TensorRT 教程

基于 8.2.3 版本

NVIDIA DevTech. Wei Li

2022年3月28日

目录

TensorRT 简介

- TensorRT 基本特性和用法
- Workflow：使用 TensorRT API 搭建
- Workflow：使用 Parser
- Workflow：使用框架内 TensorRT 接口

开发辅助工具

- trtexec
- Netron
- polygraphy
- onnx-graphsurgeon
- Nsight Systems

插件书写

- 使用 Plugin 的简单例子
- TensorRT 与 Plugin 的交互
- 关键 API
- 结合使用 Parser 和 Plugin
- Plugin 高级话题
- 使用 Plugin 的例子

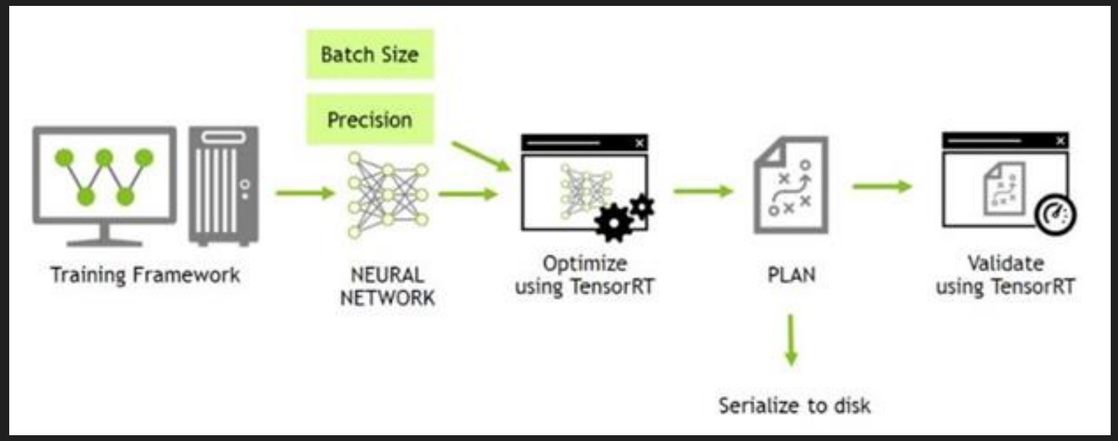
TensorRT 高级话题

- 多 Optimization Profile
- 多 Stream
- 多 Context
- CUDA Graph
- Timing Cache
- Algorithm Selector
- Refit
- Tactic Source
- 更多工具

- https://github.com/NVIDIA/trt-samples-for-hackathon-cn/tree/master/cookbook（本教程配套代码，以及更多范例和文档）

• TensorRT 简介

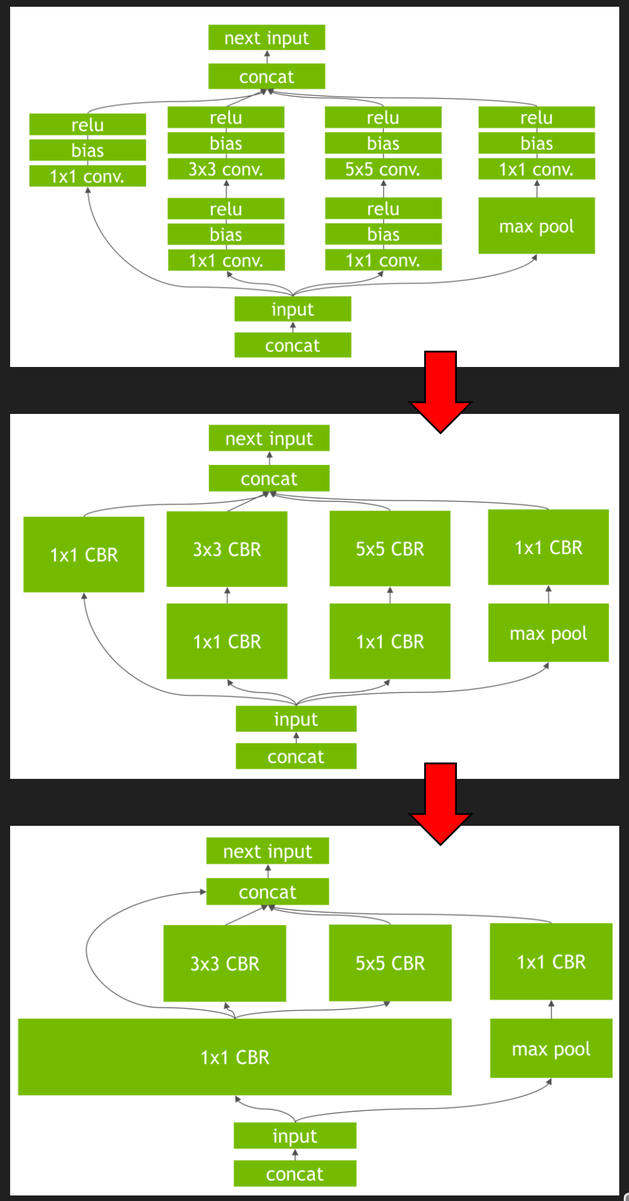
- TensorRT（最新版本 8.2.3GA 和 8.4.0EA）
  - 用于高效实现已训练好的深度学习模型的**推理过程**的 SDK
  - 内含**推理优化器**和**运行时环境**
  - 使 DL 模型能以**更高吞吐量**和**更低的延迟**运行
  - 有 C++ 和 python 的 API，完全等价可以混用



- 链接
  - https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html（TensorRT 文档）
  - https://docs.nvidia.com/deeplearning/tensorrt/api/c_api（C++ API 文档）
  - https://docs.nvidia.com/deeplearning/tensorrt/api/python_api/（python API 文档）
  - https://developer.nvidia.com/nvidia-tensorrt-download（TensorRT 下载）
  - https://github.com/NVIDIA/trt-samples-for-hackathon-cn/tree/master/cookbook（本教程配套代码，以及教程外的更多样例）

TensorRT 简介

- TensorRT 做的工作
  - **构建期（推理优化器）**
    - 模型解析 / 建立　　加载 Onnx 等其他格式的模型 / 使用原生 API 搭建模型
    - 计算图优化　　横向层融合（Conv），纵向层融合（Conv+add+ReLU），……
    - 节点消除　　去除无用层，节点变换（Pad，Slice，Concat，Shuffle），……
    - 多精度支持　　FP32 / FP16 / INT8 / TF32（可能插入 reformat 节点）
    - 优选 kernel / format　　硬件有关优化
    - 导入 plugin　　实现自定义操作
    - 显存优化　　显存池复用
  - **运行期（运行时环境）**
    - 运行时环境　　对象生命期管理，内存显存管理，异常处理
    - 序列化反序列化　　推理引擎保存为文件或从文件中加载

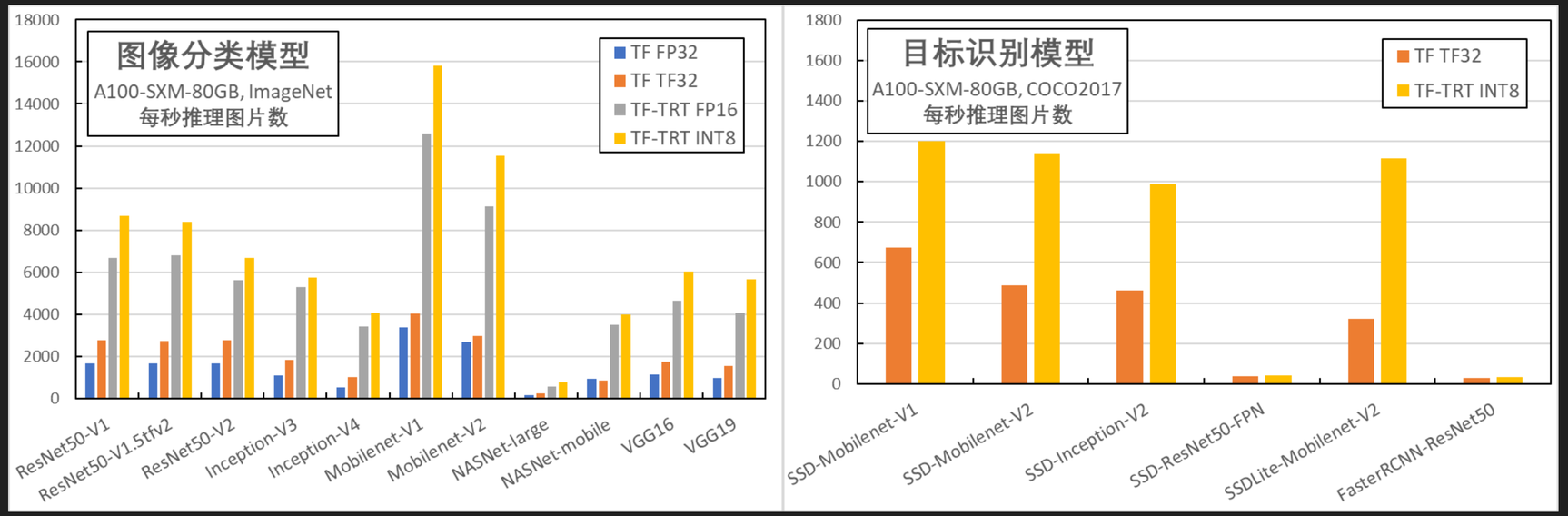


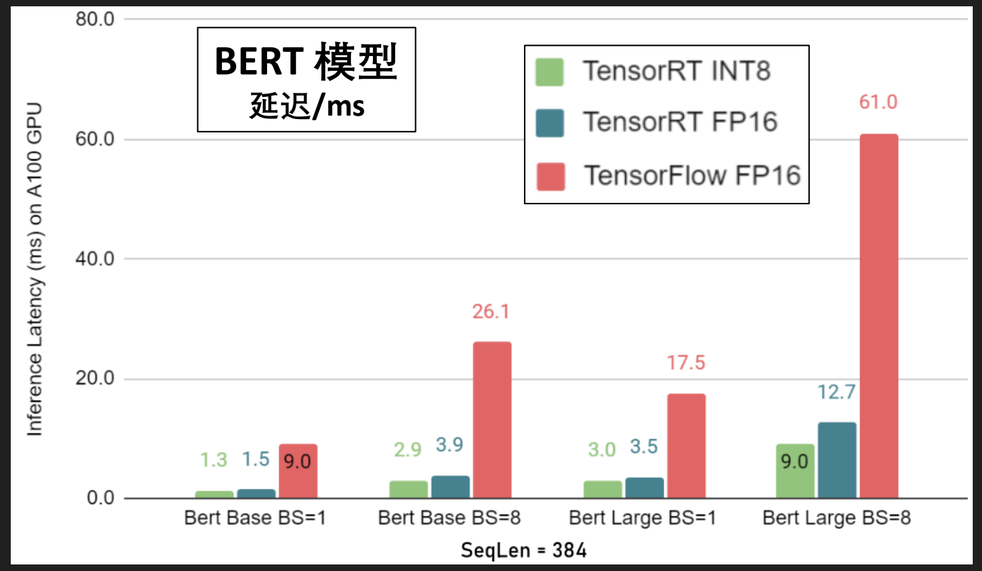
- https://developer.nvidia.com/blog/production-deep-learning-nvidia-gpu-inference-engine/

