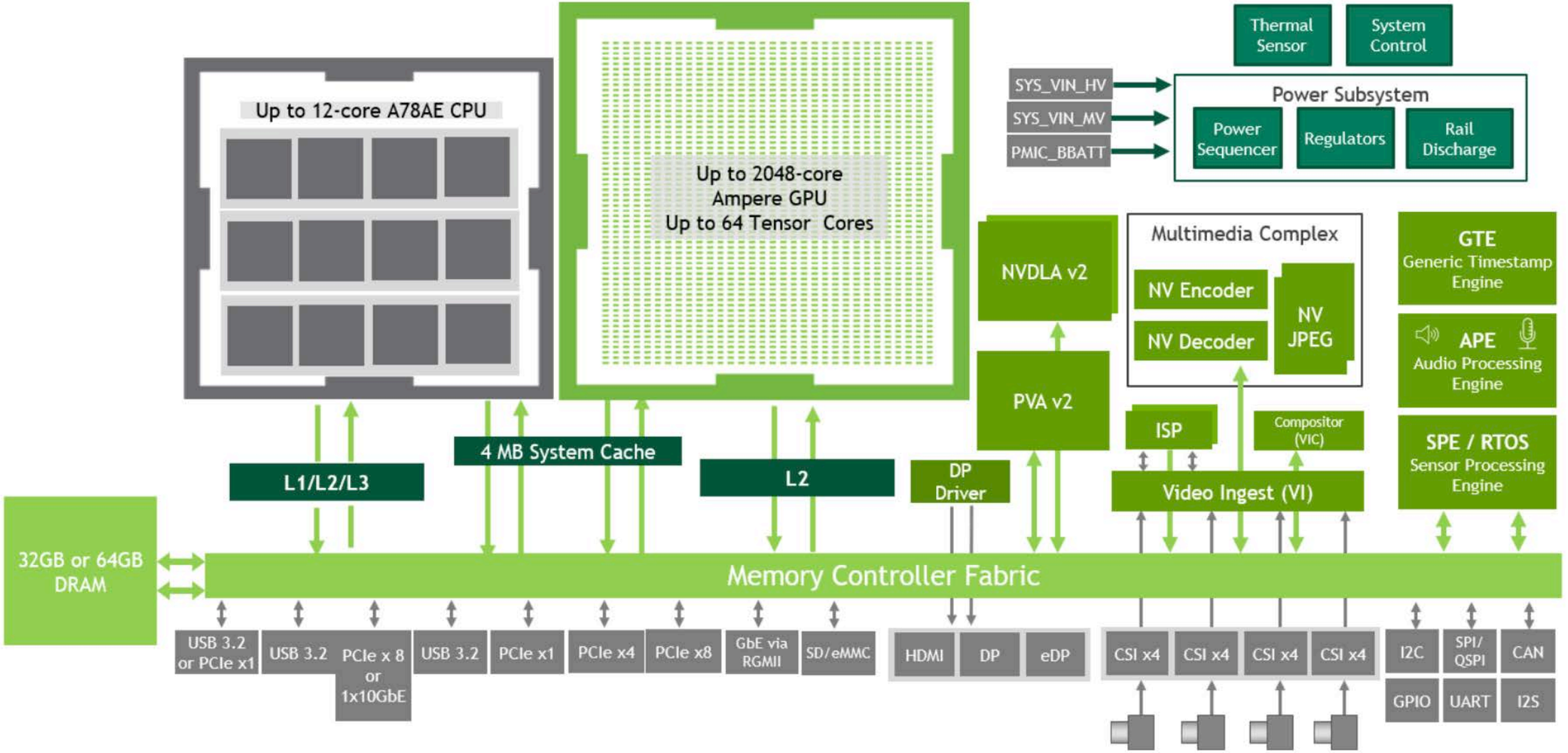


[1]NVIDIA Jetson AGX Xavier Delivers 32 TeraOps for New Era of AI in Robotics

自动驾驶中需要关注的电力消耗

NVIDIA Jetson AGX Orin

Peak performance
GPU: 170 TOPS (Sparse INT8), 85TOPS (DENSE INT8)
DLA0: 52.5 TOPS (Sparse INT8), 26.25TOPS (DENSE INT8)
DLA1: 52.5 TOPS (Sparse INT8), 26.25TOPS (DENSE INT8)
Power consumption ~60W



[1]Delivering Server-Class Performance at the Edge with NVIDIA Jetson Orin

自动驾驶中模型部署所关注的东西

重点！！

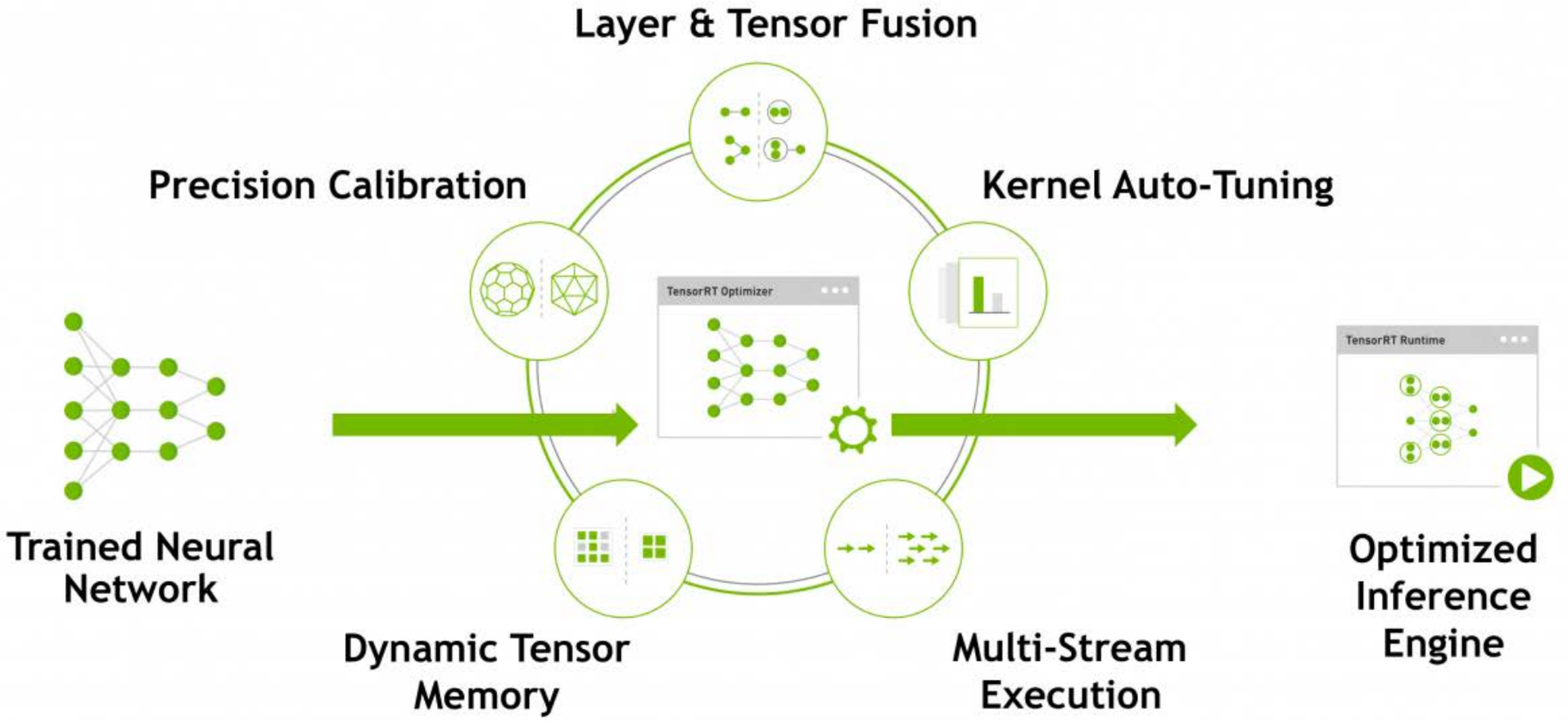
RealTime(实时性)
Power Consumption(消耗电力)
Long range Accuracy(远距离)

TensorRT的模块

Goal: 简单理解TensorRT中每个优化策略，为什么可以做层融合，以及理解量化的意义

TensorRT的优化策略

回顾一下



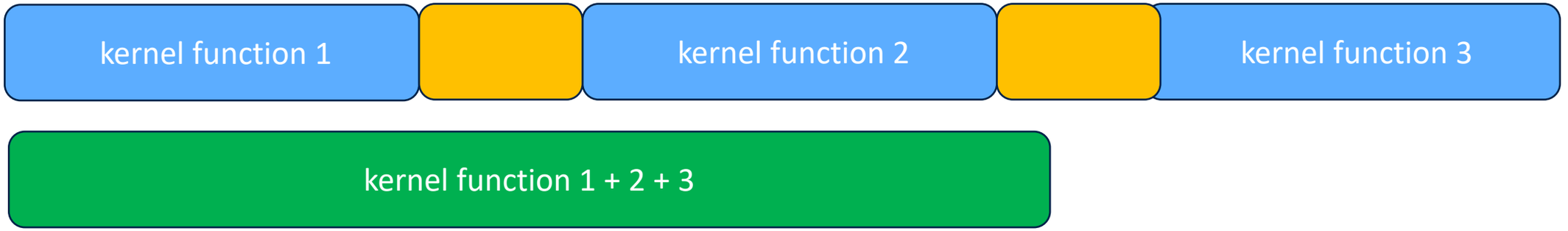
TensorRT的各类优化策略[1]

[1]NVIDIA TensorRT

TensorRT的优化策略

Layer fusion

- 层融合
  - Vertical layer fusion (垂直层融合)
  - Horizontal layer fusion (水平层融合)



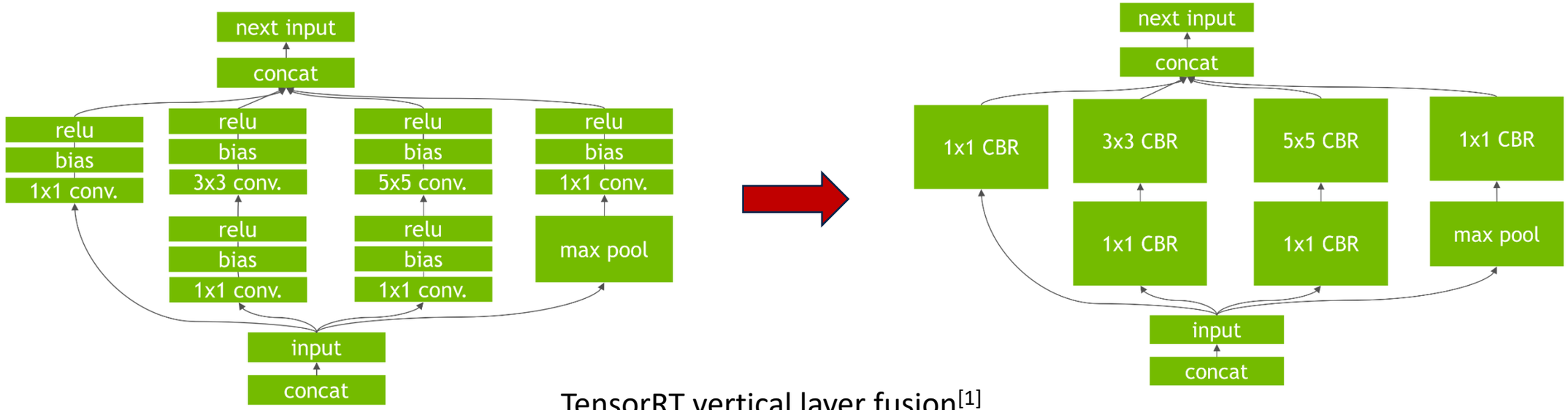
层融合可以减少启动kernel的开销与memory操作，从而提高效率
同时，有些计算可以通过层融合优化后，跟其他计算合并

TensorRT的优化策略

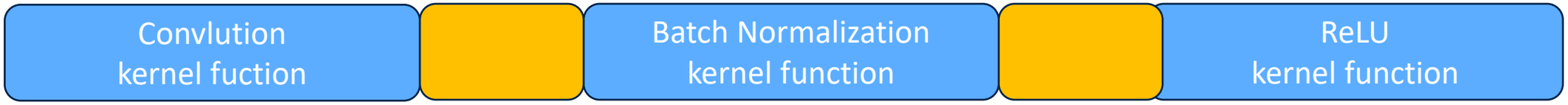
[1]TensorRT 3: Faster TensorFlow Inference and Volta Support

Layer fusion

- 层融合
  - Vertical layer fusion (垂直层融合)
    - 用的比较常见，针对conv + BN + ReLU进行融合



TensorRT vertical layer fusion[1]



Conv + BN + ReLU
kernel fuction

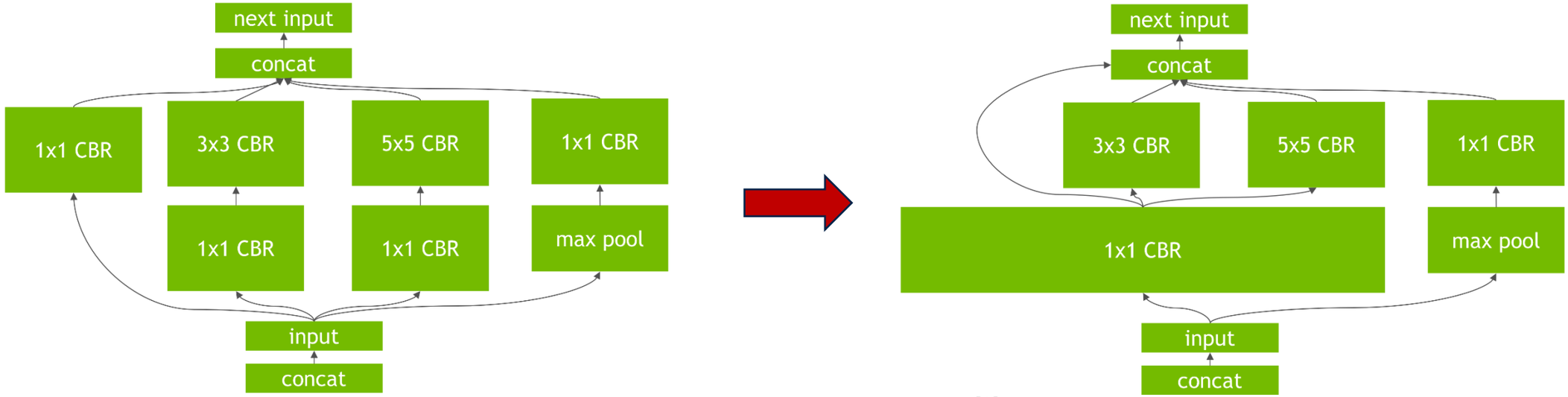
思考：为什么conv + BN + ReLU的计算时间和conv的计算时间差不多？

TensorRT的优化策略

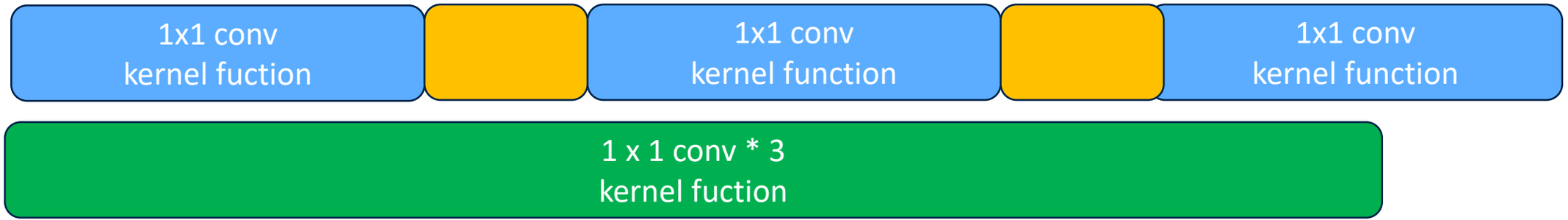
[1]TensorRT 3: Faster TensorFlow Inference and Volta Support

Layer fusion

- 层融合
  - Horizontal layer fusion (水平层融合)
    - 当模型中有水平方向上比较多的同类layer，会直接进行融合

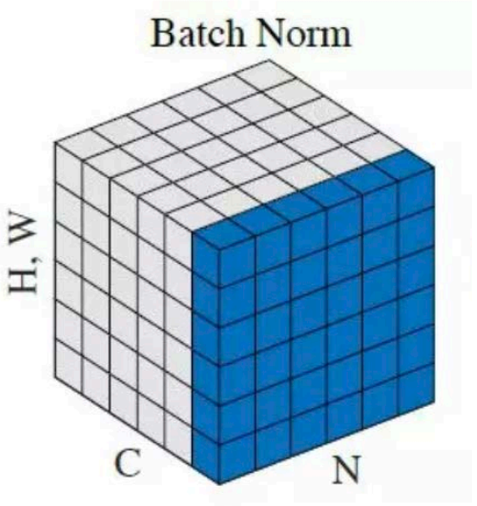


TensorRT horizontal layer fusion[1]



TensorRT的优化策略

回顾一下Batch Normalization的公式



$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} x_i$$

$$\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(x_i - \mu_B)^2 + \epsilon$$

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y_i = \gamma * \hat{x_i} + \beta$$

μ_B：代表均值
σ_B^2：代表方差
ϵ：一个大于0的浮点数，用于防止分母为0
γ：scaling，缩放因子
β：shift，偏移因子

上面这些数值在推理的时候可以提前计算出来

TensorRT的优化策略

convolution + batch normalization融合

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} x_i$$

$$\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(x_i - \mu_B)^2 + \epsilon$$

$$\widehat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y_i = \gamma * \widehat{x_i} + \beta$$

展开

$$y_i = \gamma * \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

代入 $x_i = w * x_i + b$

$$y = \gamma * \frac{w * x + b - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

TensorRT的优化策略

convolution + batch normalization融合

$$y = \gamma * \frac{w * x + b - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

$$y = \frac{\gamma * w}{\sqrt{\sigma_B^2 + \epsilon}} * x + \frac{\gamma}{\sqrt{\sigma_B^2 + \epsilon}}(b - \mu_B) + \beta$$

$$y = (\frac{\gamma * w}{\sqrt{\sigma_B^2 + \epsilon}}) * x + (\frac{\gamma}{\sqrt{\sigma_B^2 + \epsilon}}(b - \mu_B) + \beta)$$

定义

$$\hat{w} = \frac{\gamma * w}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$\hat{b} = \frac{\gamma}{\sqrt{\sigma_B^2 + \epsilon}}(b - \mu_B) + \beta$$