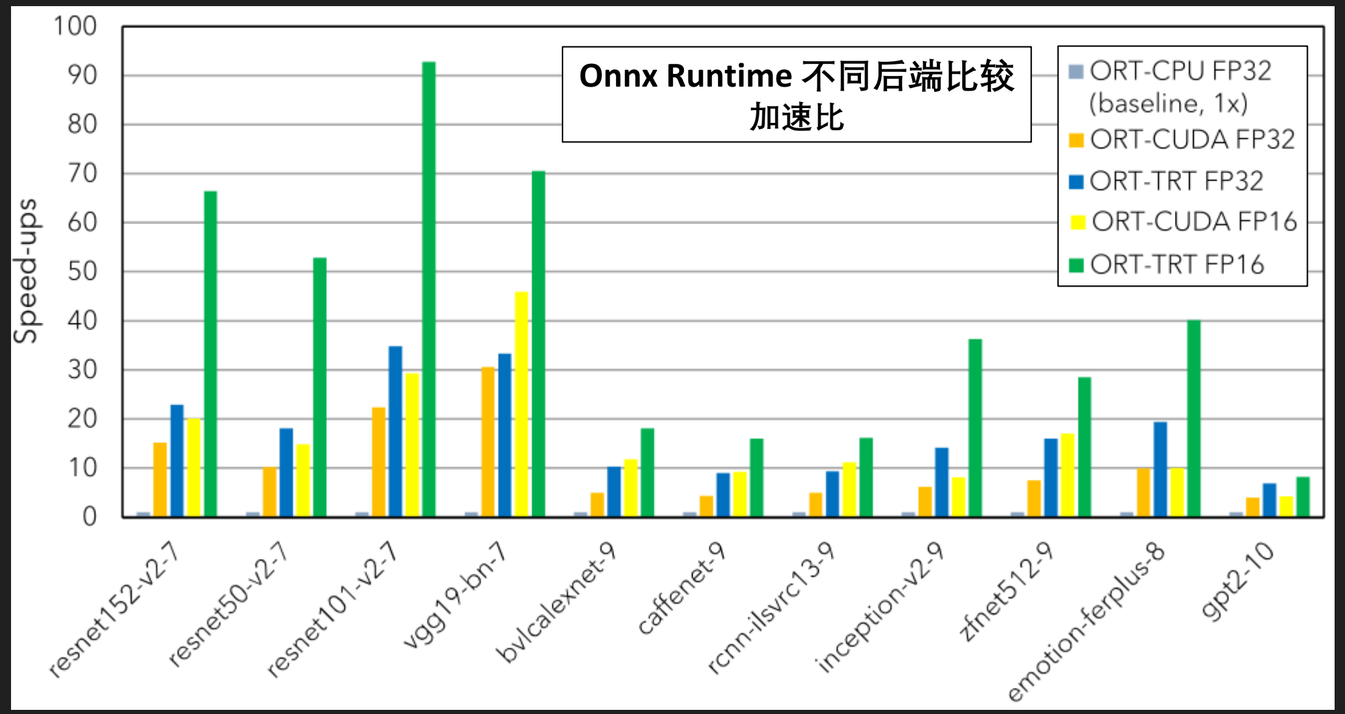
TensorRT 简介

TensorRT 的表现

- 不同模型加速效果不同
- 选用高效算子提升运算效率
- 算子融合减少访存数据、提高访问效率
- 使用低精度数据类型，节约时间空间







- S31732, Zheng(2021) Inference with Tensorflow2 Integrated with TensorRT
- S31876, Lee(2021) Accelerate Deep Learning Inference with TensorRT 8.0
- S32224, Li(2021) Accelerating Deep Learning Inference With Onnx-Runtime TensorRT

TensorRT 基本流程

- 范例代码
  - 01-SimpleDemo/TensorRT8

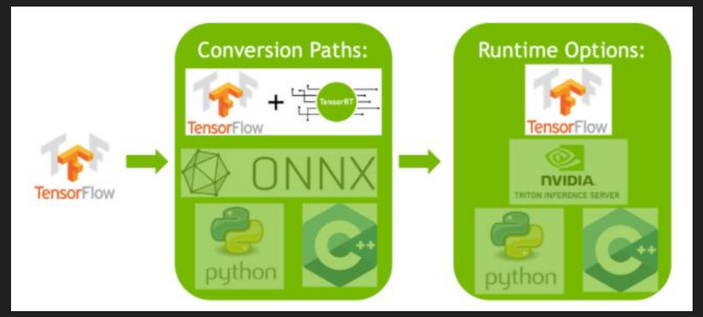
    TRT8-cudart.py 或 TRT8.cpp（python 和 C++ 等价版本）

- 基本流程
  - 构建期
    - 建立 Builder（引擎构建器）
    - 创建 Network（计算图内容）
    - 生成 SerializedNetwork（网络的 TRT 内部表示）
  - 运行期
    - 建立 Engine 和 Context
    - Buffer 相关准备（Host 端 + Device 端 + 拷贝操作）
    - 执行推理（Execute）

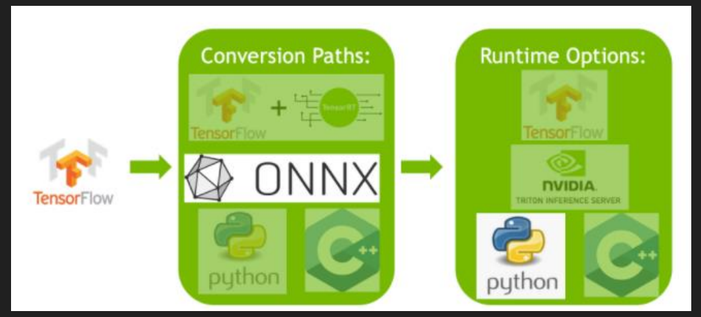
```python
logger = trt.Logger(trt.Logger.ERROR)
if os.path.isfile(trtFile):
    with open(trtFile, 'rb') as f:
        engineString = f.read()
else:
    builder = trt.Builder(logger)
    network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
    profile = builder.create_optimization_profile()
    config = builder.create_builder_config()
    config.max_workspace_size = 1 << 30

    inputTensor = network.add_input('inputT0', trt.DataType.FLOAT, [-1, -1, -1])
    profile.set_shape(inputTensor.name, [1, 1, 1], [3, 4, 5], [6, 8, 10])
    config.add_optimization_profile(profile)

    identityLayer = network.add_identity(inputTensor)
    network.mark_output(identityLayer.get_output(0))

    engineString = builder.build_serialized_network(network, config)
    with open(trtFile, 'wb') as f:
        f.write(engineString)                                                    构建期

engine = trt.Runtime(logger).deserialize_cuda_engine(engineString)               运行期
context = engine.create_execution_context()

dataShape = [3, 4, 5]
context.set_binding_shape(0, dataShape)
data = np.arange(np.prod(dataShape), dtype=np.float32).reshape(*dataShape)
bufferH = [ np.ascontiguousarray(data.reshape(-1)) ]
bufferD = [ cudart.cudaMalloc(bufferH[0].nbytes)[1] ]

cudart.cudaMemcpy(bufferD[0], bufferH[0].ctypes.data, bufferH[0].nbytes,
                  cudart.cudaMemcpyKind.cudaMemcpyHostToDevice)
context.execute_v2(bufferD)
cudart.cudaMemcpy(bufferH[i].ctypes.data, bufferD[i], bufferH[i].nbytes,
                  cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost)

cudart.cudaFree(bufferD[0])
```

• Workflow

- 使用框架自带 TRT 接口（TF-TRT，Torch-TensorRT）
  - 简单灵活，部署仍在原框架中，无需书写 Plugin



- 使用 Parser（TF/Torch/… → ONNX[1] → TensorRT）
  - 流程成熟，ONNX 通用性好，方便网络调整，兼顾效率性能



- 使用 TensorRT 原生 API 搭建网络
  - 性能最优，精细网络控制，兼容性最好

方法	易用性	性能	兼容性	开发效率	遇到不支持的OP
框架自带接口	★★★	★	★★	★★★	返回原框架计算
使用Parser	★★	★★☆	★★☆	★★	改网/改Parser/写Plugin
API搭建	★	★★★	★★★	★	写Plugin

- [1] TensorRT 也支持 UFF 和 prototxt 格式的网络，但在未来版本中将被废弃

• Workflow：使用 TensorRT API 搭建

- 使用 API 完整搭建一个 MNIST 手写识别模型的示例
  - 范例代码
    - 03-APIModel\05-MNISTExample
  - 基本流程：
    - TensorFlow 中创建并训练一个网络
    - 提取网络权重，保存为 para.npz
    - TensorRT 中重建该网络并加载 para.npz 中的权重
    - 生成推理引擎
    - 用引擎做实际推理

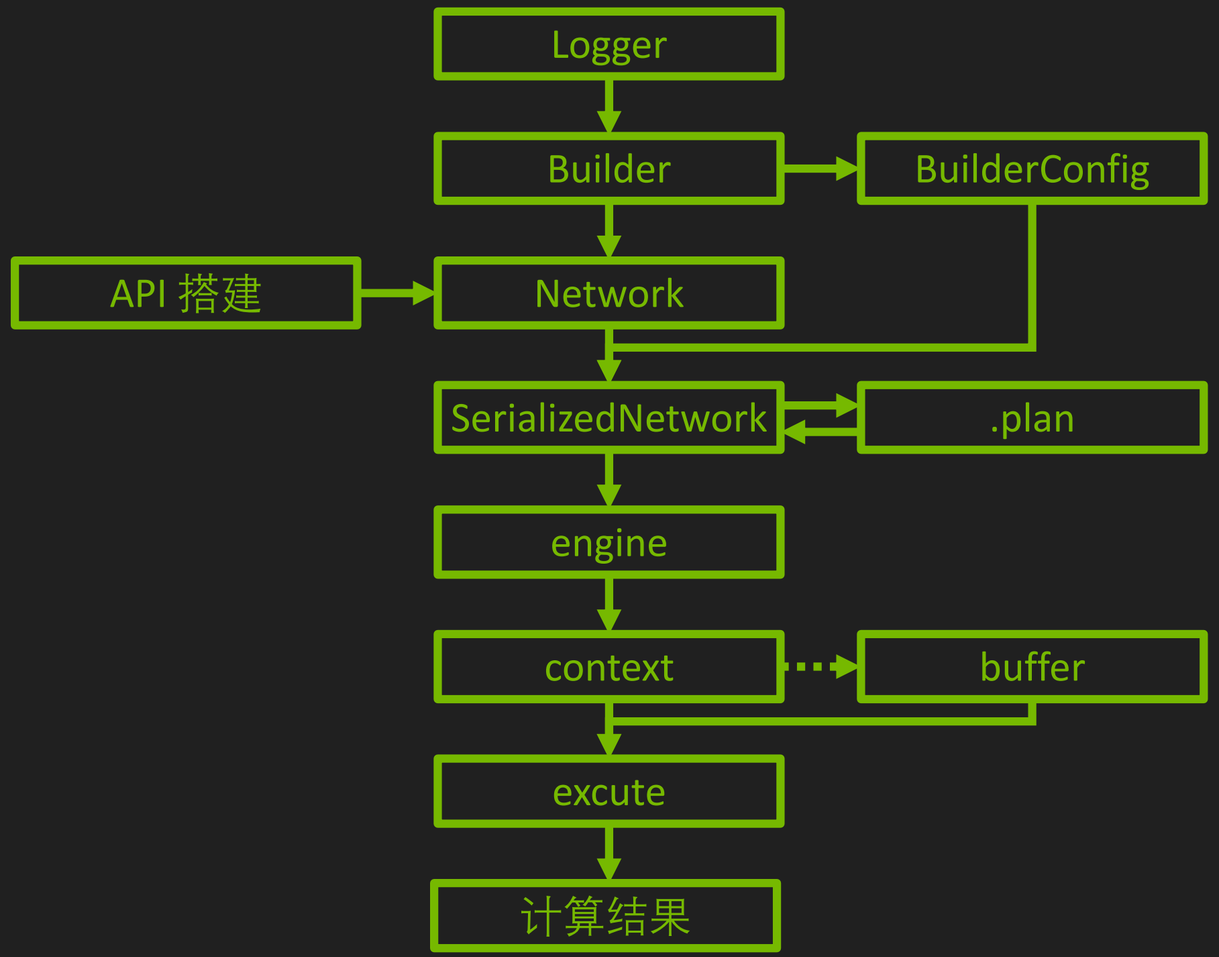
```python
# TensorFlow 中创建网络并保存为 .pb 文件 ------------------------------------
x = tf.compat.v1.placeholder(tf.float32, [None, 28, 28, 1], name='x')

...

tfPara = {}  # 保存权重
for i in tf.compat.v1.get_collection(tf.compat.v1.GraphKeys.GLOBAL_VARIABLES):
    name, value = i.name, sess.run(i)
    #print(name,value.shape)
    tfPara[name] = value
np.savez("paraTF.npz", **tfPara)

# TensorRT 中重建网络并创建 engine ----------------------------------------
logger = trt.Logger(trt.Logger.ERROR)

...

para = np.load(paraFile)

w = para['w1:0'].transpose(3, 2, 0, 1).reshape(-1)
b = para['b1:0']
_0 = network.add_convolution_nd(inputTensor, 32, [5, 5], w, b)
_0.padding_nd = [2, 2]
_1 = network.add_activation(_0.get_output(0), trt.ActivationType.RELU)
_2 = network.add_pooling_nd(_1.get_output(0), trt.PoolingType.MAX, [2, 2])

...

engineString = builder.build_serialized_network(network, config)
engine = trt.Runtime(logger).deserialize_cuda_engine(engineString)
```