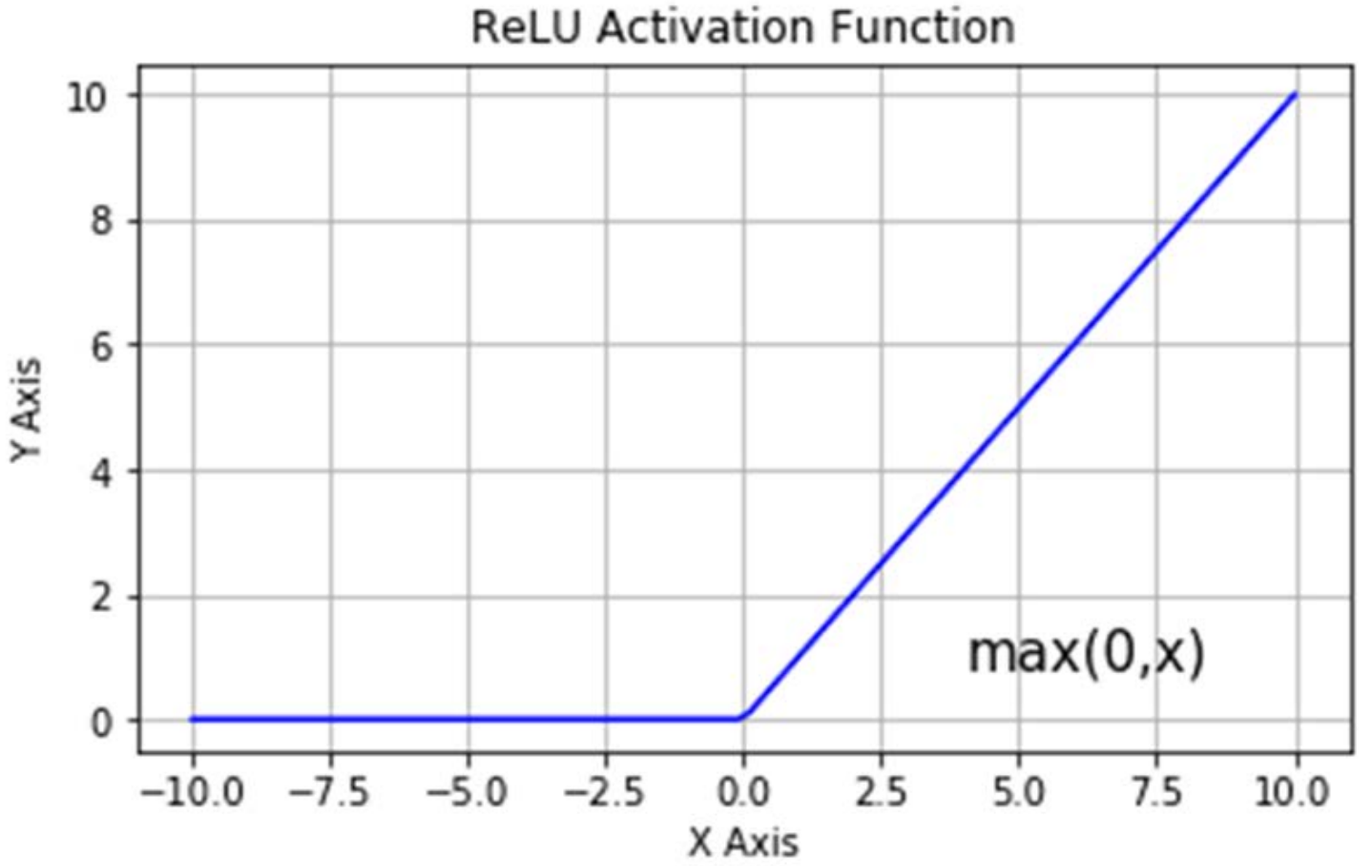
$$y = \hat{w} * x + \hat{b}$$

这两个参数值可以提前计算出来

TensorRT的优化策略

convolution + batch normalization + ReLU 融合

$$y = \hat{w} * x + \hat{b}$$



ReLU function[1]

(ReLU只是做一个截取>0的值，内部没有计算)

TIPS:

最近的很多模型经常会有很多种类的activation function，比如GELU, Swish, Mish等等，这些激活函数往往由于计算复杂很难加速，可以尝试改成ReLU看看精度的改变和性能的提升

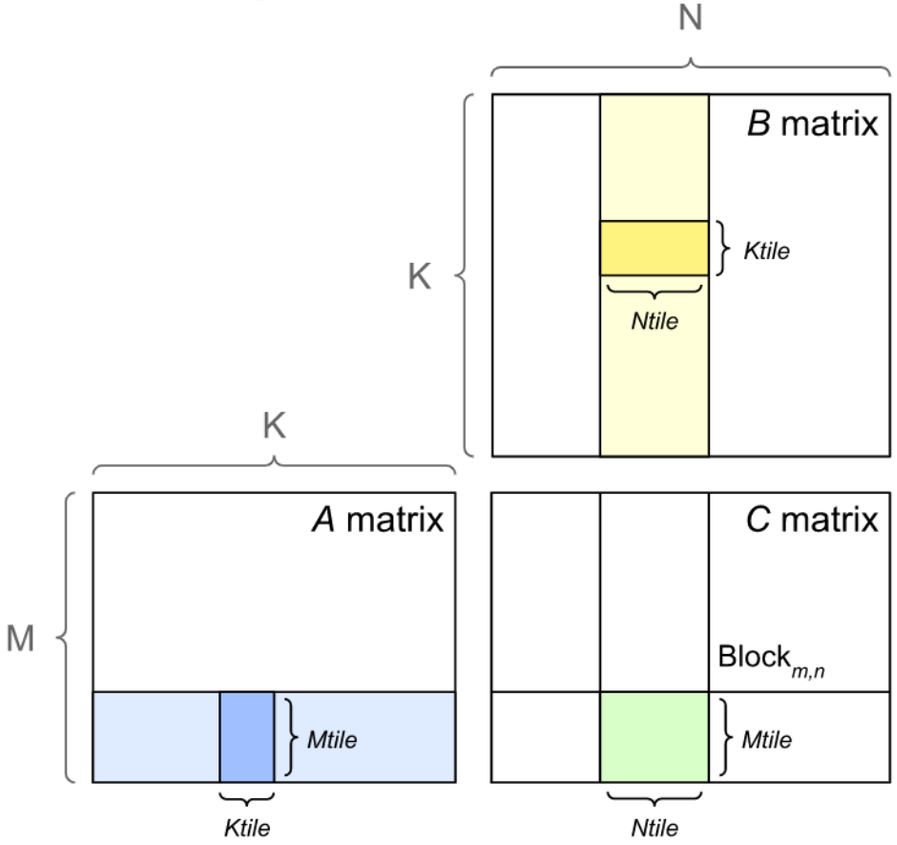
[1]What is ReLU and Sigmoid activation function?

TensorRT的优化策略

[1]Matrix Multiplication Background User's Guide
[2]Understanding GPU Architecture: Inside a Volta SM

Kernel auto-tuning

- TensorRT内部对于同一个层使用各种不同kernel函数进行性能测试
  - 比如对于FC层中的矩阵乘法，根据tile size有很多中kernel function
  - (e.g. 32x32, 32x64, 64x64, 64x128, 128x128，针对不同硬件有不同策略)



GPU的GEMM计算[1]

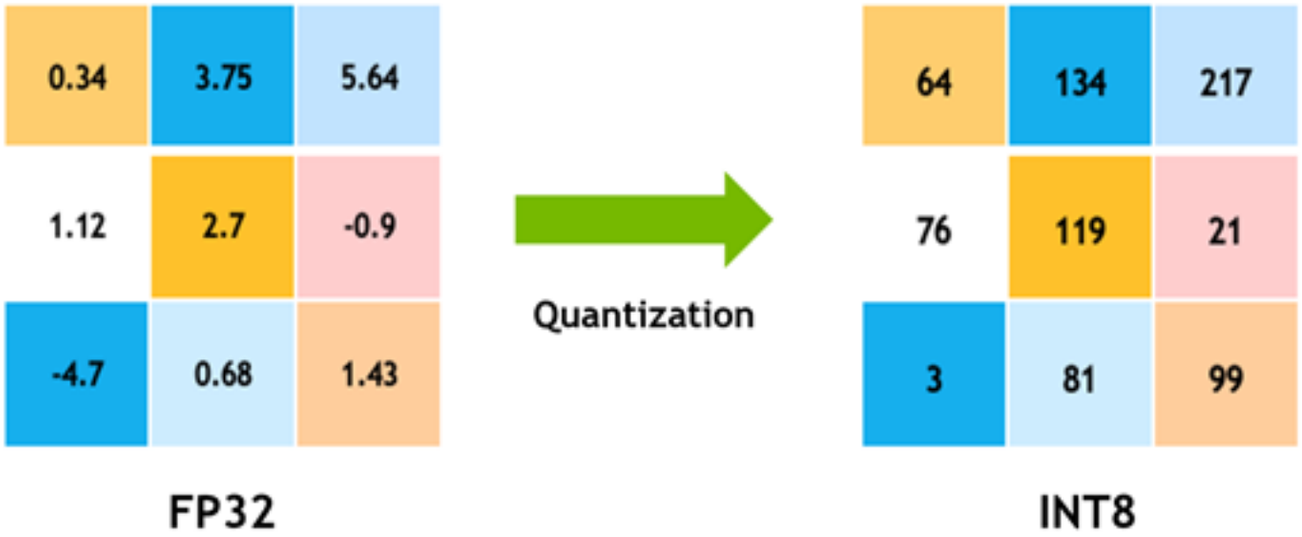
Volta architecture block diagram[2]

TIPS:
有时你的模型架构由于计算密度不够高的原因，在kernel auto tuning时TensorRT认为没必要选择tensor core而给你分配CUDA core使用

TensorRT的优化策略

Quantization

- 量化
  - 压缩模型的一个很重要的策略
  - 将单精度类型(FP32)训练权重转变为半精度(FP16)或者整型(INT8, INT4)



FP32 -> INT8的量化[1]

[1]Achieving FP32 Accuracy for INT8 Inference Using Quantization Aware Training with NVIDIA TensorRT

TensorRT的优化策略

(*)IEEE 745 standard是IEEE于1985年制定的表示浮点数的一种规格

sign
range
Precision

Quantization

- 如何表示浮点数？(e.g. -6.650490625...)
  - 很显然，我们用8bit是无法表达这种小数的
  - IEEE 745 standard(*) 规定了一种表示浮点数(floating point)的方法
    - 一个浮点数可以分为三个部分：
      - sign
      - expond (range)
      - significant (fraction, precision)

TensorRT的优化策略

sign
range
Precision

Quantization

- 如何表示浮点数？(e.g. -6.650490625...)

FP16

1	1	0	1	0	1	1	0	1	0	1	0	1	0	0	1

sign (1 bit) | expond (5 bit) | significant (10 bit)

1

sign = $-1^1 = -1$

1	0	0	0	1
2^4	2^3	2^2	2^1	2^0

expond = $2^4 + 2^0 = 17$

bias = $2^{k-1} = 2^4 = 16$

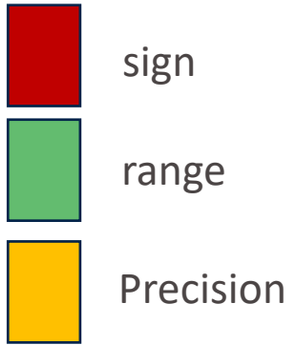
biased expoond = $expond - bias = 1$

0	0	1	0	1	0	1	0	0	1
2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}

significant = $2^{-1} + 2^{-3} + 2^{-5} + 2^{-7} + 2^{-10} = 0.6650390625$

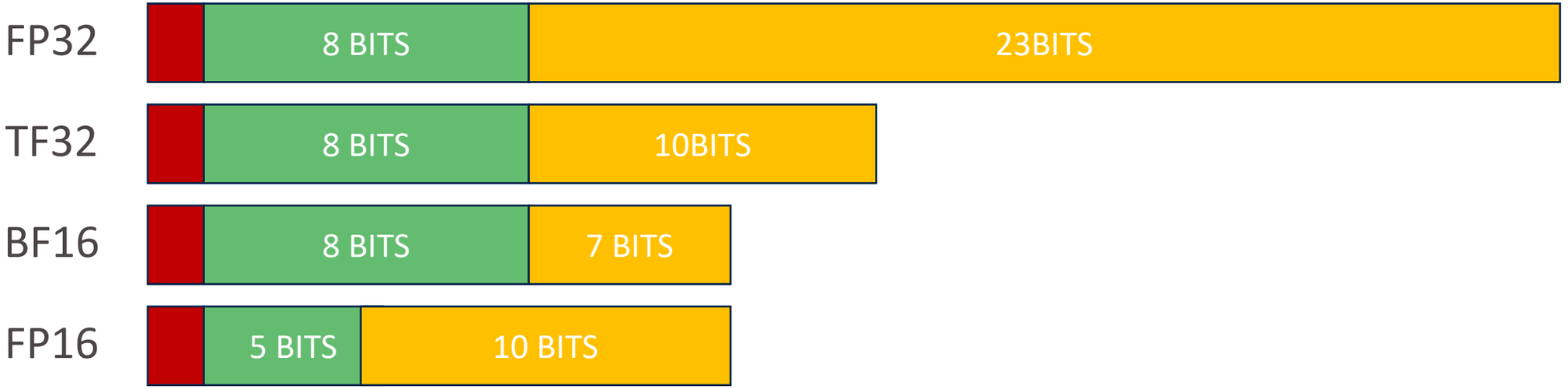
$$-1 \cdot 0.6650390625 \cdot 2^1 = -1.33078124$$

TensorRT的优化策略



Quantization

- 如何表示浮点数?
  - 表示浮点数的方式有很多。
    - FP32, TF32, BF16, FP16
    - 所占用的bits越多，表示的精度越高，但随之而来的就是占用的内存空间越大，计算时间越长



TensorRT的优化策略

(*)其实在Training的时候为了加快速度，也在使用量化的技巧。这个方法被称作Mixed precision training(混合精度学习)。感兴趣的同学可以阅读一下Baidu和NVIDIA发表在ICLR2018的”Mixed precision training[1]”,很有意思

Quantization

- 量化
  - 训练的时候，因为需要优先考虑精度而不需要太重视速度(*)，所以会使用FP32来表示权重和激活值
  - 但部署的时候，我们需要想办法把FP32的数据尽量压缩，能够用16bits，8bits，甚至4 bits来表示它们
  - 这个过程被称作量化(Quantization)

- 由于太过于重要，技巧很多，并且也是目前模型部署最优先考虑的策略，所以下一章单独拿出来详细介绍。目前只需要知道量化是干什么的就okay了

[1]Mixed precision training

导出ONNX，分析ONNX

Goal: 学习PyTorch导出ONNX的方法，以及onnxsimplify的使用

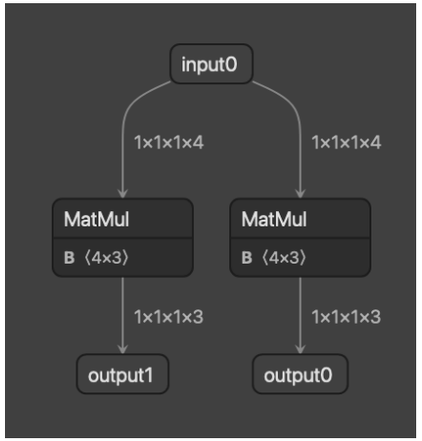
执行一下我们的python程序

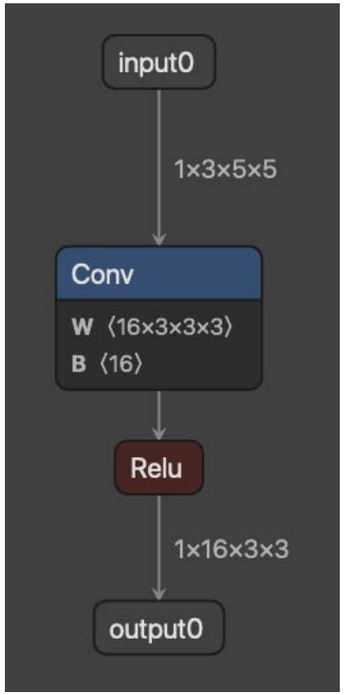
```
3.1-generate-onnx/
├── model
│   ├── example_dynamic_shape.onnx
│   ├── example.onnx
│   └── example_two_head.onnx
└── src
    ├── example_dynamic_shape.py
    ├── example.py
    └── example_two_head.py
```

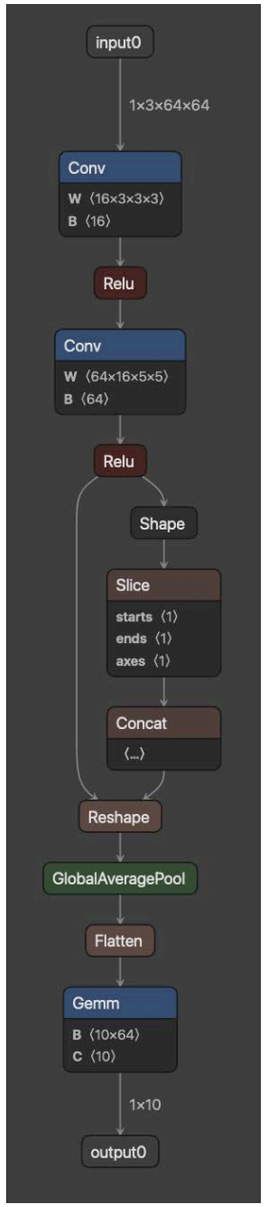
```
3.2-export-onnx/
├── models
│   ├── efficientnetb0.onnx
│   ├── efficientnetV2.onnx
│   ├── mobilenetV3.onnx
│   ├── regnet1.6gf.onnx
│   ├── resnet50.onnx
│   ├── sample-cbr.onnx
│   ├── sample-reshape.onnx
│   └── vgg11.onnx
└── src
    ├── load_torchvision.py
    ├── sample_cbr.py
    └── sample_reshape.py
```

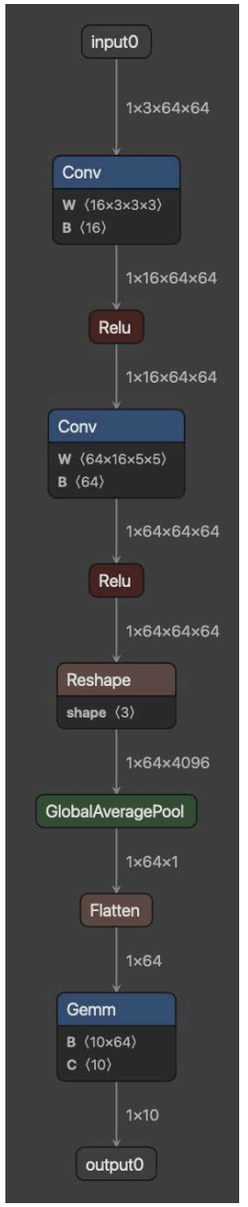
3.1-generate-onnx, 3.2-export-onnx

这两个我们今天一起看一下，熟悉导出onnx的步骤









执行结果的一部分展示

剖析onnx架构，理解ProtoBuf

Goal: 学习ONNX的Proto架构，使用onnx.helper创建onnx修改onnx

执行一下我们的python程序

```
3.3-read-and-parse-onnx/
├── models
│   ├── sample-cbr.onnx
│   ├── sample-convnet.onnx
│   └── sample-linear.onnx
└── src
    ├── create_onnx_convnet.py
    ├── create_onnx_linear.py
    ├── parse_onnx_cbr.py
    ├── parse_onnx_convnet.py
    ├── parse_onnx_linear.py
    └── parser.py
```

```
**************parse input/output*****************
Input info:
        name:       input0
        data Type: 1
        shape:      [1, 3, 64, 64]
Output info:
        name:       input0
        data Type: 1
        shape:      [1, 3, 64, 64]

**************parse node************************
node info:
        name:       conv2d_1
        op_type:    Conv
        inputs:     ['input0', 'conv2d_1.weight', 'conv2d_1.bias']
        outputs:    ['conv2d_1.output']
node info:
        name:       batchNorm1
        op_type:    BatchNormalization
        inputs:     ['conv2d_1.output', 'batchNorm1.scale', 'batchNorm1.bias', 'batchNorm1.mean', 'batchNorm1.var']
        outputs:    ['batchNorm1.output']
node info:
        name:       relu1
        op_type:    Relu
        inputs:     ['batchNorm1.output']
        outputs:    ['relu1.output']
node info:
        name:       avg_pool1
        op_type:    GlobalAveragePool
        inputs:     ['relu1.output']
        outputs:    ['avg_pool1.output']
node info:
        name:       conv2d_2
        op_type:    Conv
        inputs:     ['avg_pool1.output', 'conv2d_2.weight', 'conv2d_2.bias']
        outputs:    ['output0']
```

3.3-read-and-parse-onnx

一起学习onnx中ProtoBuf的框架，学习onnx的细节

执行结果的一部分展示

ONNX是什么?

(*)Protobuf: 全称叫做Protocal Buffer。是Google提出来的一套表示和序列化数据的机制

ONNX是一种神经网络的格式，采用Protobuf(*)二进制形式进行序列化模型。
Protobuf会根据用于定义的数据结构来进行序列化存储
同理，我们可以根据官方提供的数据结构信息，去修改或者创建onnx

1. Define proto

```
import "person_info.proto";

package persons;

message Person {
    PersonInfo info = 1; // characteristics of the person
    repeated Friend friends = 2; // friends of the person
}
```

2. Compile proto

```
_PERSON = _descriptor.Descriptor(
  name='Person',
  full_name='persons.Person',
  filename=None,
  file=DESCRIPTOR,
  containing_type=None,
  create_key=_descriptor._internal_create_key,
  fields=[
...
```

3. Serialize

```
info {
  age: 30
  height: 184
}
friends {
  friendship_duration: 365.1000061035156
  shared_hobbies: "books"
  shared_hobbies: "daydreaming"
  shared_hobbies: "unicorns"
  person {
    info {
      age: 40
      height: 165
    }
  }
}
```

Protobuf的编译以及序列化流程

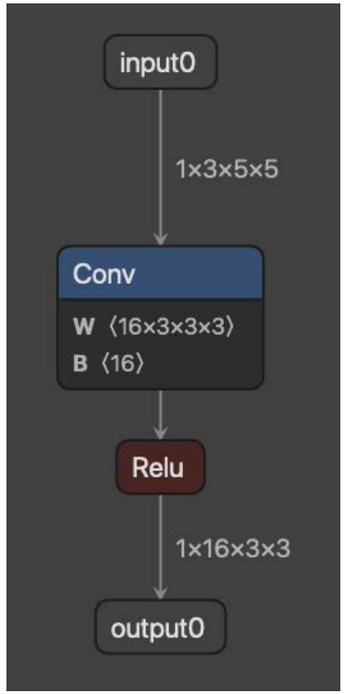
```
graph {
  node {
    input: "input0"
    input: "onnx::Conv_12"
    input: "onnx::Conv_13"
    output: "/conv1/Conv_output_0"
    name: "/conv1/Conv"
    op_type: "Conv"
    attribute {
      name: "dilations"
      type: INTS
      ints: 1
      ints: 1
    }
    attribute {
      name: "group"
      type: IN
      i: 1
    }
    attribute {
      name: "kernel_shape"
      type: INTS
      ints: 3
      ints: 3
    }
    attribute {
      name: "pads"
      type: INTS
      ints: 0
      ints: 0
      ints: 0
      ints: 0
    }
    attribute {
      name: "strides"
      type: INTS
      ints: 1
      ints: 1
    }
  }
```

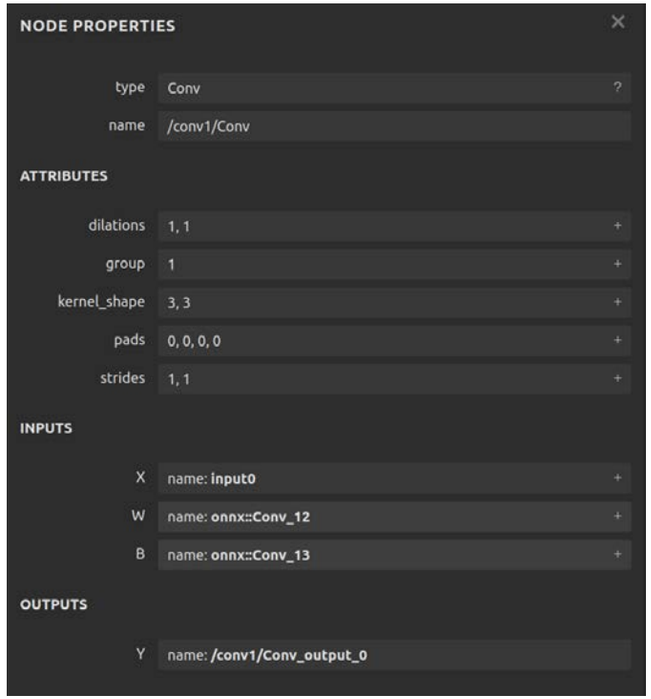
[1]The Essential Protobuf Guide for Python

onnx中的各类Proto

onnx的各类proto的定义需要看官方文档 (https://github.com/onnx/onnx/tree/main)
这里面的onnx/onnx.in.proto定义了所有onnx的Proto
有关onnx的IR(intermediate representation)信息，看这里(https://github.com/onnx/onnx/blob/main/docs/IR.md)

```
# 理解onnx中的组织结构
#   - ModelProto (描述的是整个模型的信息)
#   --- GraphProto (描述的是整个网络的信息)
#   ------ NodeProto (描述的是各个计算节点，比如conv, linear)
#   ------ TensorProto (描述的是tensor的信息，主要包括权重)
#   ------ ValueInfoProto (描述的是input/output信息)
```






```
// Graphs
//
// A graph defines the computational logic of a model and is comprised of a parame
// list of nodes that form a directed acyclic graph based on their inputs and outp
// This is the equivalent of the "network" or "graph" in many deep learning
// frameworks.
message GraphProto {
// The nodes in the graph, sorted topologically.
  repeated NodeProto node = 1;

  // The name of the graph.
  optional string name = 2;   // namespace Graph

  // A list of named tensor values, used to specify constant inputs of the graph.
  // Each initializer (both TensorProto as well SparseTensorProto) MUST have a nam
  // The name MUST be unique across both initializer and sparse_initializer,
  // but the name MAY also appear in the input list.
  repeated TensorProto initializer = 5;

  // Initializers (see above) stored in sparse format.
  repeated SparseTensorProto sparse_initializer = 15;

  // A human-readable documentation for this graph. Markdown is allowed.
  optional string doc_string = 10;

  // The inputs and outputs of the graph.
  repeated ValueInfoProto input = 11;
  repeated ValueInfoProto output = 12;

  // Information for the values in the graph. The ValueInfoProto.name's
  // must be distinct. It is optional for a value to appear in value_info list.
  repeated ValueInfoProto value_info = 13;

  // This field carries information to indicate the mapping among a tensor and its
  // quantization parameter tensors. For example:
  // For tensor 'a', it may have {'SCALE_TENSOR', 'a_scale'} and {'ZERO_POINT_TENS
  // which means, tensor 'a_scale' and tensor 'a_zero_point' are scale and zero po
  repeated TensorAnnotation quantization_annotation = 14;

  reserved 3, 4, 6 to 9;
  reserved "ir_version", "producer_version", "producer_tag", "domain";
}
```

理解onnx中的ValueInfoProto

```
// Defines information on value, including the name, the type, and
// the shape of the value.
message ValueInfoProto {
  // This field MUST be present in this version of the IR.
  optional string name = 1;     // namespace Value
  // This field MUST be present in this version of the IR for
  // inputs and outputs of the top-level graph.
  optional TypeProto type = 2;
  // A human-readable documentation for this value. Markdown is allowed.
  optional string doc_string = 3;
}
```

一般用来定义网络的input/output
(会根据input/output的type来附加属性)

```
// Types
//
// The standard ONNX data types.
message TypeProto {

  message Tensor {
    // This field MUST NOT have the value of UNDEFINED
    // This field MUST have a valid TensorProto.DataType value
    // This field MUST be present for this version of the IR.
    optional int32 elem_type = 1;
    optional TensorShapeProto shape = 2;
  }

  // repeated T
  message Sequence {
    // The type and optional shape of each element of the sequence.
    // This field MUST be present for this version of the IR.
    optional TypeProto elem_type = 1;
  };

  // map<K,V>
  message Map {
    // This field MUST have a valid TensorProto.DataType value
    // This field MUST be present for this version of the IR.
    // This field MUST refer to an integral type ([U]INT{8|16|32|64}) or STRING
    optional int32 key_type = 1;
    // This field MUST be present for this version of the IR.
    optional TypeProto value_type = 2;
  };

  // wrapper for Tensor, Sequence, or Map
  message Optional {
    // The type and optional shape of the element wrapped.
    // This field MUST be present for this version of the IR.
    // Possible values correspond to OptionalProto.DataType enum
    optional TypeProto elem_type = 1;
  };
```

理解onnx中的TensorProto

```
// Tensors
//
// A serialized tensor value.
message TensorProto {
  enum DataType {
    UNDEFINED = 0;
    // Basic types.
    FLOAT = 1;   // float
    UINT8 = 2;   // uint8_t
    INT8 = 3;    // int8_t
    UINT16 = 4;  // uint16_t
    INT16 = 5;   // int16_t
    INT32 = 6;   // int32_t
    INT64 = 7;   // int64_t
    STRING = 8;  // string
    BOOL = 9;    // bool

    // IEEE754 half-precision floating-point format (16 bits wide).
    // This format has 1 sign bit, 5 exponent bits, and 10 mantissa bits.
    FLOAT16 = 10;

    DOUBLE = 11;
    UINT32 = 12;
    UINT64 = 13;
    COMPLEX64 = 14;     // complex with float32 real and imaginary components
    COMPLEX128 = 15;    // complex with float64 real and imaginary components

    // Non-IEEE floating-point format based on IEEE754 single-precision
    // floating-point number truncated to 16 bits.
    // This format has 1 sign bit, 8 exponent bits, and 7 mantissa bits.
    BFLOAT16 = 16;

    // Non-IEEE floating-point format based on papers
    // FP8 Formats for Deep Learning, https://arxiv.org/abs/2209.05433,
    // 8-bit Numerical Formats For Deep Neural Networks, https://arxiv.org/pdf/2206.02
    // Operators supported FP8 are Cast, CastLike, QuantizeLinear, DequantizeLinear.
    // The computation usually happens inside a block quantize / dequantize
    // fused by the runtime.
    FLOAT8E4M3FN = 17;    // float 8, mostly used for coefficients, supports nan, not
    FLOAT8E4M3FNUZ = 18;  // float 8, mostly used for coefficients, supports nan, not
    FLOAT8E5M2 = 19;      // follows IEEE 754, supports nan, inf, mostly used for grad
    FLOAT8E5M2FNUZ = 20;  // follows IEEE 754, supports nan, inf, mostly used for grad

    // Future extensions go here.
  }
```

```
  }

  // The shape of the tensor.
  repeated int64 dims = 1;

  // The data type of the tensor.
  // This field MUST have a valid TensorProto.DataType value
  optional int32 data_type = 2;

  // For very large tensors, we may want to store them in chunks, in which
  // case the following fields will specify the segment that is stored in
  // the current TensorProto.
  message Segment {
    optional int64 begin = 1;
    optional int64 end = 2;
  }
  optional Segment segment = 3;

  // Tensor content must be organized in row-major order.
  //
  // Depending on the data_type field, exactly one of the fields below with
  // name ending in _data is used to store the elements of the tensor.

  // For float and complex64 values
  // Complex64 tensors are encoded as a single array of floats,
  // with the real components appearing in odd numbered positions,
  // and the corresponding imaginary component appearing in the
  // subsequent even numbered position. (e.g., [1.0 + 2.0i, 3.0 + 4.0i]
  // is encoded as [1.0, 2.0 ,3.0 ,4.0]
  // When this field is present, the data_type field MUST be FLOAT or COMPLEX64.
  repeated float float_data = 4 [packed = true];

  // For int32, uint8, int8, uint16, int16, bool, float8, and float16 values
  // float16 and float8 values must be bit-wise converted to an uint16_t prior
  // to writing to the buffer.
  // When this field is present, the data_type field MUST be
  // INT32, INT16, INT8, UINT16, UINT8, BOOL, FLOAT16, BFLOAT16, FLOAT8E4M3FN, FLOAT8E
  repeated int32 int32_data = 5 [packed = true];

  // For strings.
  // Each element of string_data is a UTF-8 encoded Unicode
  // string. No trailing null, no leading BOM. The protobuf "string"
  // scalar type is not used to match ML community conventions.
  // When this field is present, the data_type field MUST be STRING
  repeated bytes string_data = 6;
```

```
  // For int64.
  // When this field is present, the data_type field MUST be INT64
  repeated int64 int64_data = 7 [packed = true];

  // Optionally, a name for the tensor.
  optional string name = 8; // namespace Value

  // A human-readable documentation for this tensor. Markdown is allowed.
  optional string doc_string = 12;

  // Serializations can either use one of the fields above, or use this
  // raw bytes field. The only exception is the string case, where one is
  // required to store the content in the repeated bytes string_data field.
  //
  // When this raw_data field is used to store tensor value, elements MUST
  // be stored in as fixed-width, little-endian order.
  // Floating-point data types MUST be stored in IEEE 754 format.
  // Complex64 elements must be written as two consecutive FLOAT values, real componen
  // Complex128 elements must be written as two consecutive DOUBLE values, real compon
  // Boolean type MUST be written one byte per tensor element (00000001 for true, 0000
  //
  // Note: the advantage of specific field rather than the raw_data field is
  // that in some cases (e.g. int data), protobuf does a better packing via
  // variable length storage, and may lead to smaller binary footprint.
  // When this field is present, the data_type field MUST NOT be STRING or UNDEFINED
  optional bytes raw_data = 9;

  // Data can be stored inside the protobuf file using type-specific fields or raw_dat
  // Alternatively, raw bytes data can be stored in an external file, using the extern
  // external_data stores key-value pairs describing data location. Recognized keys ar
  // - "location" (required) - POSIX filesystem path relative to the directory where t
  //                           protobuf model was stored
  // - "offset" (optional) - position of byte at which stored data begins. Integer sto
  //                         Offset values SHOULD be multiples 4096 (page size) to ena
  // - "length" (optional) - number of bytes containing data. Integer stored as string
  // - "checksum" (optional) - SHA1 digest of file specified in under 'location' key.
  repeated StringStringEntryProto external_data = 13;

  // Location of the data for this tensor. MUST be one of:
  // - DEFAULT - data stored inside the protobuf message. Data is stored in raw_data (
  // - EXTERNAL - data stored in an external location as described by external_data fi
  enum DataLocation {
    DEFAULT = 0;
    EXTERNAL = 1;
  }
```

一般用来定义一个权重，比如conv的w和b

(dims是repeated类型，意味着是数组)

(raw_data是bytes类型)

理解onnx中的NodeProto

```proto
// Nodes
//
// Computation graphs are made up of a DAG of nodes, which represent what is
// commonly called a "layer" or "pipeline stage" in machine learning frameworks.
//
// For example, it can be a node of type "Conv" that takes in an image, a filter
// tensor and a bias tensor, and produces the convolved output.
message NodeProto {
  repeated string input = 1;    // namespace Value
  repeated string output = 2;   // namespace Value

  // An optional identifier for this node in a graph.
  // This field MAY be absent in ths version of the IR.
  optional string name = 3;     // namespace Node

  // The symbolic identifier of the Operator to execute.
  optional string op_type = 4;  // namespace Operator
  // The domain of the OperatorSet that specifies the operator named by op_type.
  optional string domain = 7;   // namespace Domain

  // Additional named attributes.
  repeated AttributeProto attribute = 5;

  // A human-readable documentation for this node. Markdown is allowed.
  optional string doc_string = 6;
}
```

一般用来定义一个计算节点，比如conv, linear
(input是repeated类型，意味着是数组)
(output是repeated类型，意味着是数组)
(attribute有一个自己的Proto)
(op_type需要严格根据onnx所提供的Operators写)

RandomUniform	1	
RandomUniformLike	1	
Reciprocal	13, 6, 1	
ReduceMax	18, 13, 12, 11, 1	
ReduceMean	18, 13, 11, 1	
ReduceMin	18, 13, 12, 11, 1	
ReduceProd	18, 13, 11, 1	
ReduceSum	13, 11, 1	
Reshape	19, 14, 13, 5, 1	
Resize	19, 18, 13, 11, 10	
ReverseSequence	10	
RoiAlign	16, 10	
Round	11	
STFT	17	
Scan	19, 16, 11, 9, 8	
Scatter (deprecated)	11, 9	
ScatterElements	18, 16, 13, 11	
ScatterND	18, 16, 13, 11	

https://github.com/onnx/onnx/blob/main/docs/Operators.md

理解onnx中的AttributeProto

```
// Attributes
//
// A named attribute containing either singular float, integer, string, graph,
// and tensor values, or repeated float, integer, string, graph, and tensor values.
// An AttributeProto MUST contain the name field, and *only one* of the
// following content fields, effectively enforcing a C/C++ union equivalent.
message AttributeProto {

  // Note: this enum is structurally identical to the OpSchema::AttrType
  // enum defined in schema.h.  If you rev one, you likely need to rev the other.
  enum AttributeType {
    UNDEFINED = 0;
    FLOAT = 1;
    INT = 2;
    STRING = 3;
    TENSOR = 4;
    GRAPH = 5;
    SPARSE_TENSOR = 11;
    TYPE_PROTO = 13;

    FLOATS = 6;
    INTS = 7;
    STRINGS = 8;
    TENSORS = 9;
    GRAPHS = 10;
    SPARSE_TENSORS = 12;
    TYPE_PROTOS = 14;
  }

  // The name field MUST be present for this version of the IR.
  optional string name = 1;           // namespace Attribute

  // if ref_attr_name is not empty, ref_attr_name is the attribute name in parent func
  // In this case, this AttributeProto does not contain data, and it's a reference of
  // in parent scope.
  // NOTE: This should ONLY be used in function (sub-graph). It's invalid to be used i
  optional string ref_attr_name = 21;

  // A human-readable documentation for this attribute. Markdown is allowed.
  optional string doc_string = 13;

  // The type field MUST be present for this version of the IR.
  // For 0.0.1 versions of the IR, this field was not defined, and
  // implementations needed to use has_field heuristics to determine
  // which value field was in use.  For IR_VERSION 0.0.2 or later, this
```

```
  // field MUST be set and match the f|i|s|t|... field in use.  This
  // change was made to accommodate proto3 implementations.
  optional AttributeType type = 20;   // discriminator that indicates which field belo

  // Exactly ONE of the following fields must be present for this version of the IR
  optional float f = 2;               // float
  optional int64 i = 3;               // int
  optional bytes s = 4;               // UTF-8 string
  optional TensorProto t = 5;         // tensor value
  optional GraphProto g = 6;          // graph
  optional SparseTensorProto sparse_tensor = 22;  // sparse tensor value
  // Do not use field below, it's deprecated.
  // optional ValueProto v = 12;         // value - subsumes everything but graph
  optional TypeProto tp = 14;          // type proto

  repeated float floats = 7;          // list of floats
  repeated int64 ints = 8;            // list of ints
  repeated bytes strings = 9;         // list of UTF-8 strings
  repeated TensorProto tensors = 10;  // list of tensors
  repeated GraphProto graphs = 11;    // list of graph
  repeated SparseTensorProto sparse_tensors = 23; // list of sparse tensors
  repeated TypeProto type_protos = 15;// list of type protos
}
```