• Workflow：使用 TensorRT API 搭建

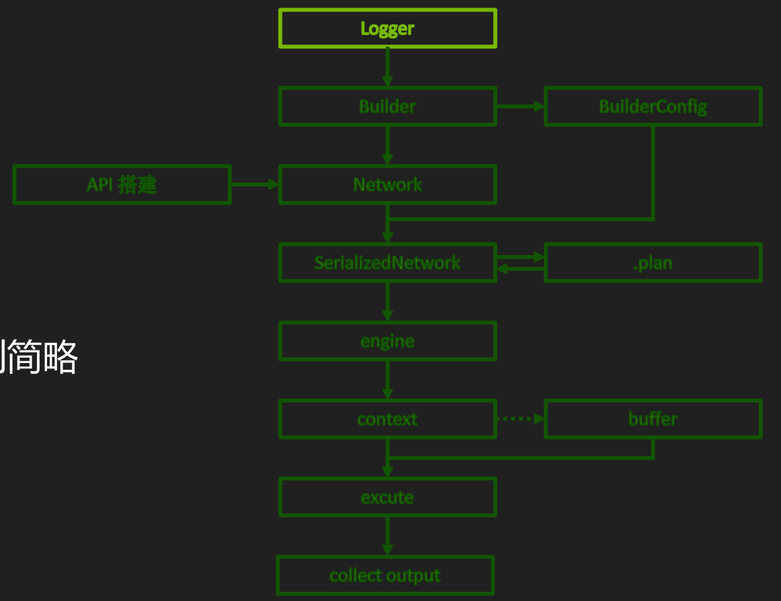
- 问题
  - 怎样从头开始写一个网络?
  - 哪些代码是 API 搭建特有的，哪些是所有 Workflow 通用的?
  - 怎么让一个 Network 跑起来?
  - 用于推理计算的输入输出内存显存怎么准备?
  - 构建引擎需要时间，怎么构建一次，反复使用?
  - TensorRT 的开发环境?

• Workflow：使用 TensorRT API 搭建

- 基本流程
  - 构建阶段
    - 建立 Logger（日志记录器）
    - 建立 Builder（网络元数据）和BuilderConfig（网络元数据的选项）
    - 创建 Network（计算图内容）
    - 生成 SerializedNetwork（网络的 TRT 内部表示）
  - 运行阶段
    - 建立 Engine
    - 创建 Context
    - Buffer 准备（Host 端 + Device 端）
    - Buffer 拷贝 Host to Device
    - 执行推理（Execute）
    - Buffer 拷贝 Device to Host
    - 善后工作



• Workflow：使用 TensorRT API 搭建



- Logger **日志记录器**
  - logger = trt.Logger(trt.Logger.VERBOSE)
  - 可选参数：VERBOSE, INFO, WARNING, ERROR, INTERNAL_ERROR，产生不同等级的日志，由详细到简略
    - VERBOSE：

      [TensorRT] VERBOSE: Graph construction and optimization completed in 0.000261295 seconds.
    - INFO：

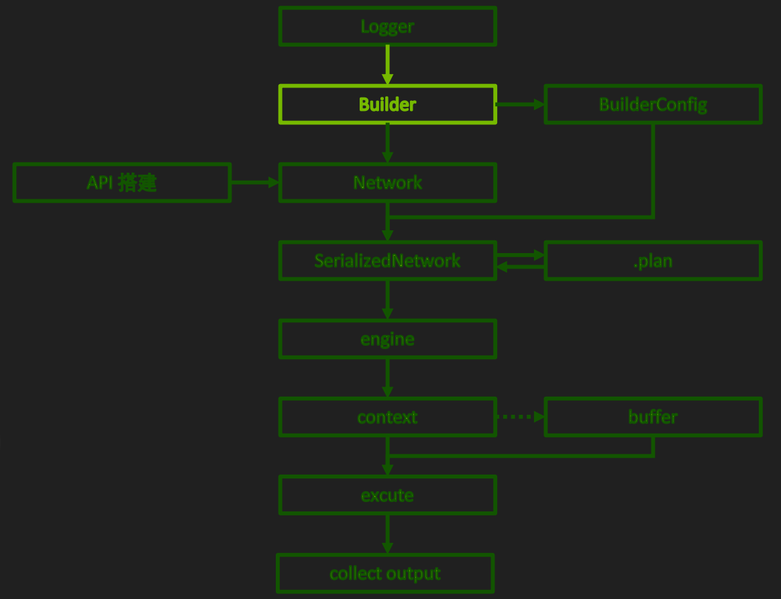
      [TensorRT] INFO: Detected 1 inputs and 1 output network tensors.
    - WARNING：

      [TensorRT] WARNING: Tensor DataType is determined at build time for tensors not marked as input or output.
    - ERROR：

      [TensorRT] ERROR: INVALID_CONFIG: Deserialize the cuda engine failed.
    - INTERNAL_ERROR：

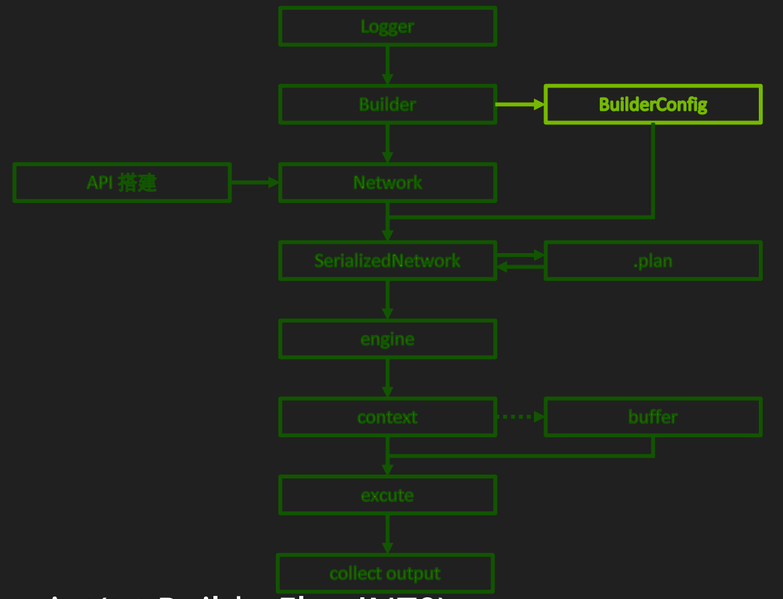
      [TensorRT] ERROR: ../builder/tacticOptimizer.cpp (1820) - TRTInternal Error in computeCosts: 0 (Could not find any implementation for node (Unnamed Layer 0) [TopK].)*

• Workflow：使用 TensorRT API 搭建



- Builder **引擎构建器**
  - builder = trt.Builder(logger)
  - 常用成员（将被废弃，不推荐使用）：
    - builder.max_batch_size = 256　　指定最大 Batch Size（Static Shape 模式下使用）
    - ~~builder.max_workspace_size = 1<<30　　指定最大可用显存（单位 Byte）~~
    - ~~builder.fp16_mode = True/False　　开启/关闭 fp16 模式~~
    - ~~builder.int8_mode = True/False　　开启/关闭 int8 模式~~
    - ~~builder.int8_calibrator = ...　　int8 模式要给定 calibrator~~
    - ~~builder.strict_type_constraints = True/False　　开启/关闭强制精度模式（通常不用）~~
    - ~~builder.refittable = True/False　　开启/关闭 refit 模式~~
  - Dynamic Shape 模式必须用 builderConfig 及相关API

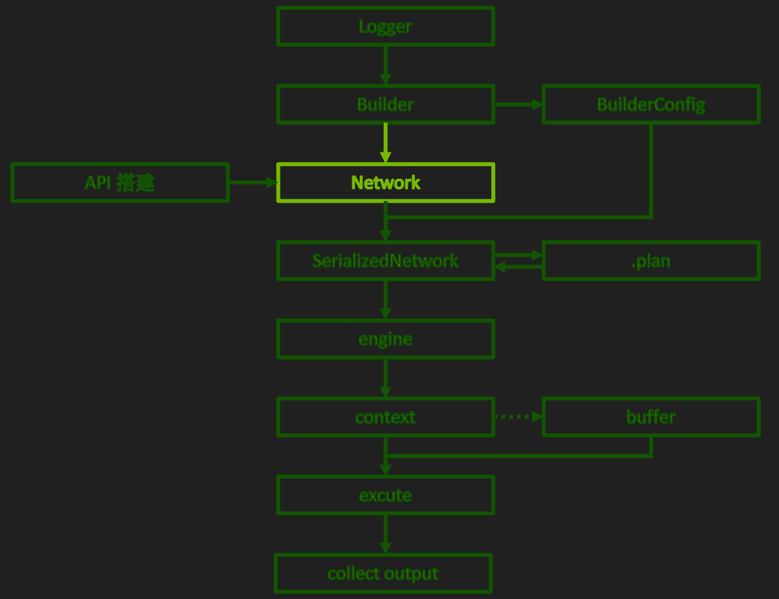
• Workflow：使用 TensorRT API 搭建



- BuilderConfig **网络属性选项**
  - config = builder.create_builder_config()
  - 常用成员：
    - config.max_workspace_size = 1<<30　　指定构建期可用显存（单位：Byte）
    - ~~config.max_batch_size = ...~~　　没有该成员，默认 Explicit batch 模式
    - config.flag = ...　　设置标志位，如 1 << int(trt.BuilderFlag.FP16)，1 << int(trt.BuilderFlag.INT8)，...
    - config.int8_calibrator = ...　　指定 calibrator
    - config. set_tactic_sources / set_timing_cache / algorithm_selector / ...　　更多高级用法（见教程第四部分）

  - Dynamic Shape 模式下使用 builder. max_workspace_size 可能报错：
    - *[TensorRT] IFO: Some tactics do not have sufficient workspace memory to run. Increasing workspace size may increase performance, please check verbose output.*
    - *[TensorRT] ERROR: Try increasing the workspace size with IBuilderConfig::setMaxWorkspaceSize() if using IBuilder::buildEngineWithConfig, or IBuilder::setMaxWorkspaceSize() if using IBuilder::buildCudaEngine.*
    - *[TensorRT] ERROR: ../builder/tacticOptimizer.cpp (1820) - TRTInternal Error in computeCosts: 0 (Could not find any implementation for node (Unnamed Layer* 0) [TopK].)*

• Workflow：使用 TensorRT API 搭建



- Network **网络具体构造**
  - network = builder.create_network()
  - 常用参数：
    - 1 << int(tensorrt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH)，使用 Explicit Batch 模式
  - 常用方法：
    - network.add_input('oneTensor' ,trt.float32, (3,4,5))　　标记网络输入张量
    - convLayer = network.add_convolution_nd(XXX)　　添加各种网络层
    - network.mark_output(convLayer.get_output(0))　　标记网络输出张量
  - 常用获取网络信息的成员：
    - network.name / network.num_layers / network.num_inputs / network.num_outputs
    - network.has_implicit_batch_dimension / network.has_explicit_precision

• Workflow：使用 TensorRT API 搭建

- Explicit Batch 模式 v.s. Implicit Batch 模式
  - TensorRT 主流 Network 构建方法，Implicit Batch 模式 builder.create_network(0) 仅用作后向兼容
  - 所有张量显式包含 Batch 维度、比 Implicit Batch 模式多一维
  - 需要使用 builder.create_network(1 << int(tensorrt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
  - Explicit Batch 模式能做、Implicit Batch 模式不能做的事举例：
    - Layer Normalization
    - Reshape/Transpose/Reduce over batch dimension
    - Dynamic shape 模式
    - Loop 结构
    - 一些 Layer 的高级用法（如 ShufleLayer.set_input）
  - 从 Onnx 导入的模型也默认使用 Explicit Batch 模式
  - 范例代码仅在 01-SimpleDemo 中 TensorRT6 和 TensorRT7 中保留了 Implicit Batch 的例子

• Workflow：使用 TensorRT API 搭建

➢ Dynamic Shape 模式

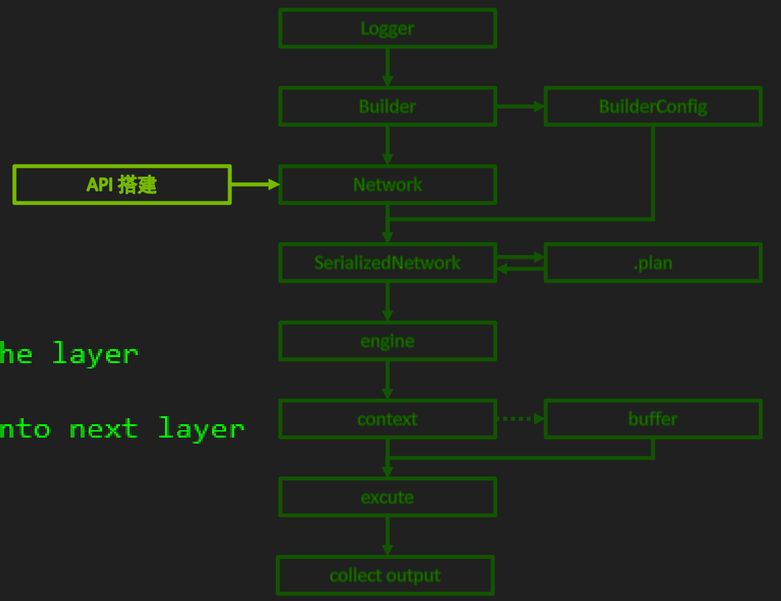
- ➢ 适用于输入张量形状在推理时才决定网络
- ➢ 除了 Batch 维，其他维度也可以推理时才决定
- ➢ 需要 Explicit Batch 模式
- ➢ 需要 Optimazation Profile 帮助网络优化
- ➢ 需用 context.set_binding_shape 绑定实际输入数据形状

```
network  = builder.create_network(1<<int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
profile  = builder.create_optimization_profile()
config   = builder.create_builder_config()

inputTensor     = network.add_input('inputT0', trt.DataType.FLOAT, [-1,-1,-1])
profile.set_shape(inputTensor.name, (1,1,1),(3,4,5),(6,8,10))
config.add_optimization_profile(profile)

# ...

context.set_binding_shape(0,[3,4,5])

# ...

outputH0    = np.empty(context.get_binding_shape(1),dtype = trt.nptype(engine.get_binding_dtype(1)))

# ...

context.execute_async_v2([int(inputD0), int(outputD0)], stream)
```

➢ Profile **指定输入张量大小范围**

- ➢ profile = builder.create_optimization_profile()
- ➢ 常用方法：
  - ➢ profile.set_shape(tensorName, minShape, commonShape, maxShape)　　给定输入张量的最小、最常见、最大尺寸
  - ➢ config.add_optimization_profile(profile)　　将设置的 profile 传递给 config 以创建网络

• Workflow：使用 TensorRT API 搭建



- Layer 和 Tensor
  - 注意区别 Layer 和 Tensor

```
oneLayer  = network.add_identity(inputTensor) # output is a layer
oneTensor = oneLayer.get_output(0)            # get tensor from the layer
nextLayer = network.add_identity(oneTensor)   # take the tensor into next layer
```

  - Layer 的常用成员和方法：
    - oneLayer.name = 'one'　　获取或指定 Layer 的名字
    - oneLayer.type　　获取该层的种类
    - oneLayer.precision　　指定改层计算精度（需配合 builder.strict_type_constraints）
    - oneLayer.get_output(i)　　获取该层第 i 个输出张量
  - Tensor 的常用成员和方法：
    - oneTensor.name = 'one'　　获取或指定 tensor 的名字
    - oneTensor.shape　　获取 tensor 的形状，可用于 print 检查或作为后续层的参数
    - oneTensor.dtype　　获取或设定 tensor 的数据类型（可用于配合 identity 层实现数据类型转换）

• Workflow：使用 TensorRT API 搭建

- 从 Network 中打印所有层和张量的信息
  - 范例代码：02-API/PrintLayerInfo
    - 外层循环遍历所有 Layer
    - 内层循环遍历该 Layer 的的所有input/output

```python
for i in range(network.num_layers):
    layer = network.get_layer(i)
    print(i,"%s,in=%d,out=%d,%s"%(str(layer.type)[10:],layer.num_inputs,layer.num_outputs,layer.name))
    for j in range(layer.num_inputs):
        tensor  =layer.get_input(j)
        if tensor == None:
            print("\tInput  %2d:"%j,"None")
        else:
            print("\tInput  %2d:%s,%s,%s"%(j,tensor.shape,str(tensor.dtype)[9:],tensor.name))
    for j in range(layer.num_outputs):
        tensor  =layer.get_output(j)
        if tensor == None:
            print("\tOutput %2d:"%j,"None")
        else:
            print("\tOutput %2d:%s,%s,%s"%(j,tensor.shape,str(tensor.dtype)[9:],tensor.name))
```

```
0 IDENTITY,in=1,out=1,node_of_127
    Input   0:(1,),INT32,1
    Output  0:(1,),INT32,127
1 CONSTANT,in=0,out=1,129
    Output  0:(),INT32,(Unnamed Layer* 1) [Constant]_output
2 SHUFFLE,in=1,out=1,node_of_131
    Input   0:(),INT32,(Unnamed Layer* 1) [Constant]_output
    Output  0:(1,),INT32,131

  ...

600 SHUFFLE,in=1,out=1,(Unnamed Layer* 600) [Shuffle]
    Input   0:(128,),FLOAT,(Unnamed Layer* 599) [Constant]_output
    Output  0:(1, 1, 128),FLOAT,(Unnamed Layer* 600) [Shuffle]_output
601 ELEMENTWISE,in=2,out=1,node_of_586
    Input   0:(16, 255, 128),FLOAT,585
    Input   1:(1, 1, 128),FLOAT,(Unnamed Layer* 600) [Shuffle]_output
    Output  0:(16, 255, 128),FLOAT,586
602 IDENTITY,in=1,out=1,2006-CastForOutput
    Input   0:(1, 1, 255),INT32,176
    Output  0:(1, 1, 255),INT32,2005-CastForOutput
```

• Workflow：使用 TensorRT API 搭建

- 权重迁移
  - 原模型中权重保存为 npz，方便 TensorRT 读取
- 范例代码
  - 03-APIModel/TensorFlow 和 03-APIModel/pyTorch 等
- 逐层搭建
  - 注意算法一致性和权重的排列方式
  - 举例：TensorFlow 中 LSTM 多种实现，各种实现导出权重的排列顺序不同
- 逐层检验输出
  - FP32 模式相对误差均值 1×10^{-6} 量级，FP16 模式相对误差均值 1×10^{-3} 量级
  - 保证 FP32 模式结果正确后，逐步尝试 FP16 和 INT8 模式

```python
# TensorFlow ------------------------------------------------------------------
para = {}
for i in tf.compat.v1.get_collection(tf.compat.v1.GraphKeys.GLOBAL_VARIABLES):
    name,value = i.name,sess.run(i)
    tfPara[name] = value
np.savez("paraTF.npz",**para)


# pyTorch ---------------------------------------------------------------------
para = {}
for name,para in model.named_parameters():
    para[name] = para
np.savez("paraPT.npz",**para)


# Paddlepaddle ----------------------------------------------------------------
file_path = "../model/model_state"
program_state = fluid.load_program_state(file_path + "/temp")
fluid.set_program_state(prog, program_state)
para = {}
for block in prog.blocks:
    for var in block.vars:
        if 'tmp' in var:
            continue
        param = block.vars[var].name
        para[param] = np.array( fluid.global_scope().find_var(param).get_tensor() )
np.savez('./paraPP.npz',**para)
```

• Workflow：使用 TensorRT API 搭建

- 常见 Layer 的使用范例
  - 范例代码 02-API/01-Layer/*.md，32 种 Layer 示例 + Condition 结构示例+ Loop 结构示例
  - 各 .md 中初始示例代码可以拷进 test.py 中，然后直接运行 python test.py
  - 各 Layer 的各参数的示例代码，可以拷进 test.py 中覆盖初始示例代码“# ---- 替换部分”之间的部分，然后运行

- 遇到 TensorRT 不原生支持的节点
  - 自己实现 Plugin（见教程第三部分讲）

• Workflow：使用 TensorRT API 搭建

- FP16 模式
  - 范例代码：04-Parser/TensorFlow-ONNX-TensorRT
  - config.flags = 1<<int(trt.BuilderFlag.FP16)
  - 建立 engine 时间比 FP32 模式更长（可能在各原节点间插入 Reformat 节点）
  - Timeline 中出现 nchwToNchw 等 kernel 调用
  - 部分层可能精度下降导致较大误差
    - 找到误差较大的层（用 polygraphy等工具，见教程第二部分 polygraphy）
    - 强制该层使用 FP32 进行计算
      - config.flags = 1<<int(trt.BuilderFlag. STRICT_TYPES)
      - layer.precision = trt.float32

• Workflow：使用 TensorRT API 搭建

- Int8 模式 —— PTQ
  - 范例代码：04-Parser/pyTorch-ONNX-TensorRT
  - 需要有校准集（输入范例数据）
  - 自己实现 calibrator（如右图）
  - config.flags = 1<<int(trt.BuilderFlag.INT8)
  - config.int8_calibrator = ...

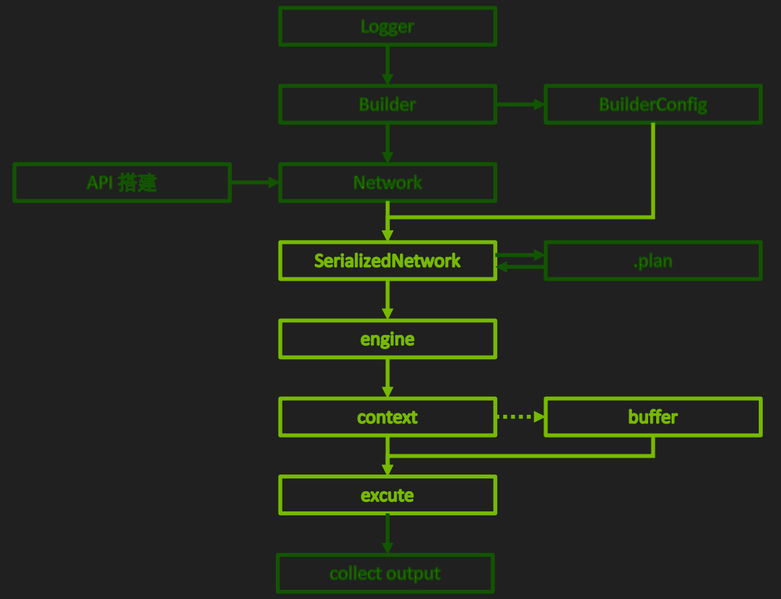
- Int8 模式 —— QAT
  - 范例代码：02-API/Int8-QAT，04-Parser/pyTorch-ONNX-TensorRT-QAT
  - config.flags = 1<<int(trt.BuilderFlag.INT8)
  - 在 pyTorch 网络中插入 Quantize/Dequantize 层

```python
class MyCalibrator(trt.IInt8EntropyCalibrator2):

    def __init__(self, calibrationDataPath, calibrationCount, inputShape, cacheFile):
        trt.IInt8EntropyCalibrator2.__init__(self)
        ...
        self.oneBatch = self.batchGenerator()

    def __del__(self):
        cudart.cudaFree(self.dIn)

    def batchGenerator(self):
        yield np.ascontiguousarray(imageList(subImageList))

    def get_batch_size(self):  # do NOT change name
        return self.shape[0]

    def get_batch(self, nameList=None, inputNodeName=None):  # do NOT change name
        try:
            data = next(self.oneBatch)
            cudart.cudaMemcpy(self.dIn, data.ctypes.data, self.buffeSize,
                              cudart.cudaMemcpyKind.cudaMemcpyHostToDevice)
            return [int(self.dIn)]
        except StopIteration:
            return None

    def read_calibration_cache(self):  # do NOT change name
        if os.path.exists(self.cacheFile):
            print("Succeed finding cahce file: %s" % (self.cacheFile))
            with open(self.cacheFile, "rb") as f:
                cache = f.read()
                return cache
        else:
            print("Failed finding int8 cache!")
            return

    def write_calibration_cache(self, cache):  # do NOT change name
        with open(self.cacheFile, "wb") as f:
            f.write(cache)
        print("Succeed saving int8 cache!")
```