一般用来定义一个node的属性。比如说kernel size

(

比较常见的方式就是把(key, value)传入Proto，之后

name = key

i = value

)

理解onnx中的GraphProto

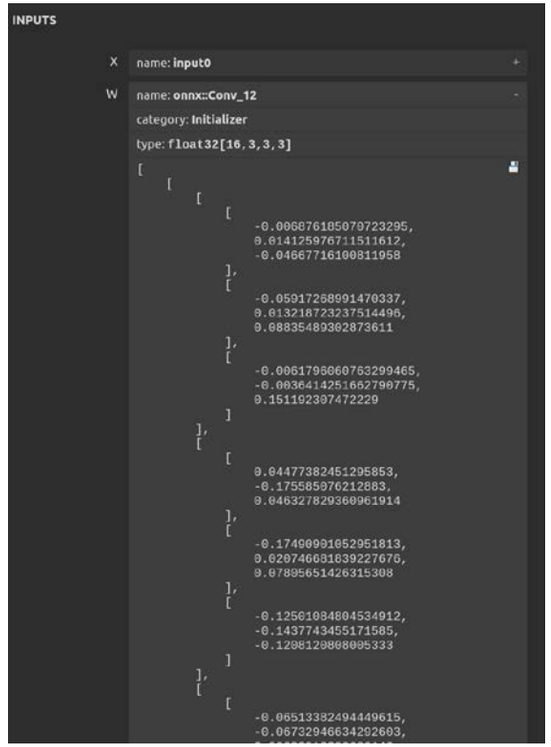
(*)initializer: 在onnx中一般表示权重信息。我们可以在netron看到

```
435
436 // Graphs
437 //
438 // A graph defines the computational logic of a model and is comprised of a parame
439 // list of nodes that form a directed acyclic graph based on their inputs and outp
440 // This is the equivalent of the "network" or "graph" in many deep learning
441 // frameworks.
442 message GraphProto {
443   // The nodes in the graph, sorted topologically.
444   repeated NodeProto node = 1;
445
446   // The name of the graph.
447   optional string name = 2;   // namespace Graph
448
449   // A list of named tensor values, used to specify constant inputs of the graph.
450   // Each initializer (both TensorProto as well SparseTensorProto) MUST have a nam
451   // The name MUST be unique across both initializer and sparse_initializer,
452   // but the name MAY also appear in the input list.
453   repeated TensorProto initializer = 5;
454
455   // Initializers (see above) stored in sparse format.
456   repeated SparseTensorProto sparse_initializer = 15;
457
458   // A human-readable documentation for this graph. Markdown is allowed.
459   optional string doc_string = 10;
460
461   // The inputs and outputs of the graph.
462   repeated ValueInfoProto input = 11;
463   repeated ValueInfoProto output = 12;
464
465   // Information for the values in the graph. The ValueInfoProto.name's
466   // must be distinct. It is optional for a value to appear in value_info list.
467   repeated ValueInfoProto value_info = 13;
468
469   // This field carries information to indicate the mapping among a tensor and its
470   // quantization parameter tensors. For example:
471   // For tensor 'a', it may have {'SCALE_TENSOR', 'a_scale'} and {'ZERO_POINT_TENS
472   // which means, tensor 'a_scale' and tensor 'a_zero_point' are scale and zero po
473   repeated TensorAnnotation quantization_annotation = 14;
474
475   reserved 3, 4, 6 to 9;
476   reserved "ir_version", "producer_version", "producer_tag", "domain";
477 }
```

一般用来定义一个网络。包括

- input/output
- initializer(*)
- node

(node是repeated，所以是数组)
(initializer是repeated，所以是数组)
(input/output是repeated，所以是数组)



理解onnx中的ModelProto

```
337 // Models
338 //
339 // ModelProto is a top-level file/container format for bundling a ML model and
340 // associating its computation graph with metadata.
341 //
342 // The semantics of the model are described by the associated GraphProto's.
343 message ModelProto {
344   // The version of the IR this model targets. See Version enum above.
345   // This field MUST be present.
346   optional int64 ir_version = 1;
347
348   // The OperatorSets this model relies on.
349   // All ModelProtos MUST have at least one entry that
350   // specifies which version of the ONNX OperatorSet is
351   // being imported.
352   //
353   // All nodes in the ModelProto's graph will bind against the operator
354   // with the same-domain/same-op_type operator with the HIGHEST version
355   // in the referenced operator sets.
356   repeated OperatorSetIdProto opset_import = 8;
357
358   // The name of the framework or tool used to generate this model.
359   // This field SHOULD be present to indicate which implementation/tool/framework
360   // emitted the model.
361   optional string producer_name = 2;
362
363   // The version of the framework or tool used to generate this model.
364   // This field SHOULD be present to indicate which implementation/tool/framework
365   // emitted the model.
366   optional string producer_version = 3;
367
368   // Domain name of the model.
369   // We use reverse domain names as name space indicators. For example:
370   // `com.facebook.fair` or `com.microsoft.cognitiveservices`
371   //
372   // Together with `model_version` and GraphProto.name, this forms the unique identity
373   // the graph.
374   optional string domain = 4;
375
376   // The version of the graph encoded. See Version enum below.
377   optional int64 model_version = 5;
378
379   // A human-readable documentation for this model. Markdown is allowed.
380   optional string doc_string = 6;
381
382   // The parameterized graph that is evaluated to execute the model.
```

```
383   optional GraphProto graph = 7;
384
385   // Named metadata values; keys should be distinct.
386   repeated StringStringEntryProto metadata_props = 14;
387
388   // Training-specific information. Sequentially executing all stored
389   // `TrainingInfoProto.algorithm`s and assigning their outputs following
390   // the corresponding `TrainingInfoProto.update_binding`s is one training
391   // iteration. Similarly, to initialize the model
392   // (as if training hasn't happened), the user should sequentially execute
393   // all stored `TrainingInfoProto.initialization`s and assigns their outputs
394   // using `TrainingInfoProto.initialization_binding`s.
395   //
396   // If this field is empty, the training behavior of the model is undefined.
397   repeated TrainingInfoProto training_info = 20;
398
399   // A list of function protos local to the model.
400   //
401   // Name of the function "FunctionProto.name" should be unique within the domain "Fun
402   // In case of any conflicts the behavior (whether the model local functions are give
403   // or standard opserator sets are given higher priotity or this is treated as error
404   // the runtimes.
405   //
406   // The operator sets imported by FunctionProto should be compatible with the ones
407   // imported by ModelProto and other model local FunctionProtos.
408   // Example, if same operator set say 'A' is imported by a FunctionProto and ModelPro
409   // or by 2 FunctionProtos then versions for the operator set may be different but,
410   // the operator schema returned for op_type, domain, version combination
411   // for both the versions should be same for every node in the function body.
412   //
413   // One FunctionProto can reference other FunctionProto in the model, however, recurs
414   // is not allowed.
415   repeated FunctionProto functions = 25;
416 };
417
```

一般用来定义模型的全局信息，比如opset

(graph并不是repeated，所以一个model对应一个graph)

根据onnx中的Proto信息，创建onnx

onnx提供了一些很方便的api来创建onnx

onnx.helper.make_tensor
onnx.helper.make_tensor_value_info
onnx.helper.make_attribute
onnx.helper.make_node
onnx.helper.make_graph
onnx.helper.make_model

```
############################创建第一个conv节点################################
conv1_output_name = "conv2d_1.output"
conv1_in_ch       = input_channel
conv1_out_ch      = 32
conv1_kernel      = 3
conv1_pads        = 1

# 创建conv节点的权重信息
conv1_weight    = np.random.rand(conv1_out_ch, conv1_in_ch, conv1_kernel, conv1_kernel)
conv1_bias      = np.random.rand(conv1_out_ch)

conv1_weight_name = "conv2d_1.weight"
conv1_weight_initializer = create_initializer_tensor(
    name         = conv1_weight_name,
    tensor_array = conv1_weight,
    data_type    = onnx.TensorProto.FLOAT)


conv1_bias_name  = "conv2d_1.bias"
conv1_bias_initializer = create_initializer_tensor(
    name         = conv1_bias_name,
    tensor_array = conv1_bias,
    data_type    = onnx.TensorProto.FLOAT)

# 创建conv节点，注意conv节点的输入有3个：input, w, b
conv1_node = onnx.helper.make_node(
    name         = "conv2d_1",
    op_type      = "Conv",
    inputs       = [
        model_input_name,
        conv1_weight_name,
        conv1_bias_name
    ],
    outputs      = [conv1_output_name],
    kernel_shape = [conv1_kernel, conv1_kernel],
    pads         = [conv1_pads, conv1_pads, conv1_pads, conv1_pads],
)
```

根据onnx中的Proto信息，读取onnx

只要知道Proto的数据结构，我们就可以parse整个onnx，或者获取特定点的信息

```python
print("\n**************parse input/output****************")
for input in inputs:
    input_shape = []
    for d in input.type.tensor_type.shape.dim:
        if d.dim_value == 0:
            input_shape.append(None)
        else:
            input_shape.append(d.dim_value)
    print("Input info: \
            \n\tname:      {} \
            \n\tdata Type: {} \
            \n\tshape:     {}".format(input.name, input.type.tensor_type.elem_type, input_shape))

for output in outputs:
    output_shape = []
    for d in output.type.tensor_type.shape.dim:
        if d.dim_value == 0:
            output_shape.append(None)
        else:
            output_shape.append(d.dim_value)
    print("Output info: \
            \n\tname:      {} \
            \n\tdata Type: {} \
            \n\tshape:     {}".format(input.name, output.type.tensor_type.elem_type, input_shape))

print("\n**************parse node**********************")
for node in nodes:
    print("node info: \
            \n\tname:      {} \
            \n\top_type:   {} \
            \n\tinputs:    {} \
            \n\toutputs:   {}".format(node.name, node.op_type, node.input, node.output))

print("\n**************parse initializer***************")
for initializer in initializers:
    print("initializer info: \
            \n\tname:      {} \
            \n\tdata_type: {} \
            \n\tshape:     {}".format(initializer.name, initializer.data_type, initializer.dims))
```

```
**************parse input/output****************
Input info:
        name:      input0
        data Type: 1
        shape:     [1, 3, 64, 64]
Output info:
        name:      input0
        data Type: 1
        shape:     [1, 3, 64, 64]

**************parse node**********************
node info:
        name:      conv2d_1
        op_type:   Conv
        inputs:    ['input0', 'conv2d_1.weight', 'conv2d_1.bias']
        outputs:   ['conv2d_1.output']
node info:
        name:      batchNorm1
        op_type:   BatchNormalization
        inputs:    ['conv2d_1.output', 'batchNorm1.scale', 'batchNorm1.bias',
        outputs:   ['batchNorm1.output']
node info:
        name:      relu1
        op_type:   Relu
        inputs:    ['batchNorm1.output']
        outputs:   ['relu1.output']
node info:
        name:      avg_pool1
        op_type:   GlobalAveragePool
        inputs:    ['relu1.output']
        outputs:   ['avg_pool1.output']
node info:
        name:      conv2d_2
        op_type:   Conv
        inputs:    ['avg_pool1.output', 'conv2d_2.weight', 'conv2d_2.bias']
        outputs:   ['output0']

**************parse initializer***************
```

根据onnx中的Proto信息，修改onnx

虽然onnx官方提供了而一些pythonAPI来修改onnx，但是我这里推荐大家使用TensorRT下的onnxsurgeon
详细请看后面的小节

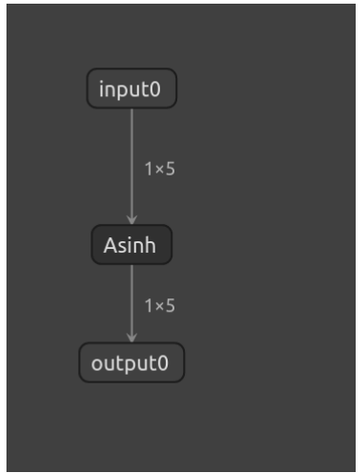
onnx注册算子的方法

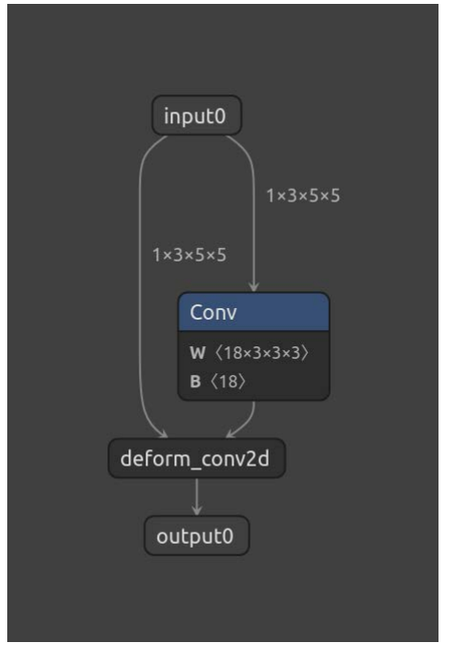
Goal: 学习pytorch导出onnx不成功的时候如何解决(without plugin篇)

执行一下我们的python程序

```
../3.4-export-unsupported-node/
├── models
│   ├── sample-asinh.onnx
│   ├── sample-deformable-conv.onnx
│   ├── swin-tiny-after-simplify.onnx
│   └── swin-tiny-before-simplify.onnx
└── src
    ├── sample_asinh.py
    ├── sample_asinh_register.py
    ├── sample_deformable_conv.py
    └── sample_deformable_conv_register.py
```

3.4-export-unsupported-node
一起学习onnx中ProtoBuf的框架，学习onnx的细节






```
============= Diagnostic Run torch.onnx.export version 2.0.1+cu117 =============
verbose: False, log level: Level.ERROR
======================== 0 NONE 0 NOTE 0 WARNING 0 ERROR ========================

Finished normal onnx export
result from Pytorch is : tensor([[0.6005, 0.6733, 0.5829, 0.2650, 0.4782]])
result from onnx is:     [array([[0.6005171 , 0.67329633, 0.5829002 , 0.26500833, 0.47822666]],
      dtype=float32)]
```

执行结果的一部分展示

转换swin-tiny时候出现的不兼容op的例子

```
  symbolic_fn = _find_symbolic_in_registry(domain, op_name, opset_version, operator_export_type)
  File "/home/kalfazed/miniconda3/envs/swin/lib/python3.7/site-packages/torch/onnx/utils.py", line 947, in _find_symbolic_in_registry
    return sym_registry.get_registered_op(op_name, domain, opset_version)
  File "/home/kalfazed/miniconda3/envs/swin/lib/python3.7/site-packages/torch/onnx/symbolic_registry.py", line 112, in get_registered_op
    raise RuntimeError(msg)
RuntimeError: Exporting the operator roll to ONNX opset version 9 is not supported. Please feel free to request support or submit a pull request o
PyTorch GitHub.
```

先使用opset9看看是否可以导出

```
  File "/home/kalfazed/miniconda3/envs/swin/lib/python3.7/site-packages/torch/onnx/utils.py", line 947, in _find_symbolic_
    return sym_registry.get_registered_op(op_name, domain, opset_version)
  File "/home/kalfazed/miniconda3/envs/swin/lib/python3.7/site-packages/torch/onnx/symbolic_registry.py", line 112, in get
    raise RuntimeError(msg)
RuntimeError: Exporting the operator roll to ONNX opset version 12 is not supported. Please feel free to request support o
```

使用opset12依然会出现问题

```python
# reverse cyclic shift
if self.shift_size > 0:
    if not self.fused_window_process:
        shifted_x = window_reverse(attn_windows, self.window_size, H, W)  # B H' W' C
        x = torch.roll(shifted_x, shifts=(self.shift_size, self.shift_size), dims=(1, 2))
    else:
        x = WindowProcessReverse.apply(attn_windows, B, H, W, C, self.shift_size, self.window_size)
else:
    shifted_x = window_reverse(attn_windows, self.window_size, H, W)  # B H' W' C
    x = shifted_x
x = x.view(B, H * W, C)
x = shortcut + self.drop_path(x)
```

swin/models/swin_transformer.py

当出现导出onnx不成功的时候，我们需要考虑的事情

(*)注意：因为onnx是一种图结构表示，并不包含各个算子的实现。除非我们是要在onnx-runtime上测试，否则我们更看重onnx-trt中这个算子的支持情况