• Workflow：使用 TensorRT API 搭建

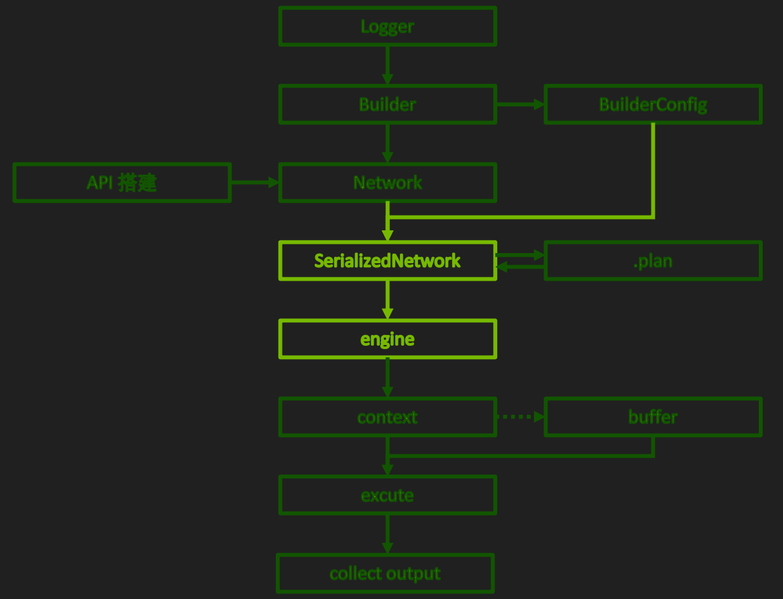
- 问题
  - 怎样从头开始写一个可以跑的网络？
  - API搭建比较复杂，那么哪些代码是API搭建特有的，哪些是所有 Workflow 通用的？
  - 怎么让一个 Network 跑起来？
  - 用于推理计算的输入输出数据怎么准备？
  - 构建引擎需要时间，怎么构建一次，反复使用？
  - TensorRT 的开发环境？

TensorRT 运行期（Runtime）



- 生成 TRT 内部表示
  - serializedNetwork = builder. build_serialized_network(network, config)
- 生成 Engine
  - engine = trt.Runtime(logger).deserialize_cuda_engine(serializedNetwork)
- 创建 Context
  - context = engine.create_execution_context()
- 绑定输入输出（Dynamic Shape 模式必须）
  - context.set_binding_shape(0,[1,1,28,28])
- 准备 Buffer
  - inputHost = np.ascontiguousarray(inputData.reshape(-1))
  - outputHost = np.empty(context.get_binding_shape(1), trt.nptype(engine.get_binding_dtype(1)))
  - inputDevice = cudart.cudaMalloc(inputHost.nbytes)[1]
  - outputDevice = cudart.cudaMalloc(outputHost.nbytes)[1]
- 执行计算
  - cudart.cudaMemcpy(inputDevice, inputHost.ctypes.data, inputHost.nbytes, cudart.cudaMemcpyKind.cudaMemcpyHostToDevice)
  - context.execute_v2([int(inputDevice), int(outputDevice)])
  - cudart.cudaMemcpy(outputHost.ctypes.data, outputDevice, outputHost.nbytes, cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost)

• Workflow：使用 TensorRT API 搭建



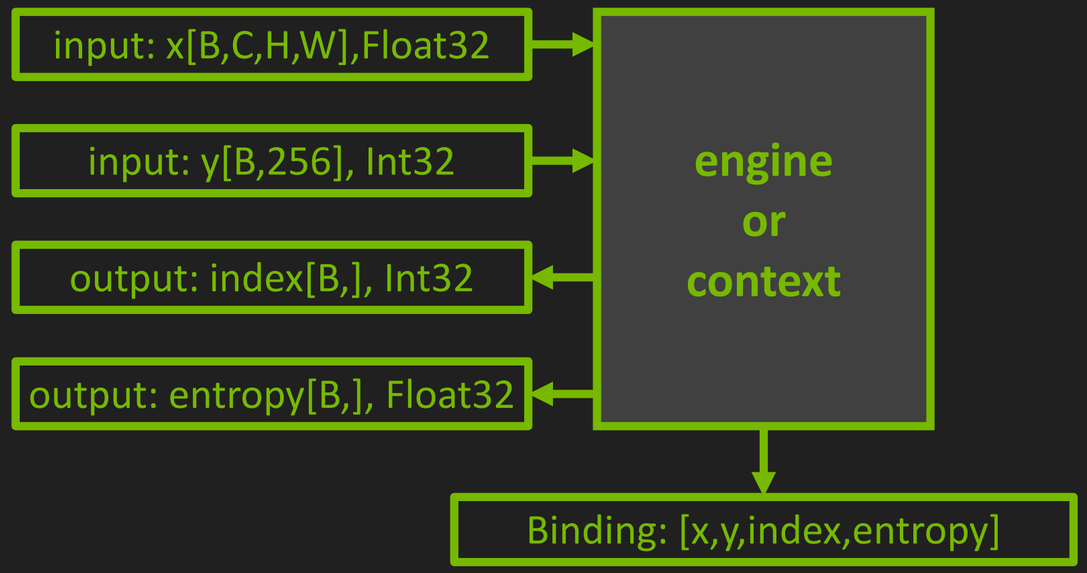
- Engine **计算引擎**
  - serializedNetwork = builder. build_serialized_network(network, config)
  - engine = trt.Runtime(logger).deserialize_cuda_engine(serializedNetwork)
  - 常用成员：
    - engine.num_bindings　　获取 engine 绑定的输入输出张量总数，n + m
    - engine.max_batch_size　　获取 engine 的最大 batch size，Explicit Batch 模式下为 1
    - engine.num_layers　　获取 engine（自动优化后）总层数
  - 常用方法：
    - engine.get_binding_dtype(i)　　第 i 个绑定张量的数据类型，索引 0~n-1 为输入张量，n~n+m-1 为输出张量
    - engine.get_binding_shape(i)　　第 i 个绑定张量的张量形状，Dynamic Shape 模式下结果可能含 -1
    - engine. binding_is_input(i)　　第 i 个绑定张量是否为输入张量
    - engine.get_binding_index('n')　　名字叫 'n' 的张量在 engine 中的绑定索引

• Workflow：使用 TensorRT API 搭建

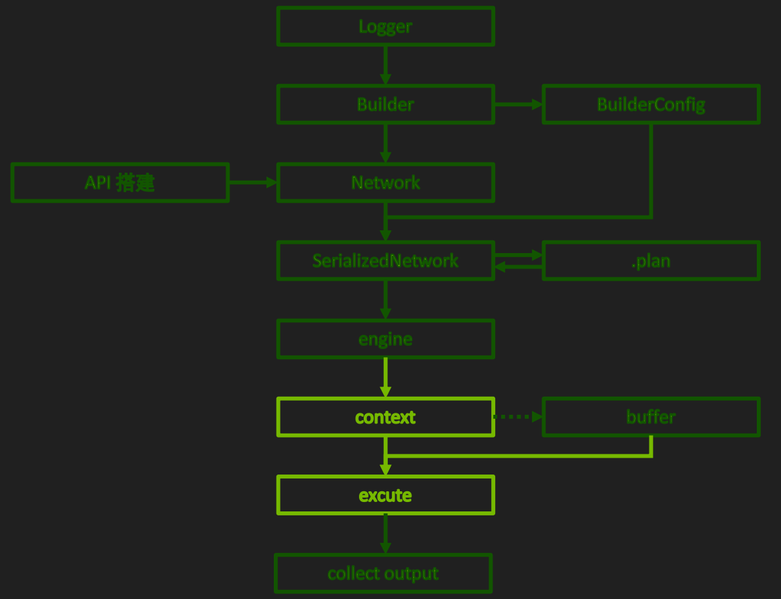
- 什么是 Binding?
  - engine / context 给所有输入输出张量安排了位置
  - 总共 engine.num_bindings 个 binding，输入张量排在最前，输出张量排在最后

  - 如右图，假设模型输入两个张量 x 和 y、输出两个张量 index 和 entropy
  - 则 engine 和 context 会按照这四个张量在计算图中的拓扑顺序给一个Binding
  - 运行期绑定张量形状时，要按指定位置绑定
    - context.set_binding_shape(0,[4,1,28,28])
    - context.set_binding_shape(1,[4,256])
    - 此时输出张量形状会自动计算，从 (-1,) 和 (-1,) 变成 (4,) 和 (4,)

  - 多 Profile 功能中 Binding 规则会变复杂一些（见教程第四部分）



• Workflow：使用 TensorRT API 搭建



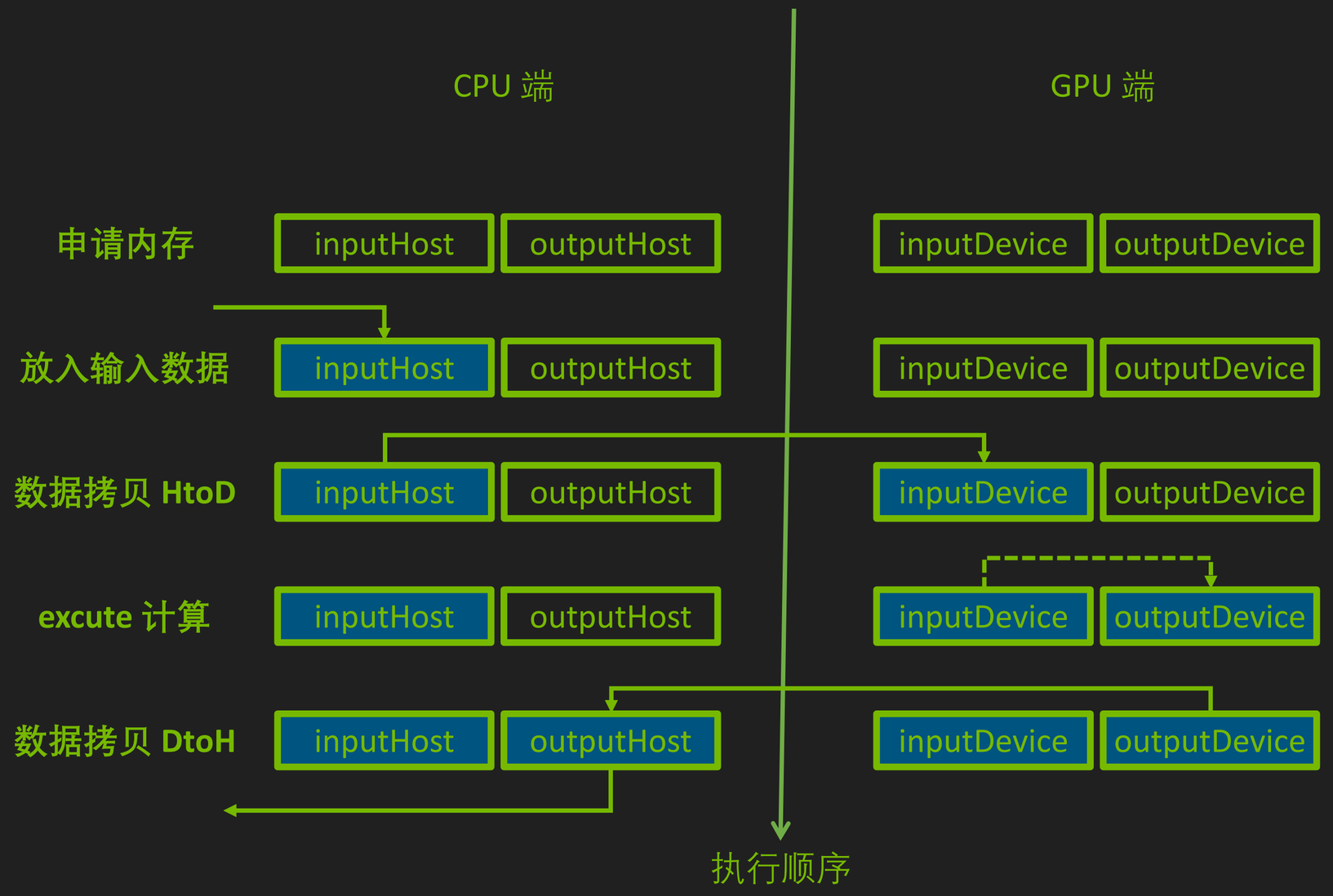
- Context **推理进程**
  - context = engine.create_execution_context()
  - 常用成员：
    - context.all_binding_shapes_specified　　确认所有绑定的输入输出张量形状均被指定（dynamic shape 模式中很重要）
  - 常用方法：
    - context.set_binding_shape(i, shapeOfInputTensor)　　设定第 i 个绑定张量的形状（Dynamic Shape 模式中使用）
    - context.get_binding_shape(i)　　获取第 i 个绑定张量的形状
    - context.execute_v2(listOfBuffer)　　Explicit batch 模式的同步执行
    - context.execute_async_v2(listOfBuffer, srteam)　　Explicit batch 模式的异步执行

• Workflow：使用 TensorRT API 搭建

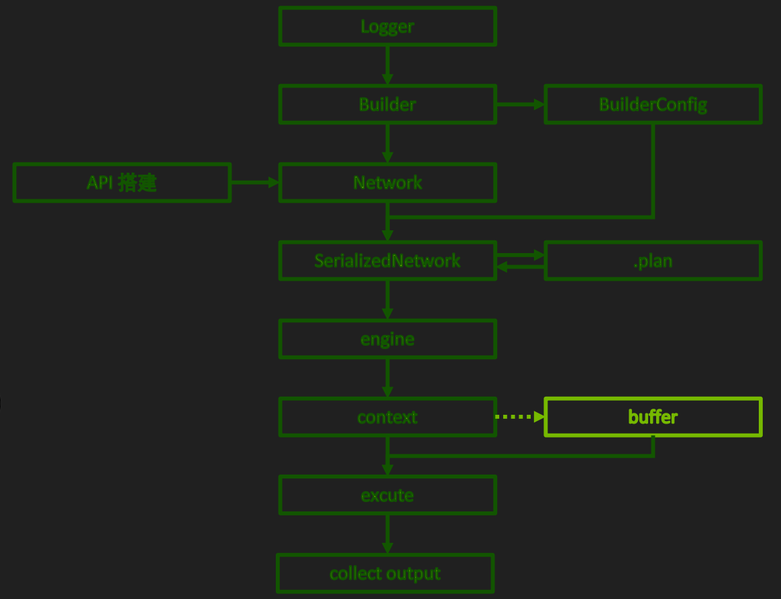
- 问题
  - 怎样从头开始写一个可以跑的网络?
  - API搭建比较复杂，那么哪些代码是API搭建特有的，哪些是所有 Workflow 通用的?
  - 怎么让一个 Network 跑起来?
  - 用于推理计算的输入输出数据怎么准备?
  - 构建引擎需要时间，怎么构建一次，反复使用?
  - TensorRT 的开发环境?

• Workflow：使用 TensorRT API 搭建

- CUDA 异构计算
  - 同时准备 CPU 端内存和 GPU端显存
  - 开始计算前把数据从内存拷贝到显存中
  - 计算过程的输入输出数据均在 GPU端读写
  - 计算完成后要把结果拷贝会内存才能使用



• Workflow：使用 TensorRT API 搭建



- Buffer
  - 内存和显存的申请
  - inputHost = np.ascontiguousarray(inputData.reshape(-1))
  - outputHost = np.empty(context.get_binding_shape(1), trt.nptype(engine.get_binding_dtype(1)))
  - inputDevice = cudart.cudaMalloc(inputHost.nbytes)[1]
  - outputDevice = cudart.cudaMalloc(outputHost.nbytes)[1]

  - 内存和显存之间的拷贝
  - cudart.cudaMemcpy(inputDevice, inputHost.ctypes.data, inputHost.nbytes, cudart.cudaMemcpyKind.cudaMemcpyHostToDevice)
  - cudart.cudaMemcpy(outputHost.ctypes.data, outputDevice, outputHost.nbytes, cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost)

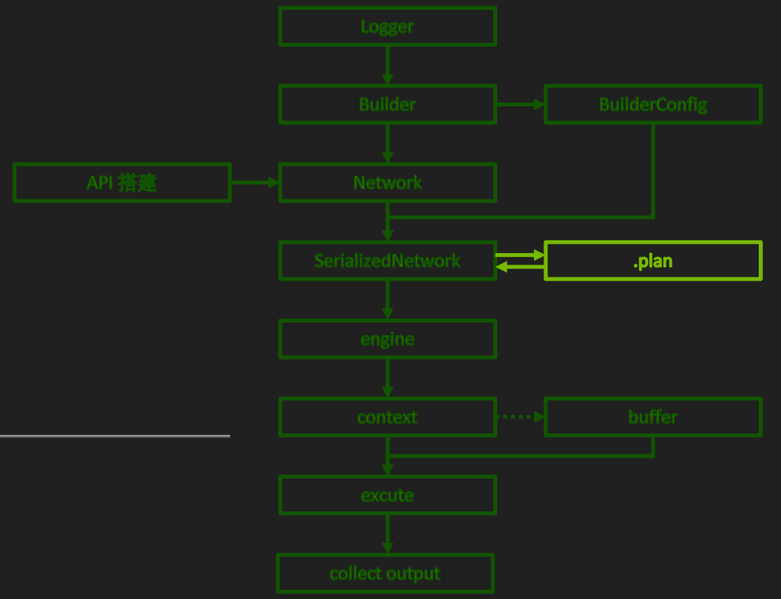
  - 推理完成后释放显存
  - cudart.cudaFree(inputDevice)
  - cudart.cudaFree(outputDevice)

- 高级话题：利用 CUDA stream 将 buffer 申请、拷贝做成异步操作（见教程第四部分）

• Workflow：使用 TensorRT API 搭建

- 问题
  - 怎样从头开始写一个可以跑的网络？
  - API搭建比较复杂，那么哪些代码是API搭建特有的，哪些是所有 Workflow 通用的？
  - 怎么让一个 Network 跑起来？
  - 用于推理计算的输入输出数据怎么准备？
  - 构建引擎需要时间，怎么构建一次，反复使用？
  - TensorRT 的开发环境？

Workflow：使用 TensorRT API 搭建



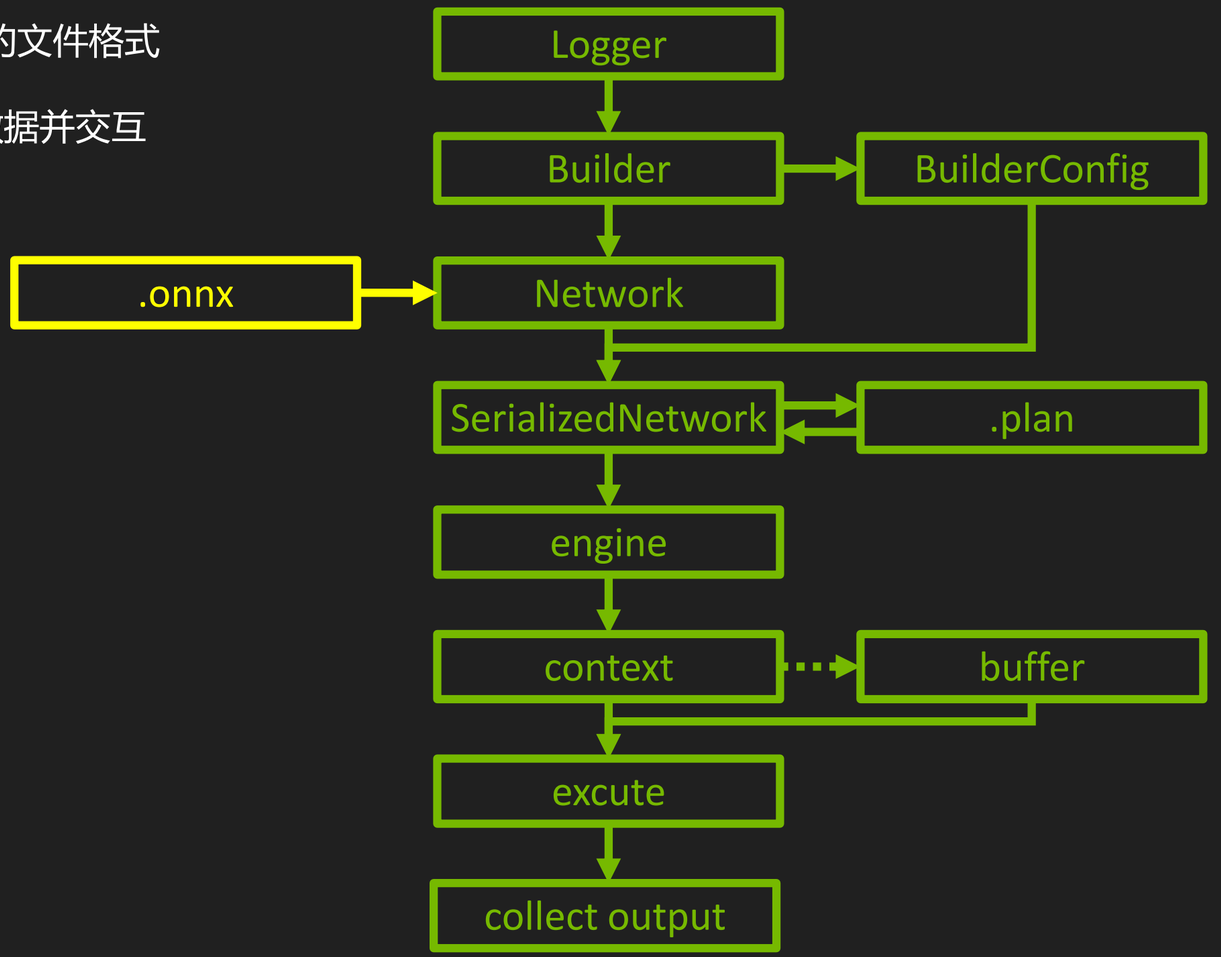
- 序列化与反序列化
  - 将 SerializedNetwork 保存为文件，下次跳过构建直接使用
  - 注意环境统一（硬件环境 + CUDA/cuDNN/TensorRT 环境）
    - Engine 包含硬件有关优化，不能跨硬件平台使用
    - 不同版本 TensorRT 生成的 engine 不能相互兼容
    - 同平台同环境多次生成的 engine 可能不同

```python
if os.path.isfile(trtFile):
    with open(trtFile, 'rb') as f:
        serializedNetwork = f.read()
else:

    ... # 构建网络

    serializedNetwork = builder.build_serialized_network(network, config)
    with open(trtFile, 'wb') as f:
        f.write(serializedNetwork)
        print("Succeeded saving .plan file!")

engine = trt.Runtime(logger).deserialize_cuda_engine(serializedNetwork)
```

  - TensorRT runtime 版本与 engine 版本不同时的报错信息
    - *[TensorRT] ERROR: INVALID_CONFIG: The engine plan file is not compatible with this version of TensorRT, expecting library version 7.2.3 got 7.2.2, please rebuild.*
    - *[TensorRT] ERROR: engine.cpp (1646) - Serialization Error in deserialize: 0 (Core engine deserialization failure)*
  - 高级话题：利用 AlgorthimSelector 或 TimingCache 多次生成一模一样的 engine（见教程第四部分）

• Workflow：使用 Parser

- ONNX
  - 简介 https://onnx.ai/
  - Open Neural Network Exchange，针对机器学习所设计的开放式的文件格式
  - 用于存储训练好的模型，使得不同框架可以采用相同格式存储模型数据并交互
  - TensorFlow / pyTorch 模型转 TensorRT 的中间表示
  - 当前 TensoRT 导入模型的主要途径
- Onnxruntime
  - 简介 https://onnxruntime.ai/
  - 利用 onnx 格式模型进行推理计算的框架
  - 兼容多硬件、多操作系统，支持多深度学习框架
  - 可用于检查 TensorFlow / Torch 模型导出到 Onnx 的正确性