难易度从第到高：

- 修改opset的版本
  - 查看不支持的算子在新的opset中是否被支持
  - 如果不考虑自己搭建plugin的话，也需要看看onnx-trt中这个算子是否被支持(*)
  - 大家习惯看官方文档
- 替换pytorch中的算子组合
  - 把某些计算替换成onnx可以识别的
- 在pytorch登记onnx中某些算子
  - 有可能onnx中有支持，但没有被登记
- 直接修改onnx，创建plugin
  - 使用onnx-surgeon
  - 一般是用在加速某些算子上使用

unsupported asinh算子

```python
class Model(torch.nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, x):
        x = torch.asinh(x)
        return x
```

```
  File "/home/kalfazed/miniconda3/envs/trt-starter/lib/python3.9/site-packages/torch/onnx/utils.py", line 1117, in _model_to
    graph = _optimize_graph(
  File "/home/kalfazed/miniconda3/envs/trt-starter/lib/python3.9/site-packages/torch/onnx/utils.py", line 665, in _optimize_
    graph = _C._jit_pass_onnx(graph, operator_export_type)
  File "/home/kalfazed/miniconda3/envs/trt-starter/lib/python3.9/site-packages/torch/onnx/utils.py", line 1901, in _run_symb
    raise errors.UnsupportedOperatorError(
torch.onnx.errors.UnsupportedOperatorError: Exporting the operator 'aten::asinh' to ONNX opset version 12 is not supported.
ll request on PyTorch GitHub: https://github.com/pytorch/pytorch/issues.
[station]~/C/d/i/t/c/3/src (main|+555) [1]$ ll
```

src/sample_asinh.py

(出现导出问题，我们先去寻找官方文档)

unsupported asinh算子

(*)opset: operator sets。可以理解为每个onnx版本所支持的算子集合。一般来说越大越多

Acosh	9	
Add	14, 13, 7, 6, 1	
And	7, 1	
ArgMax	13, 12, 11, 1	
ArgMin	13, 12, 11, 1	
Asin	7	
Asinh	9	
Atan	7	
Atanh	9	
AveragePool	19, 11, 10, 7, 1	
BatchNormalization	15, 14, 9, 7, 6, 1	
BitShift	11	
BitwiseAnd	18	

https://github.com/onnx/onnx/blob/main/docs/Operators.md

这里我们可以看到asinh算子在opset9(*)这个版本已经开始支持了。
那我们换一下opset试一下

```python
def export_norm_onnx():
    input   = torch.rand(1, 5)
    model   = Model()
    model.eval()

    file    = "../models/sample-asinh.onnx"
    torch.onnx.export(
        model         = model,
        args          = (input,),
        f             = file,
        input_names   = ["input0"],
        output_names  = ["output0"],
        opset_version = 9)
    print("Finished normal onnx export")
```

```
    graph = _C._jit_pass_onnx(graph, operator_export_type)
  File "/home/kalfazed/miniconda3/envs/trt-starter/lib/python3.9/site-packages/torch/onnx/utils.py", line 1901, in _run_sym
    raise errors.UnsupportedOperatorError(
torch.onnx.errors.UnsupportedOperatorError: Exporting the operator 'aten::asinh' to ONNX opset version 9 is not supported.
l request on PyTorch GitHub: https://github.com/pytorch/pytorch/issues.
```

unsupported asinh算子

(*)opset: operator sets。可以理解为每个onnx版本所支持的算子集合。一般来说越大越多

```
- 3.2k 22  6月  01:10  __init__.py
x    - 22  6月  01:10  __pycache__
-  411 22  6月  01:10  _constants.py
- 2.0k 22  6月  01:10  _deprecation.py
- 1.0k 22  6月  01:10  _experimental.py
- 1.7k 22  6月  01:10  _exporter_states.py
- 3.2k 22  6月  01:10  _globals.py
x    - 22  6月  01:10  _internal
- 3.3k 22  6月  01:10  _onnx_supported_ops.py
-  12k 22  6月  01:10  _type_utils.py
- 3.9k 22  6月  01:10  errors.py
-  560 22  6月  01:10  operators.py
-  11k 22  6月  01:10  symbolic_caffe2.py
-  62k 22  6月  01:10  symbolic_helper.py
- 2.1k 22  6月  01:10  symbolic_opset7.py
-  15k 22  6月  01:10  symbolic_opset8.py
- 231k 22  6月  01:10  symbolic_opset9.py
-  29k 22  6月  01:10  symbolic_opset10.py
-  56k 22  6月  01:10  symbolic_opset11.py
-  16k 22  6月  01:10  symbolic_opset12.py
-  31k 22  6月  01:10  symbolic_opset13.py
- 3.4k 22  6月  01:10  symbolic_opset14.py
- 2.9k 22  6月  01:10  symbolic_opset15.py
- 3.9k 22  6月  01:10  symbolic_opset16.py
- 1.4k 22  6月  01:10  symbolic_opset17.py
- 1.7k 22  6月  01:10  symbolic_opset18.py
-  80k 22  6月  01:10  utils.py
-  70k 22  6月  01:10  verification.py
~/m/e/t/l/p/s/t/onnx $ pwd
fazed/miniconda3/envs/trt-starter/lib/python3.9/site-packages/torch/onnx
```

```
    "addcmul",
    "addmm",
    "alias",
    "amax",
    "amin",
    "aminmax",
    "arange",
    "argmax",
    "argmin",
    "as_strided",
    "as_tensor",
    "asin",
    "atan",
    "baddbmm",
    "batch_norm",
    "bernoulli",
    "bitwise_not",
    "bitwise_or",
    "bmm",
    "broadcast_tensors",
    "bucketize",
    "cat",
    "cdist",
    "ceil"
```

torch/onnx/symbolic_opset9.py
(很遗憾，虽然onnx支持asinh,但是pytorch2onnx没有建立起桥梁)

unsupported asinh算子

```python
@_onnx_symbolic("aten::_shape_as_tensor")
@_beartype.beartype
def _shape_as_tensor(g: jit_utils.GraphContext, input):
    return g.op("Shape", input)


@_onnx_symbolic("aten::_reshape_from_tensor")
@_beartype.beartype
def _reshape_from_tensor(g: jit_utils.GraphContext, input, shape):
    if isinstance(shape, list):
        shape = g.op("Concat", *shape, axis_i=0)
    return reshape(g, input, shape)


@_onnx_symbolic("aten::reshape")
@symbolic_helper.quantized_args(True)
@_beartype.beartype
def reshape(g: jit_utils.GraphContext, self, shape):
    return symbolic_helper._reshape_helper(g, self, shape)


@_onnx_symbolic("aten::reshape_as")
@symbolic_helper.quantized_args(True)
@_beartype.beartype
def reshape_as(g: jit_utils.GraphContext, self, other):
    shape = g.op("Shape", other)
    return reshape(g, self, shape)
```

torch/onnx/symbolic_opset9.py
但我们可以从这里面知道pytorch中的各个op是如何登记的(register)

aten::xxx

- c++的一个namespace，pytorch的很多算子的底层都是在aten这个命名空间下进行以c++进行实现的

onnx_symblic

- 负责绑定
- 绑定pytorch中的算子与aten命名空间下的算子的一一对应

unsupported asinh算子

```python

# 创建一个asinh算子的symblic，符号函数，用来登记
# 符号函数内部调用g.op，为onnx计算图添加Asinh算子
#   g: 就是graph，计算图
#   也就是说，在计算图中添加onnx算子
#   由于我们已经知道Asinh在onnx是有实现的，所以我们只要在g.op调用这个op的名字就好了
#   symblic的参数需要与Pytorch的asinh接口函数的参数对齐
#       def asinh(input: Tensor, *, out: Optional[Tensor]=None) -> Tensor: ...
def asinh_symbolic(g, input, *, out=None):
    return g.op("Asinh", input)

# 在这里，将asinh_symbolic这个符号函数，与PyTorch的asinh算子绑定。也就是所谓的“注册算子”
# asinh是在名为aten的一个c++命名空间下进行实现的
register_custom_op_symbolic('aten::asinh', asinh_symbolic, 12)


# 这里容易混淆的地方：
# 1. register_op中的第一个参数是PyTorch中的算子名字：aten::asinh
# 2. g.op中的第一个参数是onnx中的算子名字：Asinh

```

那么我们就可以按照类似的方法去自己创建这个联系

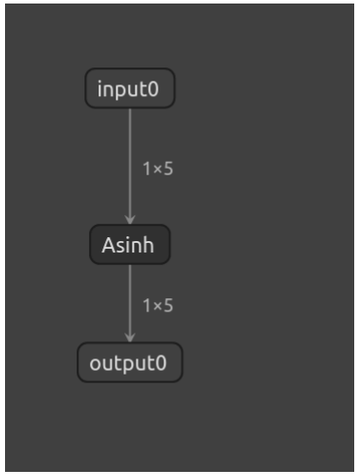
- 创建symbolic符号函数，从而创建一个onnx operator
- 注册这个onnx operator，并让它与底层的aten中的asinh实现绑定

(注意：symbolic符号函数中的参数，需要严格遵循pytorch中的定义)

```python
def asin_(input: Tensor) -> Tensor: ...
def asinh(input: Tensor, *, out: Optional[Tensor]=None) -> Tensor: ...
```

unsupported asinh算子

```
[station]~/C/d/i/t/c/3/src (main|+555) $ python sample_asinh_register.py
============= Diagnostic Run torch.onnx.export version 2.0.1+cu117 =============
verbose: False, log level: Level.ERROR
======================= 0 NONE 0 NOTE 0 WARNING 0 ERROR ========================

Finished normal onnx export
result from Pytorch is : tensor([[0.3245, 0.1989, 0.4941, 0.5102, 0.2483]])
result from onnx is:     [array([[0.3245445 , 0.19885206, 0.4941113 , 0.51015097, 0.2483138 ]],
      dtype=float32)]
```



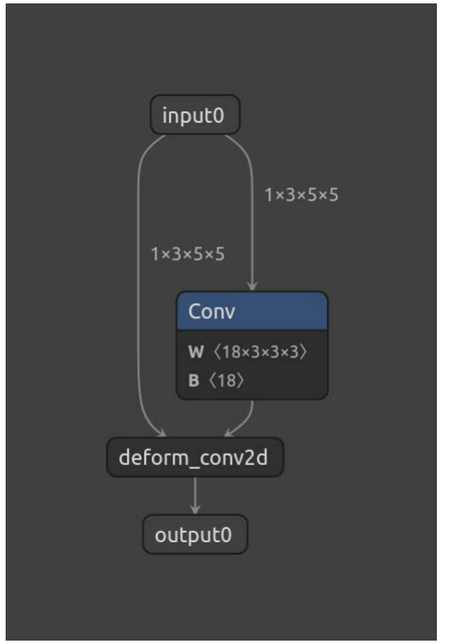
成功导出asinh，并可以使用onnxruntime进行验证

```python
def validate_onnx():
    input = torch.rand(1, 5)

    # PyTorch的推理
    model = Model()
    x     = model(input)
    print("result from Pytorch is :", x)

    # onnxruntime的推理
    sess  = onnxruntime.InferenceSession('../models/sample-asinh.onnx')
    x     = sess.run(None, {'input0': input.numpy()})
    print("result from onnx is:    ", x)
```

onnxruntime的验证。很简单

unsupported deformable conv算子

```python

# 注意
#   这里需要把args的各个参数的类型都指定
#   这里还没有实现底层对deform_conv2d的实现
#   具体dcn的底层实现是在c++完成的，这里会在后面的TensorRT plugin中回到这里继续讲这个案例
#   这里先知道对于不支持的算子，onnx如何导出即可
@parse_args("v", "v", "v", "v", "v", "i", "i", "i", "i", "i","i", "i", "i", "none")
def dcn_symbolic(
        g,
        input,
        weight,
        offset,
        mask,
        bias,
        stride_h, stride_w,
        pad_h, pad_w,
        dil_h, dil_w,
        n_weight_grps,
        n_offset_grps,
        use_mask):
    return g.op("custom::deform_conv2d", input, offset)

register_custom_op_symbolic("torchvision::deform_conv2d", dcn_symbolic, 12)


class Model(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 18, 3)
        self.conv2 = torchvision.ops.DeformConv2d(3, 3, 3)

    def forward(self, x):
        x = self.conv2(x, self.conv1(x))
        return x
```



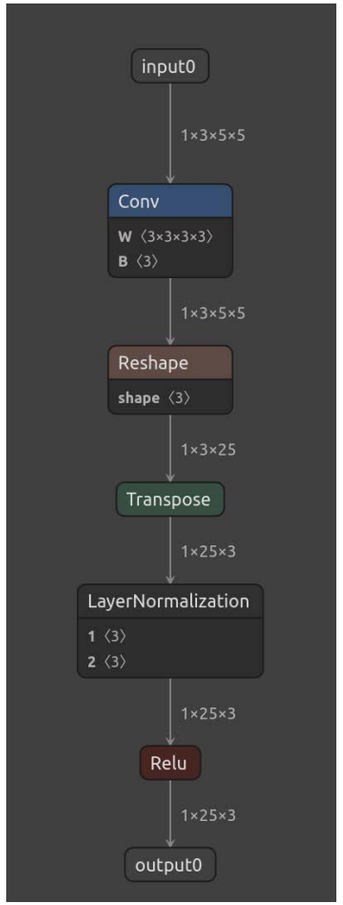
onnx-graph-surgeon

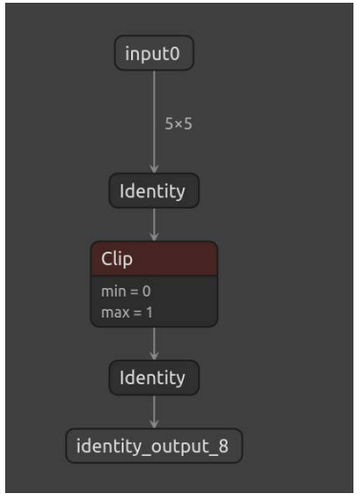
Goal: 学习使用onnx-surgeon，比较与onnx.helper的区别，学习快速修改onnx以及替换算子/创建算子的技巧

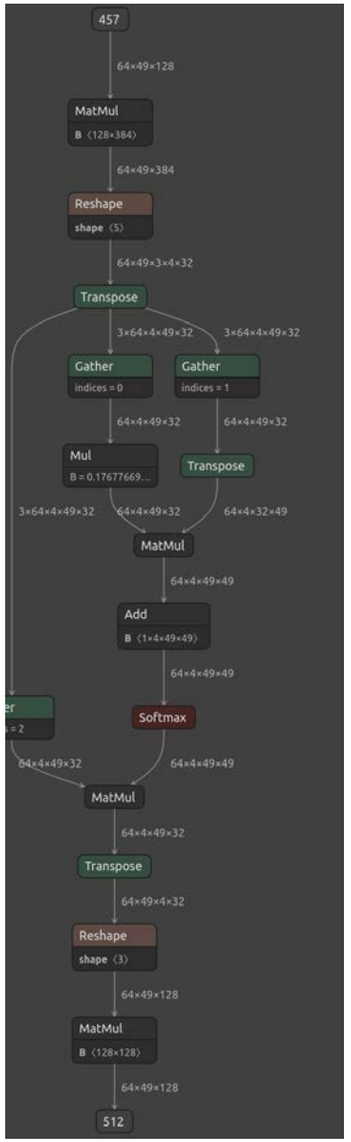
执行一下我们的python程序

```
3.5-onnxsurgeon/
├── models
│   ├── sample-complicated-graph.onnx
│   ├── sample-conv.onnx
│   ├── sample-ln-after.onnx
│   ├── sample-ln-before.onnx
│   ├── sample-minmax.onnx
│   ├── sample-minmax-to-clip.onnx
│   ├── swin-subgraph-LN.onnx
│   ├── swin-subgraph-MSHA.onnx
│   └── swin-tiny.onnx
└── src
    ├── gs_create_complicate_graph.py
    ├── gs_create_conv.py
    ├── gs_create_subgraph.py
    ├── gs_replace_LN.py
    └── gs_replace_op.py
```

3.5-onnxsurgeon
学习使用onnxsurgeon来创建/修改onnx





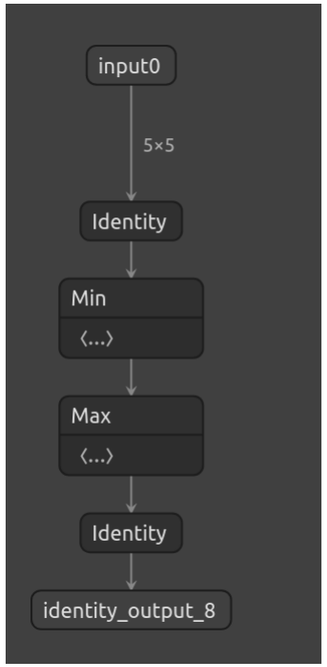


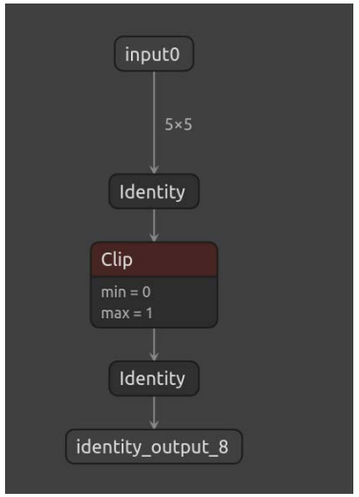
执行结果的一部分展示

onnx-graph-surgeon

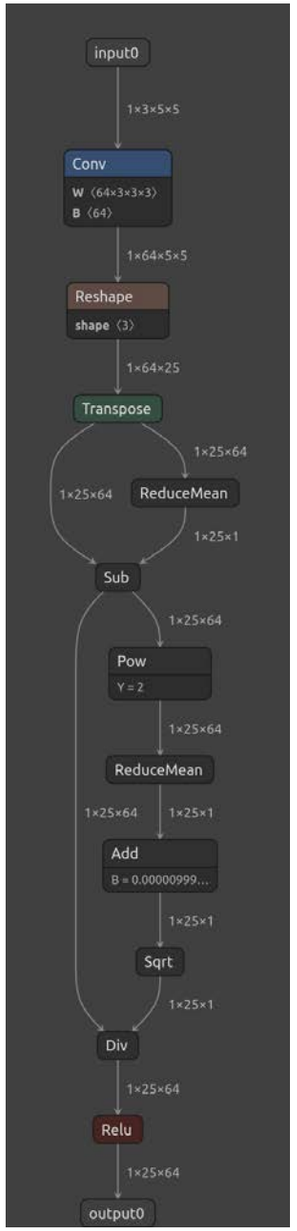
创建/修改onnx的工具。在TensorRT/tools中可以安装

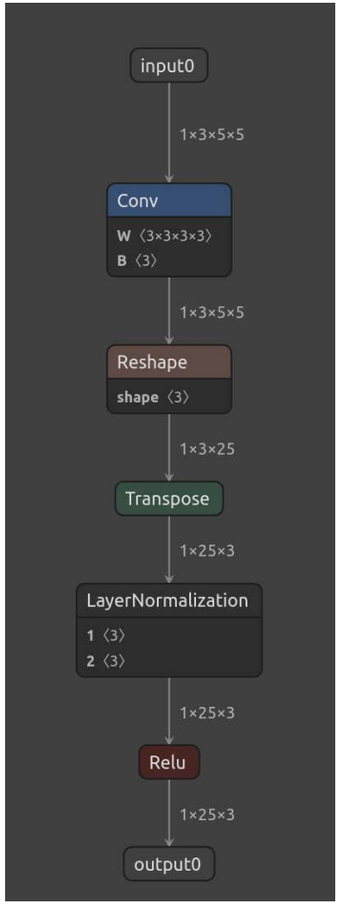
- 更加方便的添加/修改onnx节点
- 更加方便的修改子图
- 更加方便的替换算子
- (底层一般是用的onnx.helper，但是给做了一些封装)





将min-max替换成clip算子
(clip算子在onnx中有支持，所以直接替换)





将一系列复杂的LayerNorm计算替换成一个LN算子
(TensorRT 8.6以前的做法)

onnx-surgeon vs onnx.hepler

```
4
5  # 理解onnx中的组织结构
6  #   - ModelProto (描述的是整个模型的信息)
7  #   --- GraphProto (描述的是整个网络的信息)
8  #   ------ NodeProto (描述的是各个计算节点，比如conv, linear)
9  #   ------ TensorProto (描述的是tensor的信息，主要包括权重)
10 #   ------ ValueInfoProto (描述的是input/output信息)
11 #   ------ AttributeProto (描述的是node节点的各种属性信息)
12
```

onnx ProtoBuf中的IR表示

```
5  # onnx_graph_surgeon(gs)中的IR会有以下三种结构
6  # Tensor
7  #    -- 有两种类型
8  #       -- Variable:  主要就是那些不到推理不知道的变量
9  #       -- Constant:  不用推理时，而在推理前就知道的变量
10 # Node
11 #    -- 跟onnx中的NodeProto差不多
12 # Graph
13 #    -- 跟onnx中的GraphProto差不多
14
```

onnx graph surgeon中的IR表示

- gs帮助我们隐藏了很多信息
- node的属性以前使用AttributeProto保存，但是gs中统一用dict来保存

onnx-surgeon vs onnx.hepler

```python
def create_initializer_tensor(
        name: str,
        tensor_array: np.ndarray,
        data_type: onnx.TensorProto = onnx.TensorProto.FLOAT
) -> onnx.TensorProto:

    initializer = onnx.helper.make_tensor(
        name      = name,
        data_type = data_type,
        dims      = tensor_array.shape,
        vals      = tensor_array.flatten().tolist())

    return initializer
```

```python
    ########################创建第一个conv节点############################
    conv1_output_name = "conv2d_1.output"
    conv1_in_ch       = input_channel
    conv1_out_ch      = 32
    conv1_kernel      = 3
    conv1_pads        = 1

    # 创建conv节点的权重信息
    conv1_weight    = np.random.rand(conv1_out_ch, conv1_in_ch, conv1_kernel, conv1_kernel)
    conv1_bias      = np.random.rand(conv1_out_ch)

    conv1_weight_name = "conv2d_1.weight"
    conv1_weight_initializer = create_initializer_tensor(
        name         = conv1_weight_name,
        tensor_array = conv1_weight,
        data_type    = onnx.TensorProto.FLOAT)


    conv1_bias_name  = "conv2d_1.bias"
    conv1_bias_initializer = create_initializer_tensor(
        name         = conv1_bias_name,
        tensor_array = conv1_bias,
        data_type    = onnx.TensorProto.FLOAT)

    # 创建conv节点，注意conv节点的输入有3个：input, w, b
    conv1_node = onnx.helper.make_node(
        name         = "conv2d_1",
        op_type      = "Conv",
        inputs       = [
            model_input_name,
            conv1_weight_name,
            conv1_bias_name
        ],
        outputs      = [conv1_output_name],
        kernel_shape = [conv1_kernel, conv1_kernel],
        pads         = [conv1_pads, conv1_pads, conv1_pads, conv1_pads],
    )
```