• Workflow：使用 Parser

- pyTorch 的例子
- 范例代码 04-Parser/02-pyTorch-ONNX-TensorRT

- 基本流程：
  - pyTorch 中创建网络并保存为 .pt 文件
  - 使用 pyTorch 内部 API 将 .pt 转化为 .onnx
  - TensorRT 中读取 .onnx 构建 engine 并做推理

- 本例在 TensorRT 中开启了 Int8 模式
  - 需要自己实现 calibrator 类（calibrator.py 可作为 Int8 通用样例）

```python
# pyTorch 中创建网络并保存为 .pt 文件 ------------------------------------------------
# ...

t.save(net, ptFile)
print("Succeeded building model in pyTorch!")

# 将 .pt 文件转换为 .onnx 文件 ------------------------------------------------------
t.onnx.export(net,
                t.randn(1, 1, imageHeight, imageWidth, device="cuda"),
                "./model.onnx",
                example_outputs=[t.randn(1,10,device="cuda"),t.randn(1,device="cuda")],
                input_names=['x'],
                output_names=['y','z'],
                do_constant_folding=True,
                verbose=True,
                keep_initializers_as_inputs=True,
                opset_version=12,
                dynamic_axes={"x":{0:"nBatchSize"},"z":{0: "nBatchSize"}})
print("Succeeded converting model into onnx!")

# TensorRT 中加载 .onnx 创建 engine -------------------------------------------------
logger = trt.Logger(trt.Logger.ERROR)

# ...

    # 关键部分，用 Parser 加载 .onnx
    with open(onnxFile, 'rb') as model:
        if not parser.parse( model.read() ):
            print ("Failed parsing ONNX file!")
            for error in range(parser.num_errors):
                print (parser.get_error(error))
            exit()
        print ("Succeeded parsing ONNX file!")

# ...

# 准备 TensorRT runtime 和 buffer，并进行推理
context = engine.create_execution_context()

#...

print("Succeeded running model in TensorRT!")
```

• Workflow：使用 Parser

- TensorFlow 的例子
- 范例代码 04-Parser/01-TensorFlow-ONNX-TensorRT

- 基本流程：
  - TensorFlow 中创建网络并保存为 .pb 文件
  - 使用 tf2onnx 将 .pb 转化为 .onnx
  - TensorRT 中读取 .onnx构建 engine 并做推理

- 本例在 TensorRT 中开启了 FP16 模式

```python
# TensorFlow 中创建网络并保存为 .pb 文件 ------------------------------------------------------------
x   = tf.compat.v1.placeholder(tf.float32, [None,28,28,1], name='x')

# ...

# 保存模型为 .pb
constantGraph = tf.graph_util.convert_variables_to_constants(sess, sess.graph_def,['z'])
with tf.gfile.FastGFile("./model.pb", mode='wb') as f:
    f.write(constantGraph.SerializeToString())
sess.close()
print("Succeeded building model in TensorFlow!")

# 将 .pb 文件转换为 .onnx 文件 ----------------------------------------------------------------------
os.system("python -m tf2onnx.convert --input %s --output %s --inputs 'x:0' --outputs 'z:0'"%(pbFile,onnxFile))
print("Succeeded converting model into onnx!")

# TensorRT 中加载 .onnx 创建 engine ----------------------------------------------------------------
logger = trt.Logger(trt.Logger.ERROR)

# ...

    # 关键部分，用 Parser 加载 .onnx
    with open(onnxFile, 'rb') as model:
        if not parser.parse( model.read() ):
            print ("Failed parsing ONNX file!")
            for error in range(parser.num_errors):
                print (parser.get_error(error))
            exit()
        print ("Succeeded parsing ONNX file!")

# ...

# 准备 TensorRT runtime 和 buffer，并进行推理
context = engine.create_execution_context()

# ...
print("Succeeded running model in TensorRT!")
```

• Workflow：使用 Parser

- 如果不用 Int8 模式，onnx parser 的代码几乎是通用的
  - 有命令行工具可用，基本等价于脚本中的 API（见教程第二部分 trtexec）
- 遇到 TensorRT 不支持的节点
  - 修改源模型
  - 修改 Onnx 计算图（见教程第二部分 onnx-surgeon）
  - TensorRT 中实现 Plugin（见教程第三部分）
  - 修改 Parser：修改 TRT 源码并重新编译 TRT（见 https://github.com/NVIDIA/TensorRT 项目）
- trtexec，onnx-graphsurgeon 和 plugin 都是使用 parser 的必备知识，将在后面的教程展开

• Workflow：使用框架内 TensorRT 接口

- TF-TRT
  - 范例代码 07-FrameworkTRT/TFTRT
- Torch-TensorRT（旧名 TRTorch）
  - 范例代码 07-FrameworkTRT/Torch-TensorRT

• Workflow：使用 TensorRT API 搭建

- 问题
  - 怎样从头开始写一个可以跑的网络?
  - API搭建比较复杂，那么哪些代码是API搭建特有的，哪些是所有 Workflow 通用的?
  - 怎么让一个 Network 跑起来?
  - 用于推理计算的输入输出数据怎么准备?
  - 构建引擎需要时间，怎么构建一次，反复使用?
  - TensorRT 的开发环境?

TensorRT 环境

- nvidia-docker
  - 安装步骤：https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html#docker
  - 镜像列表：https://docs.nvidia.com/deeplearning/frameworks/support-matrix/index.html
  - 推荐镜像：nvcr.io/nvidia/tensorrt:21.12-py3，tensorflow:21.12-tf1-py3， tensorflow:21.12-tf2-py3，nvcr.io/nvidia/pytorch:21.12-py3
- python 库：
  - nvidia-pyindex，cuda-python（python≥3.7），pycuda，onnx，onnx-surgeon，onnxruntime-gpu，opencv-python，polygraphy
- 推荐使用：
  - 最新的 TensorRT8，图优化内容更多，优化过程和推理过程显存使用量更少
  - builderConfig API，功能覆盖旧版 builder API，旧版 builder API 将被废弃
  - explicit batch 模式，ONNX 格式默认模式，灵活性和应用场景更广
  - dynamic shape 模式，使模型适应性更好
  - cuda-python 库，完整的 CUDA API 支持，修复 pycuda 库可能存在的问题（如与其他框架交互使用时的同步操作等）

Part 2

• 开发辅助工具

➢ trtexec	TensorRT 命令行工具，主要的 End2End 性能测试工具
➢ Netron	网络可视化
➢ onnx-graphsurgeon	onnx 计算图编辑
➢ polygraphy	结果验证与定位，图优化
➢ Nsight Systems	性能分析

开发辅助工具

- 希望解决的问题
  - 不想写脚本来跑 TensorRT — 用命令行搞定
  - 怎么进行简单的推理性能测试? — 测量延迟、吞吐量等
  - 网络结构可视化?
  - 计算图上有些节点阻碍 TensorRT 自动优化 — 手工调整以便 TensorRT 能够处理
  - 怎么处理 TensorRT 不支持的网络结构? — 手工调整以便 TensorRT 能够处理
  - 怎么检验 TensorRT 上计算结果正确性 / 精度? — 同时在原框架和 TensorRT 上运行
  - 怎么找出计算错误 / 精度不足的层? — 模型逐层比较
  - 怎么进行简单的计算图优化? — 手工调整
  - 怎样找出最耗时的层? — 找到热点集中优化

• trtexec

- TensorRT 的命令行工具，
  - 随 TensorRT 安装，位于 tensorrt-XX/bin/trtexec
- 功能
  - 由 .onnx 模型文件生成 TensorRT 引擎并序列化为 .plan
  - 查看 .onnx 或 .plan 文件的网络逐层信息
  - 模型性能测试（测试TensorRT 引擎基于随机输入或给定输入下的性能）
- 范例代码
  - 08-Tool/trtexec，运行 ./command.sh

```
clear

rm ./*.pb ./*.onnx ./*.plan ./result-*.txt

# 从 TensorFlow 创建一个 .onnx 用来做 trtexec 的输入文件
python3 getOnnxModel.py

# 用上面的 .onnx 构建一个 TensorRT 引擎并作推理
trtexec \
    --onnx=model.onnx \
    --minShapes=x:0:1x1x28x28 \
    --optShapes=x:0:4x1x28x28 \
    --maxShapes=x:0:16x1x28x28 \
    --workspace=1024 \
    --saveEngine=model-FP32.plan \
    --shapes=x:0:4x1x28x28 \
    --verbose \
    > result-FP32.txt

# 用上面的 .onnx 构建一个 TensorRT 引擎并作推理，使用 FP16 模式
trtexec \
    --onnx=model.onnx \
    --minShapes=x:0:1x1x28x28 \
    --optShapes=x:0:4x1x28x28 \
    --maxShapes=x:0:16x1x28x28 \
    --workspace=1024 \
    --saveEngine=model-FP16.plan \
    --shapes=x:0:4x1x28x28 \
    --verbose \
    --fp16 \
    > result-FP16.txt

# 读取上面构建的 result-FP32.plan 并作推理
trtexec \
    --loadEngine=./model-FP32.plan \
    --shapes=x:0:4x1x28x28 \
    --verbose \
    > result-load-FP32.txt
```

• trtexec

- 常用选项，构建阶段

--onnx=./model-NCHW.onnx	指定输入模型文件名
~~--output=y:0~~	指定输出张量名（使用 Onnx 时该选项无效）
--minShapes=x:0:1x1x28x28, --optShapes=x:0:4x1x28x28 , --maxShapes=x:0:16x1x28x28	指定输入形状的范围最小值、最常见值、最大值
--workspace=1024（TRT8.4 以后要用 --memPoolSize）	优化过程可使用显存最大值（单位：MiB）
--fp16，--int8，--noTF32，--best，--sparsity=...	指定引擎精度和稀疏性等属性
--saveEngine=./model.plan	指定输出引擎文件名
--buildOnly	只创建引擎不运行
--verbose	打印详细日志
--timingCacheFile=timing.cache	指定输出优化计时缓存文件名[1]
--profilingVerbosity=detailed	构建期保留更多的逐层信息
--dumpLayerInfo，--exportLayerInfo=layerInfo.txt	导出引擎逐层信息，可与 profilingVerbosity 合用[2]

- [1][2] 教程第四部分有单独的使用介绍

• trtexec

- 常用选项，运行阶段
  - --loadEngine=model.plan　　读取 engine 文件，而不是输入 ONNX 文件
  - --shapes=x:0:1x1x28x28　　指定输入张量形状
  - --warmUp=1000　　热身阶段最短运行时间（单位：ms）
  - --duration=10　　测试阶段最短运行时间（单位：s）
  - --iterations=100　　指定测试阶段运行的最小迭代次数
  - --useCudaGraph　　使用 CUDAGraph 来捕获和执行推理过程[1]
  - --noDataTransfers　　关闭 Host 和 Device 之间的数据传输
  - --streams=2　　使用多个 stream 来运行推理[2]
  - --verbose　　打印详细日志
  - --dumpProfile，--exportProfile=layerProfile.txt　　保存逐层性能数据信息