使用原生的onnx.hepler创建onnx

```python
    input = gs.Variable(
            name  = "input0",
            dtype = np.float32,
            shape = (1, 3, 224, 224))

    weight = gs.Constant(
            name  = "conv1.weight",
            values = np.random.randn(5, 3, 3, 3))

    bias   = gs.Constant(
            name  = "conv1.bias",
            values = np.random.randn(5))

    output = gs.Variable(
            name  = "output0",
            dtype = np.float32,
            shape = (1, 5, 224, 224))

    node = gs.Node(
            op      = "Conv",
            inputs  = [input, weight, bias],
            outputs = [output],
            attrs   = {"pads":[1, 1, 1, 1]})

    graph = gs.Graph(
            nodes   = [node],
            inputs  = [input],
            outputs = [output])

    model = gs.export_onnx(graph)
```

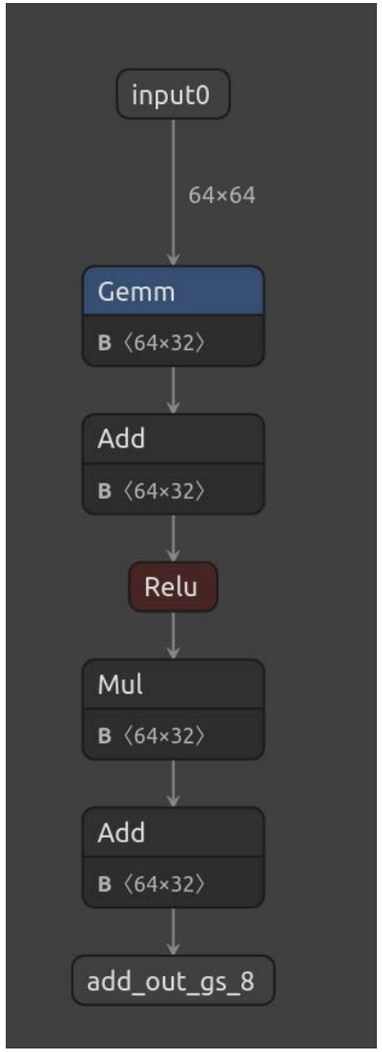
使用原生的gs创建onnx

onnx-surgeon vs onnx.hepler

gs可以自定义一些函数去创建onnx，使整个onnx的创建更加方便

```python
####################在graph注册调用的函数#########################
@gs.Graph.register()
def add(self, a, b):
    return self.layer(op="Add", inputs=[a, b], outputs=["add_out_gs"])

@gs.Graph.register()
def mul(self, a, b):
    return self.layer(op="Mul", inputs=[a, b], outputs=["mul_out_gs"])

@gs.Graph.register()
def gemm(self, a, b, trans_a=False, trans_b=False):
    attrs = {"transA": int(trans_a), "transB": int(trans_b)}
    return self.layer(op="Gemm", inputs=[a, b], outputs=["gemm_out_gs"], attrs=attrs)

@gs.Graph.register()
def relu(self, a):
    return self.layer(op="Relu", inputs=[a], outputs=["act_out_gs"])
```

```python
# 设计网络架构
gemm0    = graph.gemm(input0, consA, trans_b=True)
relu0    = graph.relu(*graph.add(*gemm0, consB))
mul0     = graph.mul(*relu0, consC)
output0  = graph.add(*mul0, consD)
```



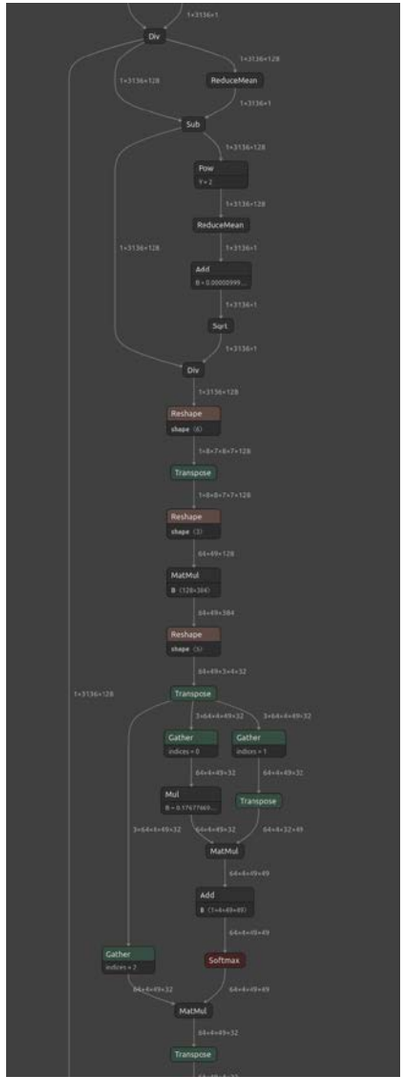
类似于onnx中symbolic符号函数来注册算子一样

我们完全可以自己创建一些算子在这里使用

onnx-surgeon vs onnx.hepler

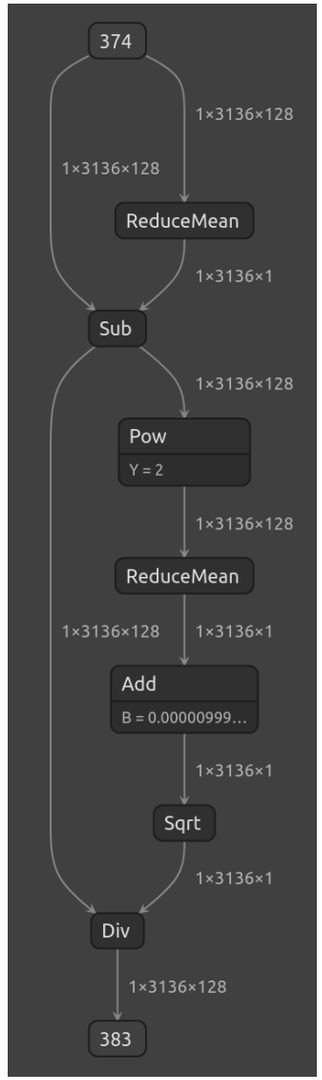
(*)polygraphy是TensorRT用来分析模型的一个非常非常重要的一个软件，会在后面的章节详细展开讲

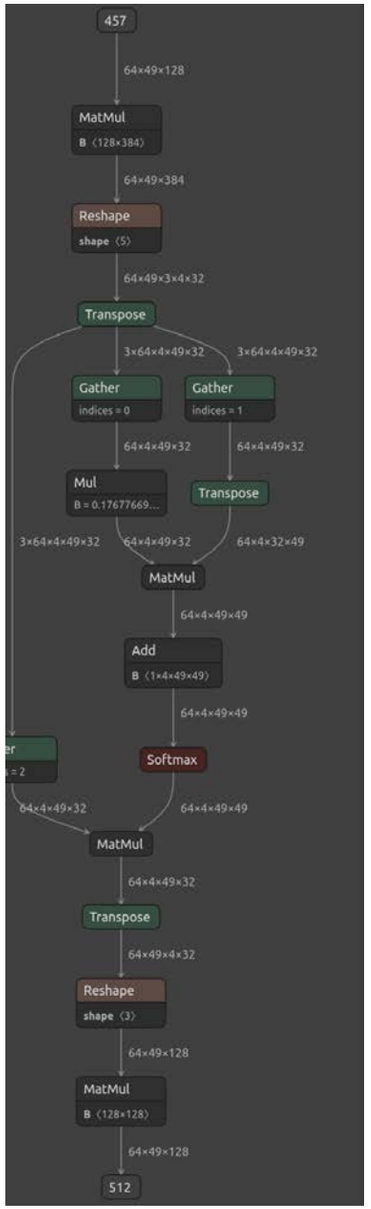
gs可以方便我们把整个网络中的一些子图给“挖”出来，以此来分析细节
(一般配合polygraphy(*)使用，去寻找量化掉精度严重的子图)



```python
# LayerNorm部分
print(tensors["374"])  # LN的input1: 1 x 3136 x 128
print(tensors["375"])  # LN的input2: 1 x 3136 x 1
print(tensors["383"])  # LN的输出:   1 x 3136 x 128
graph.inputs = [
        tensors["374"].to_variable(dtype=np.float32, shape=(1, 3136, 128))]
graph.outputs = [
        tensors["383"].to_variable(dtype=np.float32, shape=(1, 3136, 128))]
graph.cleanup()
onnx.save(gs.export_onnx(graph), "../models/swin-subgraph-LN.onnx")
```

```python
# MHSA部分
graph = gs.import_onnx(model)
tensors = graph.tensors()
print(tensors["457"])   # MHSA输入matmul:        64 x 49 x 128
print(tensors["5509"])  # MHSA输入matmul的权重: 128 x 384
print(tensors["5518"])  # MHSA输出matmul的权重: 128 x 128
print(tensors["512"])   # MHSA输出:              64 x 49 x 128
graph.inputs = [
        tensors["457"].to_variable(dtype=np.float32, shape=(64, 49, 128))]
graph.outputs = [
        tensors["512"].to_variable(dtype=np.float32, shape=(64, 49, 128))]
graph.cleanup()
onnx.save(gs.export_onnx(graph), "../models/swin-subgraph-MSHA.onnx")
```



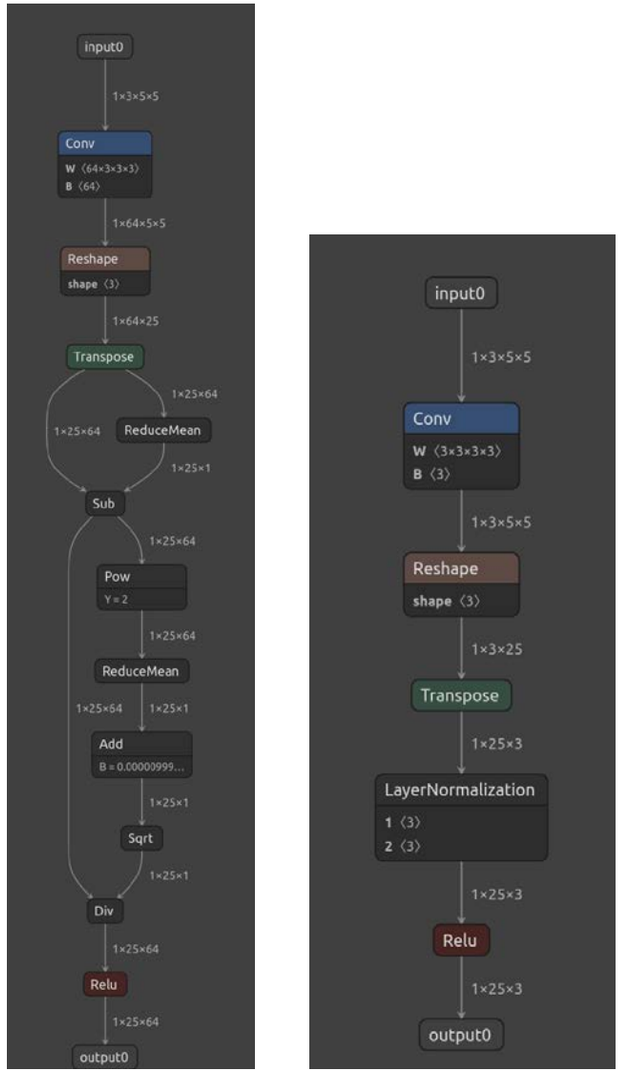


onnx-surgeon vs onnx.hepler

gs中最重要的一个特点，在于我们可以使用gs来替换算子或者创建算子
(这个会直接跟后面的TensorRT plugin绑定，实现算子的加速或者不兼容算子的实现)

```python
@gs.Graph.register()
def layerNorm(self, inputs, outputs, axis, epsilon):
    attrs = {'axis': np.int64(axis), 'epsilon': np.float(epsilon)}
    return self.layer(op="LayerNormalization", inputs=inputs, outputs=outputs, attrs=attrs)

@gs.Graph.register()
def layerNorm_default(self, inputs, outputs):
    return self.layer(op="LayerNormalization", inputs=inputs, outputs=outputs)
```

```python
def change_onnx_graph():
    graph = gs.import_onnx(onnx.load_model('../models/sample-ln-before.onnx'))
    tensors = graph.tensors()

    norm_scale = gs.Constant(name="norm.weight", values=np.ones(shape=[3], dtype=np.float32))
    norm_bias  = gs.Constant(name="norm.bias", values=np.zeros(shape=[3], dtype=np.float32))

    inputs  = [tensors["/Transpose_output_0"]]
    outputs = [tensors["/norm/Div_output_0"]]

    # 因为要替换子网，所以需要把子网和周围的所有节点都断开联系
    for item in inputs:
        item.outputs.clear()

    for item in outputs:
        item.inputs.clear()

    # 为了迎合onnx中operator中的设计，这里把scale和bias给加上
    inputs = [tensors["/Transpose_output_0"],
              norm_scale,
              norm_bias]

    # 这个onnx中的epsilon，我们给加上。当然，我们也可以选择默认的值
    epsilon = [tensors["/norm/Constant_1_output_0"]]
    print(type(epsilon[0].values))

    # 通过注册的LayerNorm，重新把断开的联系链接起来
    graph.layerNorm(inputs, outputs, axis=-1, epsilon=epsilon[0].values)

    # 删除所有额外的节点
    graph.cleanup()

    onnx.save(gs.export_onnx(graph), "../models/sample-ln-after.onnx")
```



onnx修改前(左)，onnx修改后(右)

快速分析开源代码并导出onnx

Goal: 以Swin Transformer为例学习快速导出onnx并分析onnx的方法

执行一下我们的python程序

```
/C/d/i/t/chapter3-tensorrt-basics-and-onnx (main|+555) $ cd 3.6-export-onnx-from-oss/
 - 23  7月  17:56 models
 - 16  7月  13:50 Swin-Transformer
 - 16  7月  12:54 weights
/C/d/i/t/c/3.6-export-onnx-from-oss (main|+555) $ tree models/

ny-after-simplify.onnx
ny-after-simplify-opset17.onnx
ny-before-simplify.onnx
ny.onnx

es, 4 files
/C/d/i/t/c/3.6-export-onnx-from-oss (main|+555) $ tree weights/

ny_patch4_window7_224.pth
```

3.6-export-onnx-from-oss
(快速分析开源代码)

执行结果的一部分展示

转换swin-tiny时候出现的不兼容op的解决方案

```
  symbolic_fn = _find_symbolic_in_registry(domain, op_name, opset_version, operator_export_type)
  File "/home/kalfazed/miniconda3/envs/swin/lib/python3.7/site-packages/torch/onnx/utils.py", line 947, in _find_symbolic_in_registry
    return sym_registry.get_registered_op(op_name, domain, opset_version)
  File "/home/kalfazed/miniconda3/envs/swin/lib/python3.7/site-packages/torch/onnx/symbolic_registry.py", line 112, in get_registered_op
    raise RuntimeError(msg)
RuntimeError: Exporting the operator roll to ONNX opset version 9 is not supported. Please feel free to request support or submit a pull request o
PyTorch GitHub.
```

先使用opset9看看是否可以导出

```
  File "/home/kalfazed/miniconda3/envs/swin/lib/python3.7/site-packages/torch/onnx/utils.py", line 947, in _find_symbolic_
    return sym_registry.get_registered_op(op_name, domain, opset_version)
  File "/home/kalfazed/miniconda3/envs/swin/lib/python3.7/site-packages/torch/onnx/symbolic_registry.py", line 112, in get
    raise RuntimeError(msg)
RuntimeError: Exporting the operator roll to ONNX opset version 12 is not supported. Please feel free to request support o
```

使用opset12依然会出现问题

```python
# reverse cyclic shift
if self.shift_size > 0:
    if not self.fused_window_process:
        shifted_x = window_reverse(attn_windows, self.window_size, H, W)  # B H' W' C
        x = torch.roll(shifted_x, shifts=(self.shift_size, self.shift_size), dims=(1, 2))
    else:
        x = WindowProcessReverse.apply(attn_windows, B, H, W, C, self.shift_size, self.window_size)
else:
    shifted_x = window_reverse(attn_windows, self.window_size, H, W)  # B H' W' C
    x = shifted_x
x = x.view(B, H * W, C)
x = shortcut + self.drop_path(x)
```

swin/models/swin_transformer.py

转换swin-tiny时候出现的不兼容op的解决方案

TORCH.ROLL

torch.roll(*input*, *shifts*, *dims=None*) → Tensor

Roll the tensor `input` along the given dimension(s). Elements that are shifted beyond the last position are re-introduced at the first position. If `dims` is *None*, the tensor will be flattened before rolling and then restored to the original shape.

Parameters:

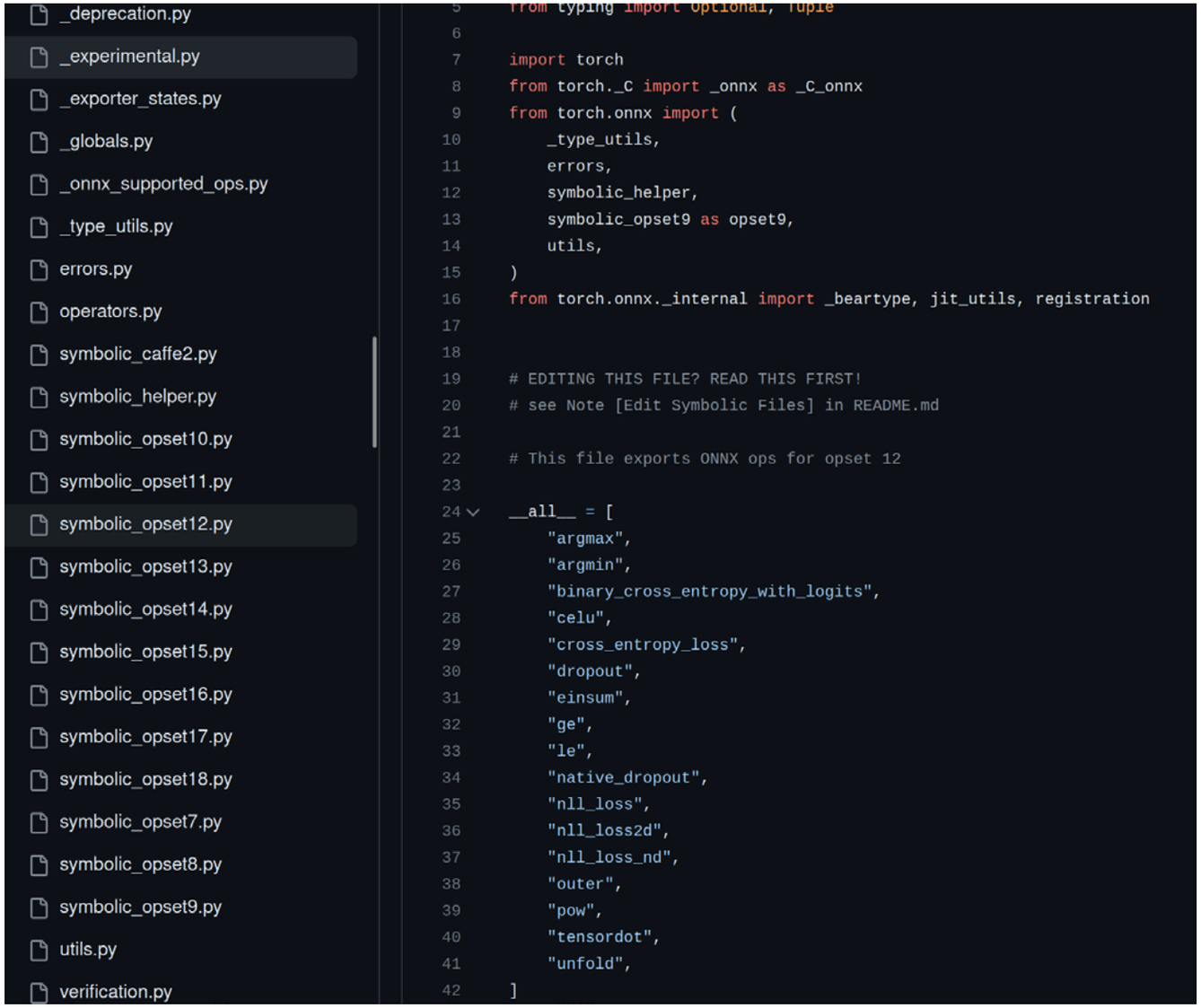
- **input** (*Tensor*) – the input tensor.
- **shifts** (*int or tuple of ints*) – The number of places by which the elements of the tensor are shifted. If shifts is a tuple, dims must be a tuple of the same size, and each dimension will be rolled by the corresponding value
- **dims** (*int or tuple of ints*) – Axis along which to roll

https://pytorch.org/docs/stable/generated/torch.roll.html

RandomUniform	1	
RandomUniformLike	1	
Reciprocal	13, 6, 1	
ReduceMax	18, 13, 12, 11, 1	
ReduceMean	18, 13, 11, 1	
ReduceMin	18, 13, 12, 11, 1	
ReduceProd	18, 13, 11, 1	
ReduceSum	13, 11, 1	
Reshape	19, 14, 13, 5, 1	
Resize	19, 18, 13, 11, 10	
ReverseSequence	10	
RoiAlign	16, 10	
Round	11	
STFT	17	
Scan	19, 16, 11, 9, 8	
Scatter (deprecated)	11, 9	
ScatterElements	18, 16, 13, 11	
ScatterND	18, 16, 13, 11	

https://github.com/onnx/onnx/blob/main/docs/Operators.md

转换swin-tiny时候出现的不兼容op的解决方案



https://github.com/pytorch/pytorch/blob/main/torch/onnx/symbolic_opset12.py

转换swin-tiny时候出现的不兼容op的解决方案

```
[station]~/m/e/s/l/p/s/torch [1]$ cd onnx/
.rw-rw-r--  16k 27  2月   2021  __init__.py
drwxrwxr-x    - 16  7月  12:58  __pycache__
.rw-rw-r--  579 27  2月   2021  operators.py
.rw-rw-r-- 9.6k 27  2月   2021  symbolic_caffe2.py
.rw-rw-r--  35k 27  2月   2021  symbolic_helper.py
.rw-rw-r-- 1.8k 27  2月   2021  symbolic_opset7.py
.rw-rw-r--  11k 27  2月   2021  symbolic_opset8.py
.rw-rw-r-- 120k 27  2月   2021  symbolic_opset9.py
.rw-rw-r--  12k 27  2月   2021  symbolic_opset10.py
.rw-rw-r--  40k 27  2月   2021  symbolic_opset11.py
.rw-rw-r-- 7.7k 27  2月   2021  symbolic_opset12.py
.rw-rw-r-- 8.0k 27  2月   2021  symbolic_opset13.py
.rw-rw-r-- 4.9k 27  2月   2021  symbolic_registry.py
.rw-rw-r--  60k 27  2月   2021  utils.py
[station]~/m/e/s/l/p/s/t/onnx $ pwd
/home/kalfazed/miniconda3/envs/swin/lib/python3.7/site-packages/torch/onnx
```

因为是opset的问题，所以我们在这里面找

```python
from sys import maxsize as maxsize
```

```python
@parse_args('v', 'is', 'is')
def roll(g, self, shifts, dims):
    assert len(shifts) == len(dims)

    result = self
    for i in range(len(shifts)):
        shapes = []
        shape = sym_help._slice_helper(g,
                                       result,
                                       axes=[dims[i]],
                                       starts=[-shifts[i]],
                                       ends=[maxsize])
        shapes.append(shape)
        shape = sym_help._slice_helper(g,
                                       result,
                                       axes=[dims[i]],
                                       starts=[0],
                                       ends=[-shifts[i]])
        shapes.append(shape)
        result = g.op("Concat", *shapes, axis_i=dims[i])

    return result
```

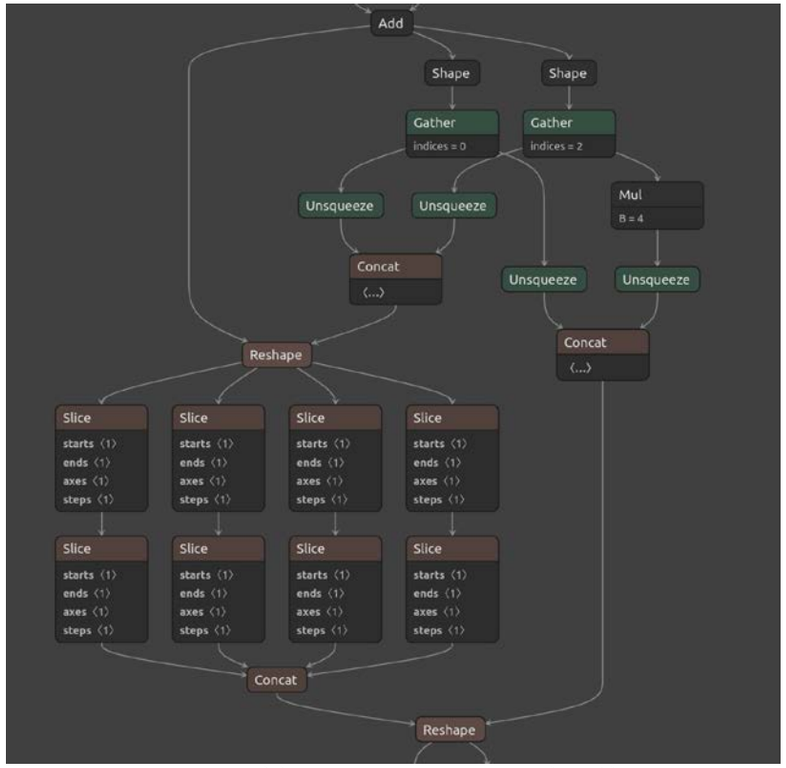
添加pytorch2onnx的roll算子实现

swin-tiny的onnx分析

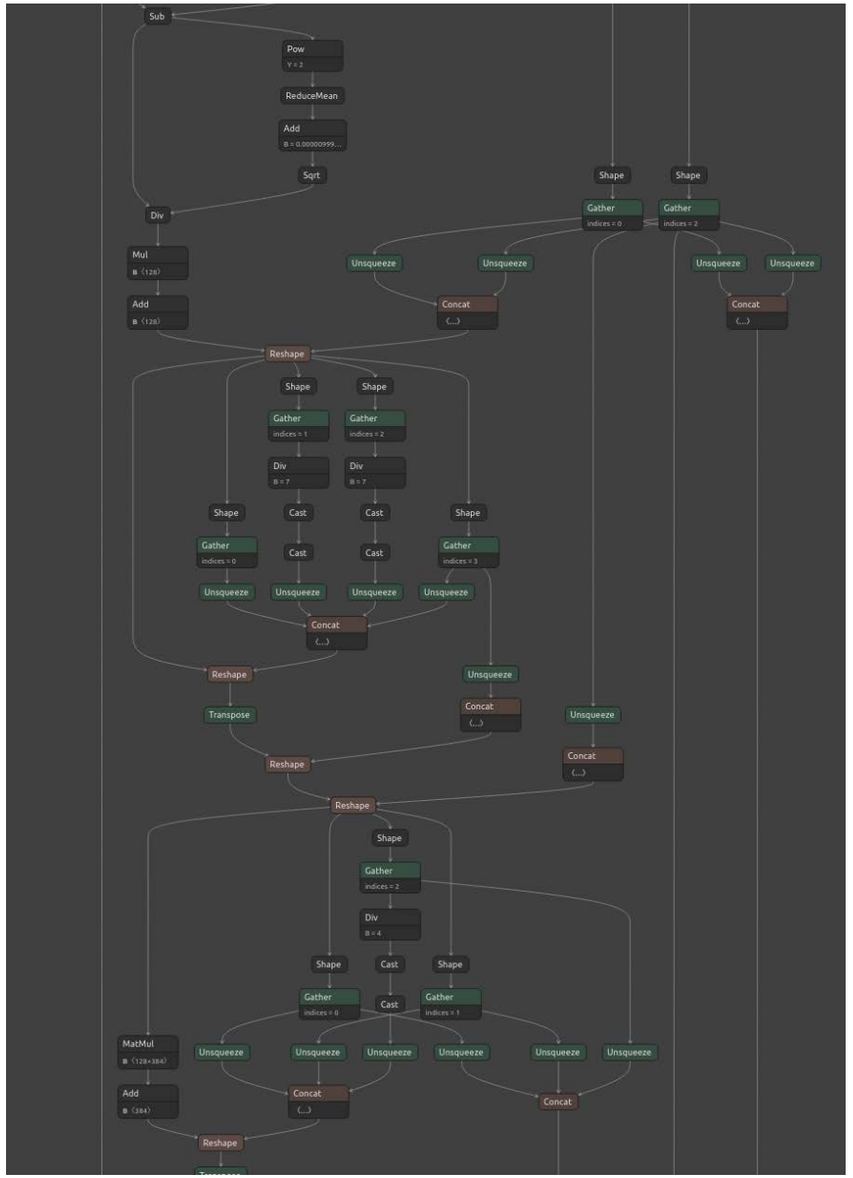
```
Finished normal onnx export
[station]~/C/d/i/t/c/3/Swin-Transformer (
```

成功通过

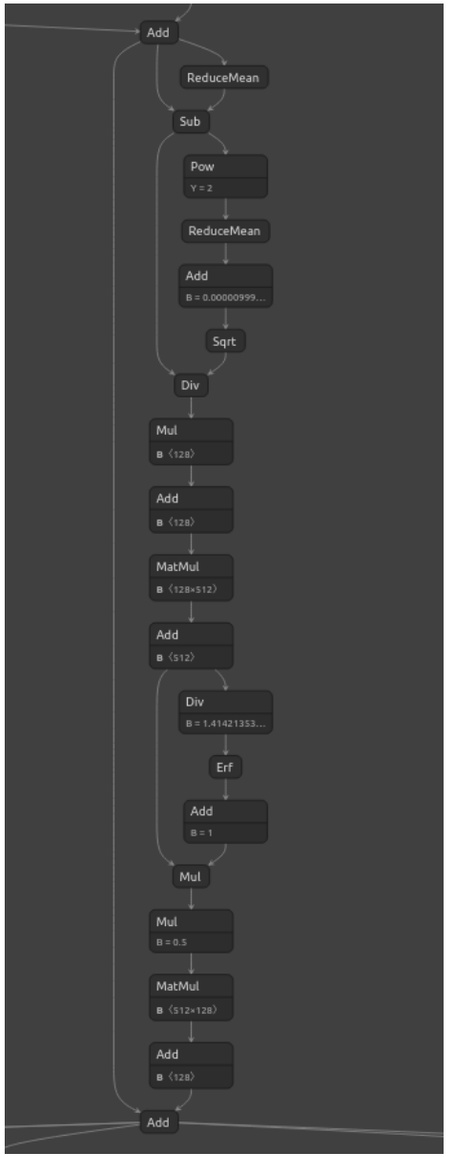
opset9即便把Roll的问题解决还会出现其他算子不兼容问题，所以直接在opset12上进行解决



swin的window shift部分

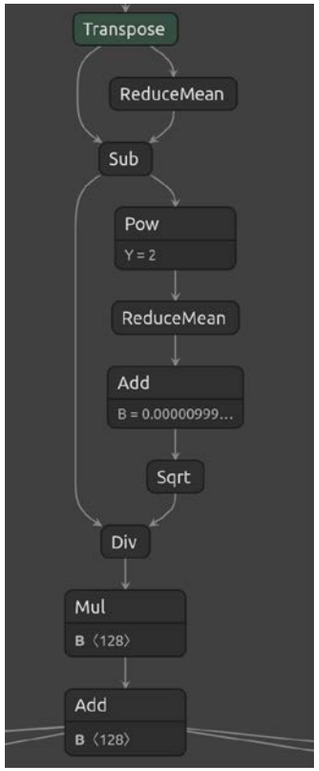


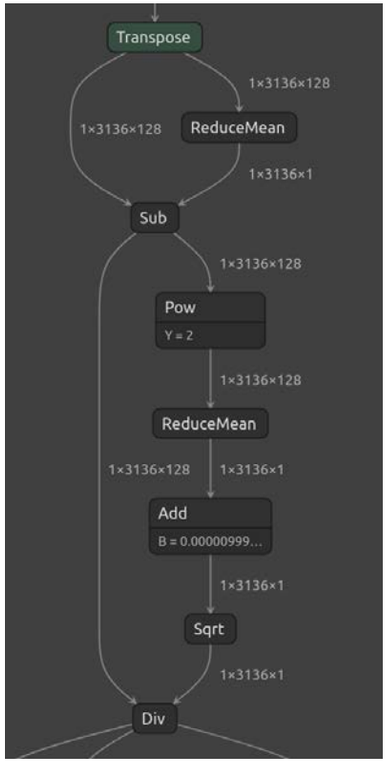
swin的attention部分



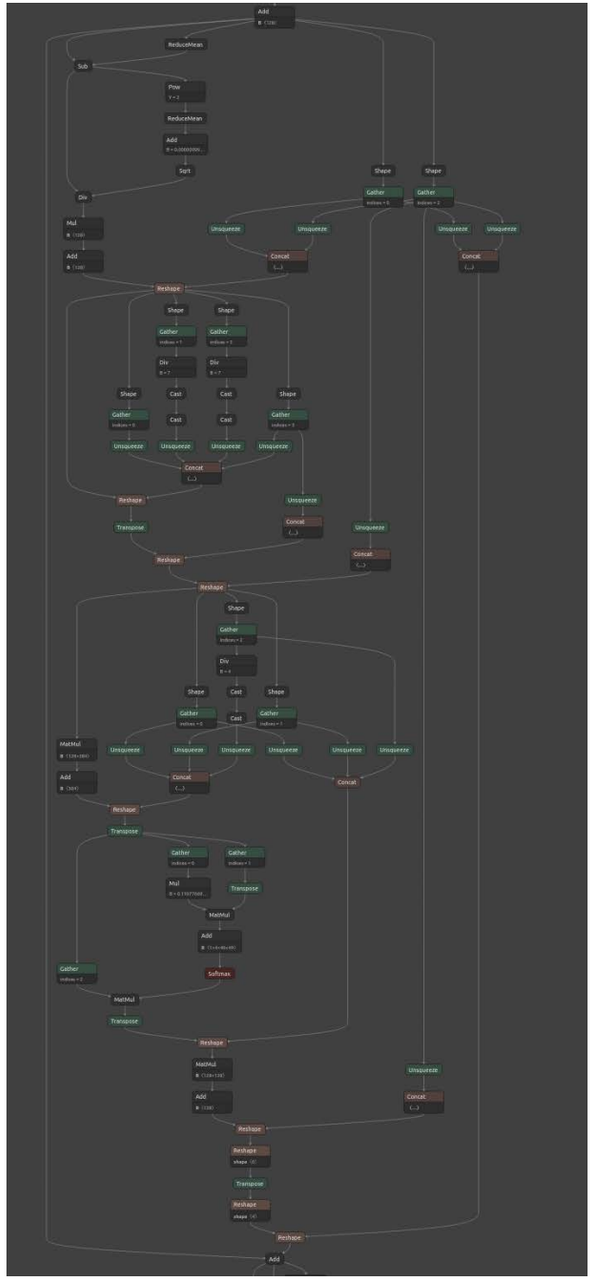
swin的LN部分

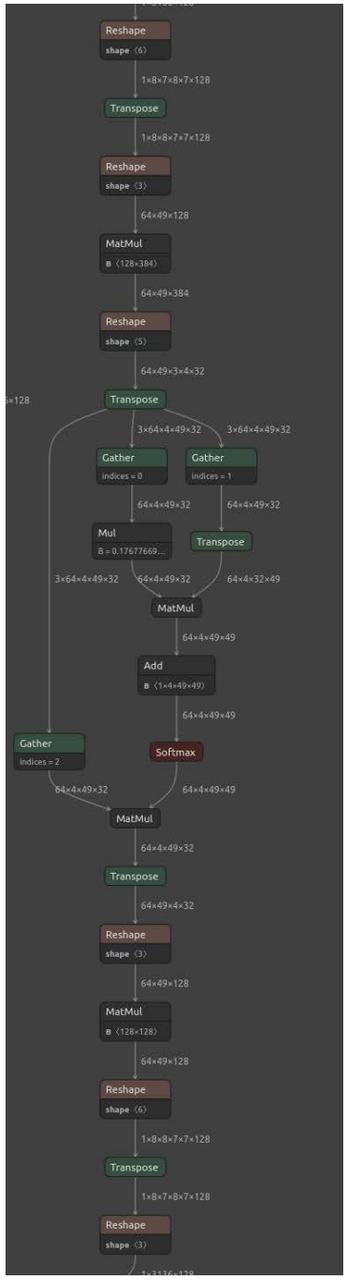
swin-tiny的onnx分析





onnxsim对LN的简化

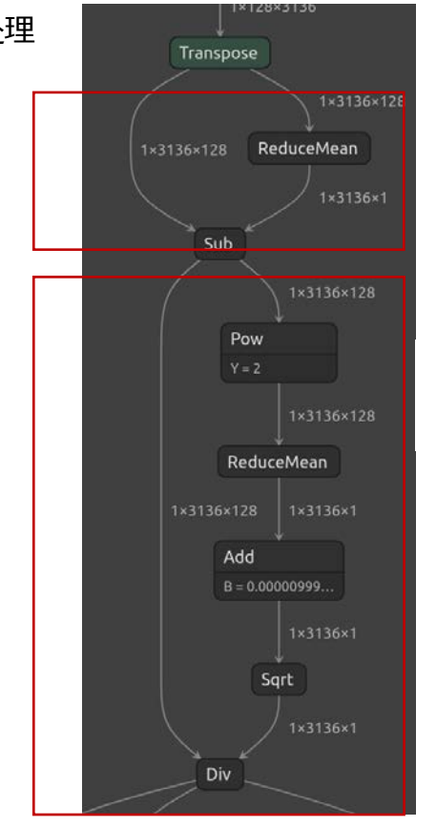




onnxsim对attention的简化

根据onnx中的Proto信息，修改onnx

Transformer的LN处理



$\mathbf{a}^l - \mu^l$

$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2}$

$\hat{\mathbf{a}}^l = \frac{\mathbf{a}^l - \mu^l}{\sqrt{(\sigma^l)^2 + \epsilon}}$

$\mathbf{h}^l = f(\mathbf{g}^l \odot \hat{\mathbf{a}}^l + \mathbf{b}^l)$

$\mathbf{h} = f(\frac{\mathbf{g}}{\sqrt{\sigma^2 + \epsilon}} \odot (\mathbf{a} - \mu) + \mathbf{b})$

LN和BN的计算其实是差不多的

- 两者都是在做normalization，只不过是在不同的维度上

step1：mean和std的计算

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2}$$

step2：normalization

$$\hat{\mathbf{a}}^l = \frac{\mathbf{a}^l - \mu^l}{\sqrt{(\sigma^l)^2 + \epsilon}}$$

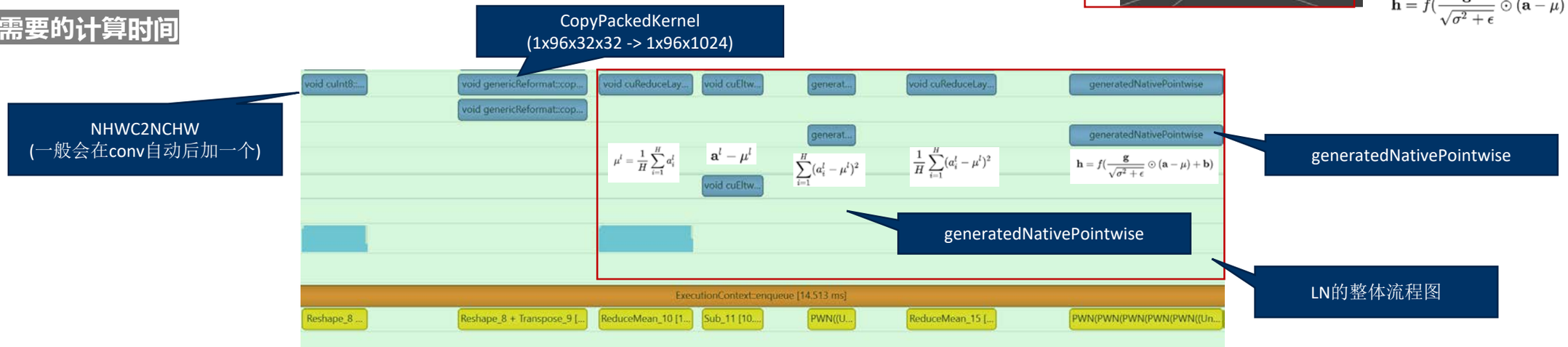
step3：创建一个可以学习的mean和var，最后在接一个激活函数

$$\mathbf{h}^l = f(\mathbf{g}^l \odot \hat{\mathbf{a}}^l + \mathbf{b}^l)$$

总结下来

$$\mathbf{h} = f(\frac{\mathbf{g}}{\sqrt{\sigma^2 + \epsilon}} \odot (\mathbf{a} - \mu) + \mathbf{b})$$

LN所需要的计算时间



計算時間大概是165 us、主要卡在了reshape和pointwise上了

根据onnx中的Proto信息，修改onnx

HardSigmoid	6, 1	18
HardSwish	14	14
LayerNormalization	17	17, 18
LeakyRelu	16, 6, 1	16
LessOrEqual	16, 12	16
LogSoftmax	13, 11, 1	13, 18

https://github.com/onnx/onnx/blob/main/docs/Operators.md

TensorRT 8.6 EA Release - 2023-3-13

Added

For more details, see the 8.6 EA release notes for new features added in TensorRT 8.6.

- Added support for `GroupNormalization`, `LayerNormalization`, `IsInf` operations
- Added support for INT32 input types for `Argmin`, `Argmax`, and `TopK`
- Added support for `ReverseSequence` operators with dynamic shapes
- Added support for `TopK` operators with dynamic `K` values
- Added `OnnxParserFlag` enum and `setFlag` interfaces to the ONNX parser to modify the default parsing behavior
- Added metadata tracking, now ONNX node metadata will be embedded into TensorRT layers

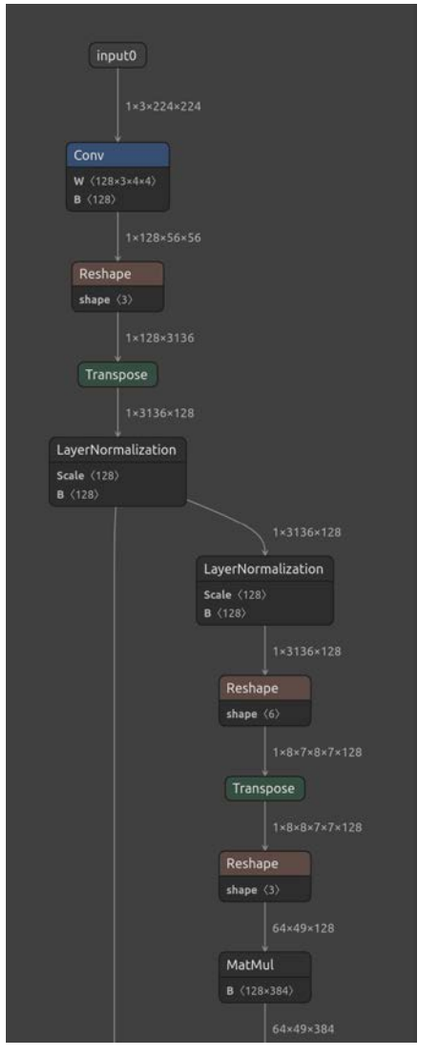
Changed

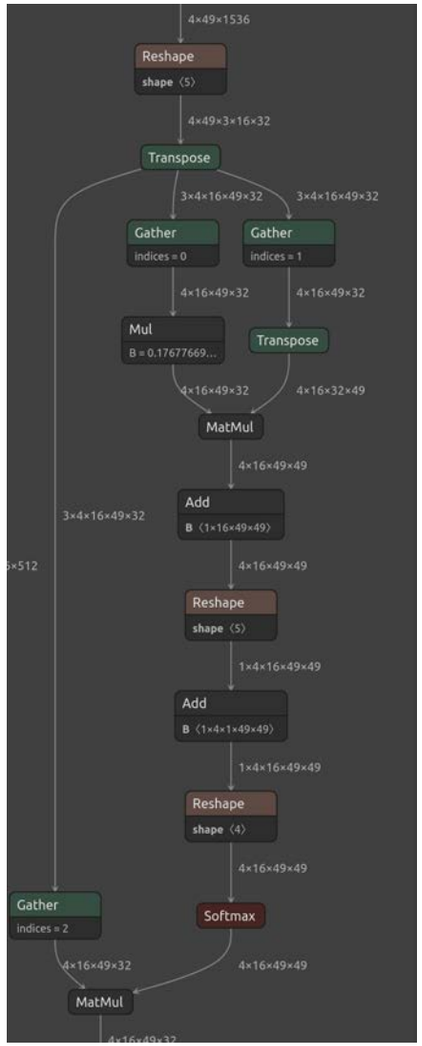
- All cast operations will now use the new `CastLayer` over the pervious `IdentityLayer`.

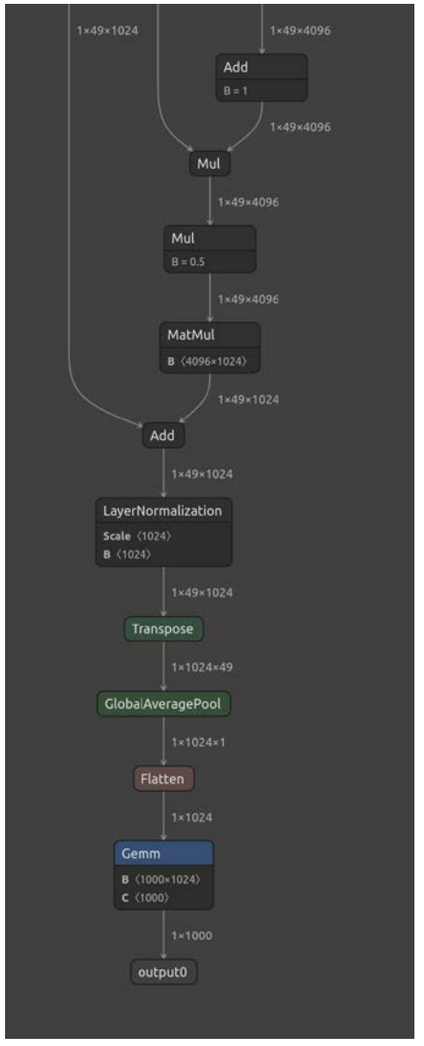
https://github.com/onnx/onnx-tensorrt/blob/main/docs/Changelog.md

```python
def export_norm_onnx(model, file, input):
    torch.onnx.export(
        model         = model,
        args          = (input,),
        f             = file,
        input_names   = ["input0"],
        output_names  = ["output0"],
        opset_version = 17)

    print("Finished normal onnx export")

    model_onnx = onnx.load(file)

    # 检查导入的onnx model
    onnx.checker.check_model(model_onnx)

    # 使用onnx-simplifier来进行onnx的简化。
    print(f"Simplifying with onnx-simplifier {onnxsim.__version__}...")
    model_onnx, check = onnxsim.simplify(model_onnx)
    assert check, "assert check failed"
    onnx.save(model_onnx, file)
```

成功导出onnx







开头，中间，结尾部分的onnx展示

TensorRT导出

导出了onnx不是重点。我们需要将这些onnx导出为TensorRT模型，并查看性能。我们有几种方式导出

- trtexec命令行 (快速测试推理引擎)
- TensorRT python API (大量的单元测试)
- TensorRT C++ API (结合其他的前处理后处理进行底层部署)

使用trtexec

Goal: 学习使用trtexec，以及通过trtexec的日志理解TensorRT内部所做的优化

执行一下我们的shell脚本

```
../3.7-trtexec-analysis/
├── build
│   ├── engines
│   │   └── sample.engine
│   └── log
│       └── sample
│           ├── build
│           │   ├── build_layer_info.log
│           │   ├── build.log
│           │   ├── build_output.log
│           │   └── build_profile.log
│           ├── infer
│           │   ├── infer_layer_info.log
│           │   ├── infer.log
│           │   ├── infer_output.log
│           │   └── infer_profile.log
│           └── profile
│               └── sample.nsys-rep
├── models
│   ├── resnet50.engine
│   ├── resnet50.onnx
│   ├── sample2.engine
│   ├── sample2.onnx
│   ├── sample.engine
│   ├── sample.onnx
│   ├── swin-tiny-op17.onnx
│   ├── vgg16.engine
│   └── vgg16.onnx
├── src
│   ├── load_torchvision.py
│   ├── sample_cbr.py
│   └── sample_reshape.py
└── tools
    ├── build.sh
    ├── infer.sh
    └── profile.sh
```

3.7-学习如何使用trtexec

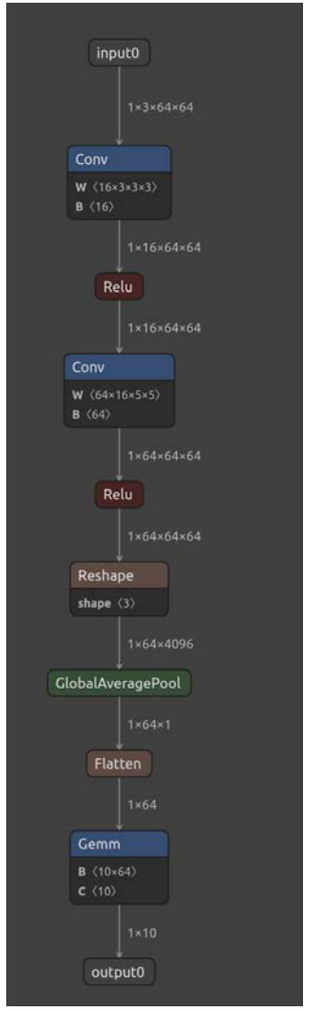
```
[
  { "count" : 131343 }
, { "name" : "/conv1/Conv + /act1/Relu", "timeMs" : 776.193, "averageMs" : 0.00590966, "medianMs" : 0.006144, "percentage" : 100 }
]
```

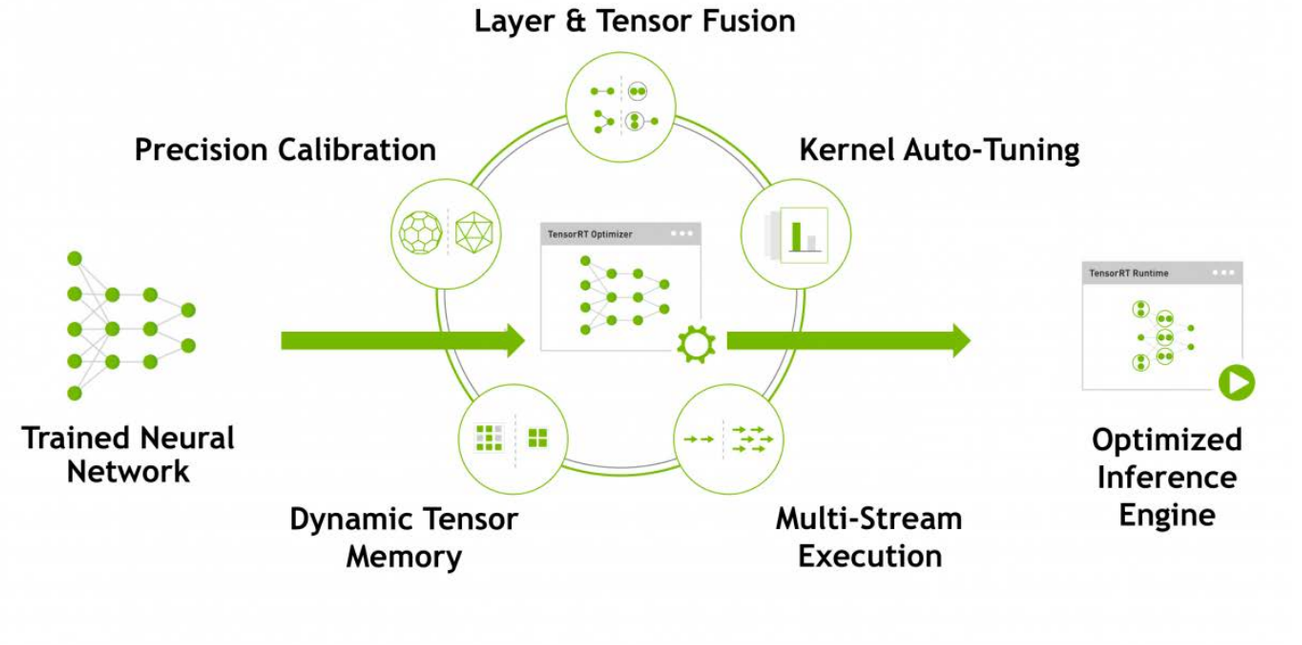
```
[07/23/2023-19:20:27] [I] Starting inference
[07/23/2023-19:20:31] [I] The e2e network timing is not reported since it is inaccurate due to the extra synchronizations wh
[07/23/2023-19:20:31] [I] To show e2e network timing report, add --separateProfileRun to profile layer timing in a separate
[07/23/2023-19:20:31] [I] Output Tensors:
[07/23/2023-19:20:31] [I] output0: (1x16x3x3)
[07/23/2023-19:20:31] [I] 0.317039 0.0287001 0.182876 0.229466 0.663934 0.626932 0.433769 0 0.395937 0 0.0185916 0.25016 0 0
[07/23/2023-19:20:31] [I]
[07/23/2023-19:20:31] [I] === Profile (131343 iterations ) ===
[07/23/2023-19:20:31] [I]                                Layer   Time (ms)   Avg. Time (ms)   Median Time (ms)   Time %
[07/23/2023-19:20:31] [I]  /conv1/Conv + /act1/Relu      776.19          0.0059           0.0061    100.0
[07/23/2023-19:20:31] [I]                                Total      776.19          0.0059           0.0061    100.0
[07/23/2023-19:20:31] [I]
```

执行结果的一部分展示

trtexec

是TensorRT中的一个非常好用的快速生成trt推理引擎的工具。便于我们不使用API快速查看推理引擎的加速效果，以及TensorRT优化后的模型架构






```
Layer
Reformatting CopyNode for Input Tensor 0 to /conv1/Conv + /act1/Relu
/conv1/Conv + /act1/Relu
/conv2/Conv + /act2/Relu
Reformatting CopyNode for Input Tensor 0 to /Reshape
/avgpool/GlobalAveragePool
/head/Gemm
Total
```

trtexec

trtexec提供的参数非常的多。我这里做了一些几个脚本在tools/目录下，方便大家使用

```
trtexec --onnx=${1} \
        --memPoolSize=workspace:2048 \
        --saveEngine=${ENGINE_PATH}/${PREFIX}.engine \
        --profilingVerbosity=detailed \
        --dumpOutput \
        --dumpProfile \
        --dumpLayerInfo \
        --exportOutput=${LOG_PATH}/build_output.log\
        --exportProfile=${LOG_PATH}/build_profile.log \
        --exportLayerInfo=${LOG_PATH}/build_layer_info.log \
        --warmUp=200 \
        --iterations=50 \
        > ${LOG_PATH}/build.log
```

tools/build.sh

trtexec

trtexec提供的参数非常的多。我这里做了一些几个脚本在tools/目录下，方便大家使用

```
trtexec --loadEngine=${ENGINE_PATH}/${PREFIX}.engine \
        --verbose \
        --dumpOutput \
        --dumpProfile \
        --dumpLayerInfo \
        --exportOutput=${LOG_PATH}/infer_output.log\
        --exportProfile=${LOG_PATH}/infer_profile.log \
        --exportLayerInfo=${LOG_PATH}/infer_layer_info.log \
        --warmUp=200 \
        --iterations=50 \
        > ${LOG_PATH}/infer.log
```

tools/infer.sh

trtexec

trtexec提供的参数非常的多。我这里做了一些几个脚本在tools/目录下，方便大家使用

```
nsys profile \
    --output=${LOG_PATH}/${PREFIX} \
    --force-overwrite true \
    trtexec --loadEngine=${ENGINE_PATH}/${PREFIX}.engine \
            --warmUp=200 \
            --iterations=50 \
```

tools/profile.sh

trtexec

trtexec的参数一览，大家去阅读一下TensorRT官方文档。链接如下
https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#trtexec

trtexec可以配合nsys一起使用进行profile分析
我们对模型进行优化的时候，这个是我们要做的第一步事情