- [1][2] 教程第四部分有单独的使用介绍

• trtexec

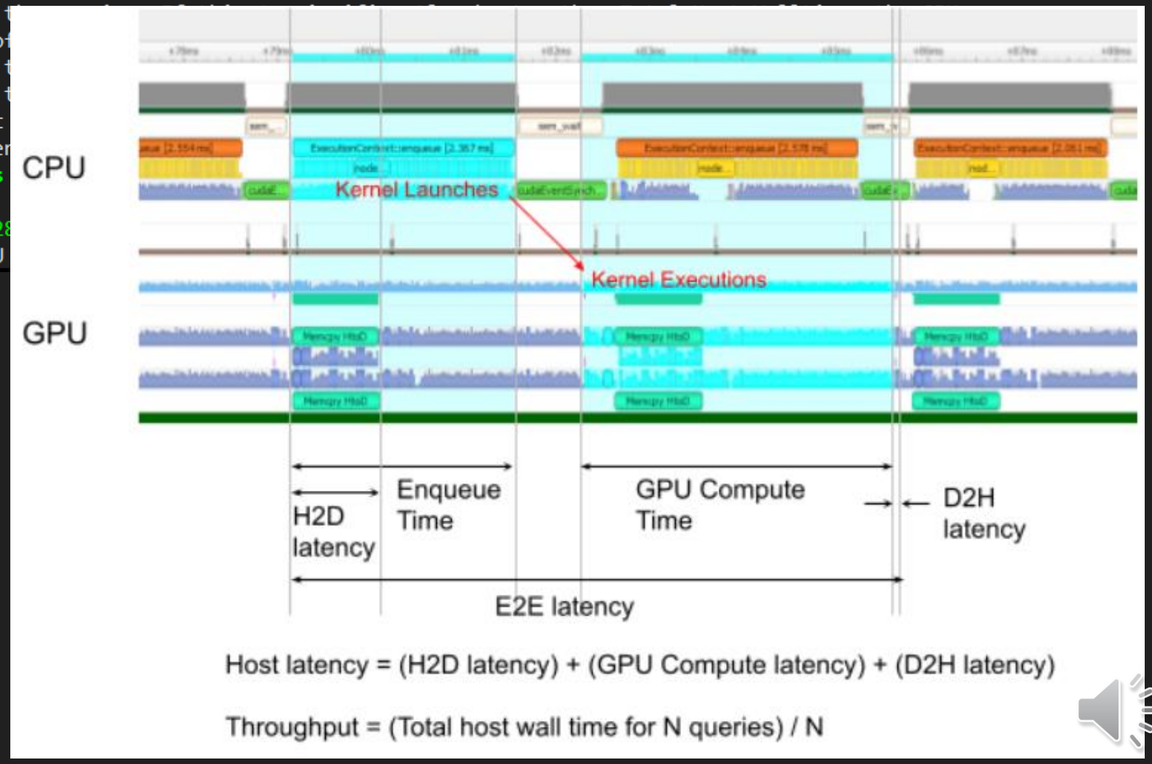
➢ 性能测试结果

吞吐量：每秒完成 query 数量

各种延迟的统计结果

各种延迟的解释

```
[03/07/2022-05:05:52] [I] === Performance summary ===
[03/07/2022-05:05:52] [I] Throughput: 6430.89 qps
[03/07/2022-05:05:52] [I] Latency: min = 0.156982 ms, max = 0.384766 ms, mean = 0.160129 ms, median = 0.159668 ms, percentile(99%) = 0.165771 ms
[03/07/2022-05:05:52] [I] End-to-End Host Latency: min = 0.171387 ms, max = 0.504395 ms, mean = 0.284758 ms, median = 0.289673 ms, percentile(99%) = 0.298828 ms
[03/07/2022-05:05:52] [I] Enqueue Time: min = 0.0300293 ms, max = 0.126465 ms, mean = 0.0396717 ms, median = 0.0333252 ms, percentile(99%) = 0.0698242 ms
[03/07/2022-05:05:52] [I] H2D Latency: min = 0.00317383 ms, max = 0.0455933 ms, mean = 0.00408154 ms, median = 0.00366211 ms, percentile(99%) = 0.00854492 ms
[03/07/2022-05:05:52] [I] GPU Compute Time: min = 0.151367 ms, max = 0.373779 ms, mean = 0.15368 ms, median = 0.153564 ms, percentile(99%) = 0.156677 ms
[03/07/2022-05:05:52] [I] D2H Latency: min = 0.00170898 ms, max = 0.0136719 ms, mean = 0.00237128 ms, median = 0.00219727 ms, percentile(99%) = 0.00341797 ms
[03/07/2022-05:05:52] [I] Total Host Walltime: 3.00036 s
[03/07/2022-05:05:52] [I] Total GPU Compute Time: 2.96525 s
[03/07/2022-05:05:52] [I] Explanations of the performance metrics are print
[03/07/2022-05:05:52] [V]
[03/07/2022-05:05:52] [V] === Explanations of the performance metrics ===
[03/07/2022-05:05:52] [V] Total Host Walltime: the host walltime from when the first query (after warmups) is enqueued to when the last query is completed.
[03/07/2022-05:05:52] [V] GPU Compute Time: the GPU latency to execute the kernels for a query.
[03/07/2022-05:05:52] [V] Total GPU Compute Time: the summation of the GPU Compute Time of all
[03/07/2022-05:05:52] [V] Throughput: the observed throughput computed by dividing the number of
[03/07/2022-05:05:52] [V] Enqueue Time: the host latency to enqueue a query. If this is longer
[03/07/2022-05:05:52] [V] H2D Latency: the latency for host-to-device data transfers for input
[03/07/2022-05:05:52] [V] D2H Latency: the latency for device-to-host data transfers for output
[03/07/2022-05:05:52] [V] Latency: the summation of H2D Latency, GPU Compute Time, and D2H Later
[03/07/2022-05:05:52] [V] End-to-End Host Latency: the duration from when the H2D of a query is
[03/07/2022-05:05:52] [I]
&&&& PASSED TensorRT.trtexec [TensorRT v8003] # trtexec --onnx=model.onnx --minShapes=x:0:1x1x28
[03/07/2022-05:05:52] [I] [TRT] [MemUsageChange] Init cuBLAS/cuBLASLt: CPU +0, GPU +0, now: CPU
```

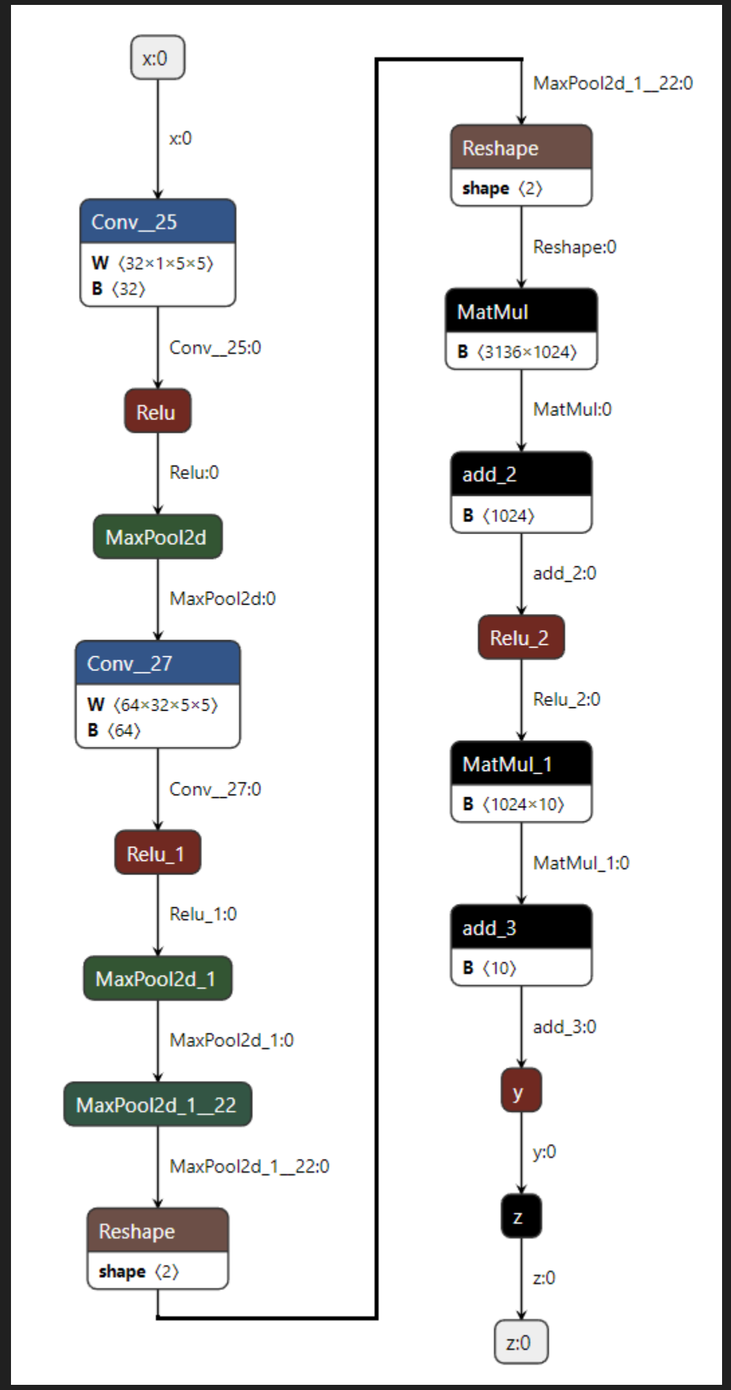


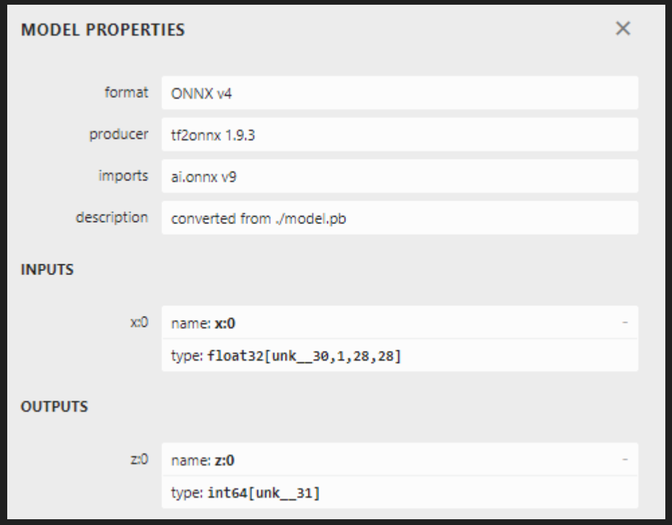
• 开发辅助工具

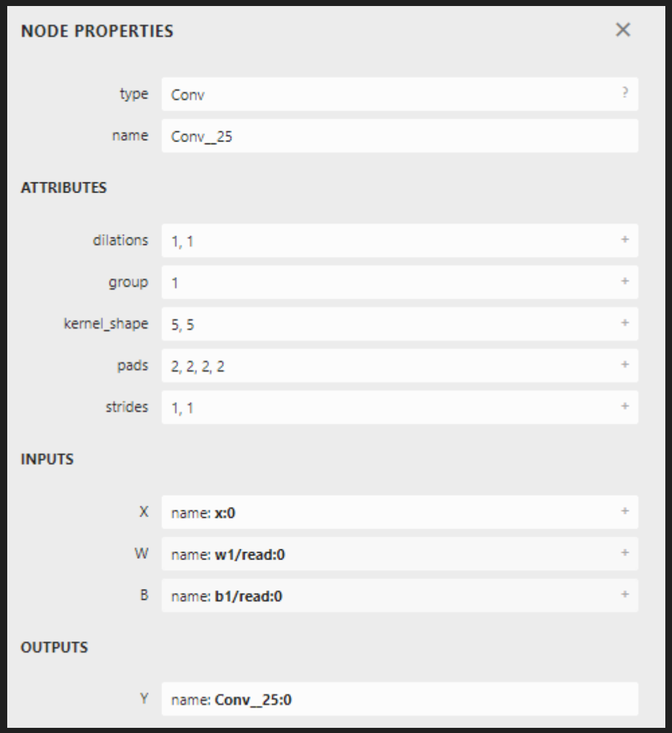
- 希望解决的问题
  - 不想写脚本来跑 TensorRT　　用命令行搞定
  - 怎么进行简单的推理性能测试?　　测量延迟、吞吐量等
  - 网络结构可视化?
  - 计算图上有些节点阻碍 TensorRT 自动优化　　手工调整以便 TensorRT 能够处理
  - 怎么处理 TensorRT 不支持的网络结构?　　手工调整以便 TensorRT 能够处理
  - 怎么检验 TensorRT 上计算结果正确性 / 精度?　　同时在原框架和 TensorRT 上运行
  - 怎么找出计算错误 / 精度不足的层?　　模型逐层比较
  - 怎么进行简单的计算图优化?　　手工调整
  - 怎样找出最耗时的层?　　找到热点集中优化

Netron

- 模型网络可视化工具
- 下载安装
  - https://github.com/lutzroeder/Netron
- 查看网络结构
- 查看计算图信息
- 查看节点信息







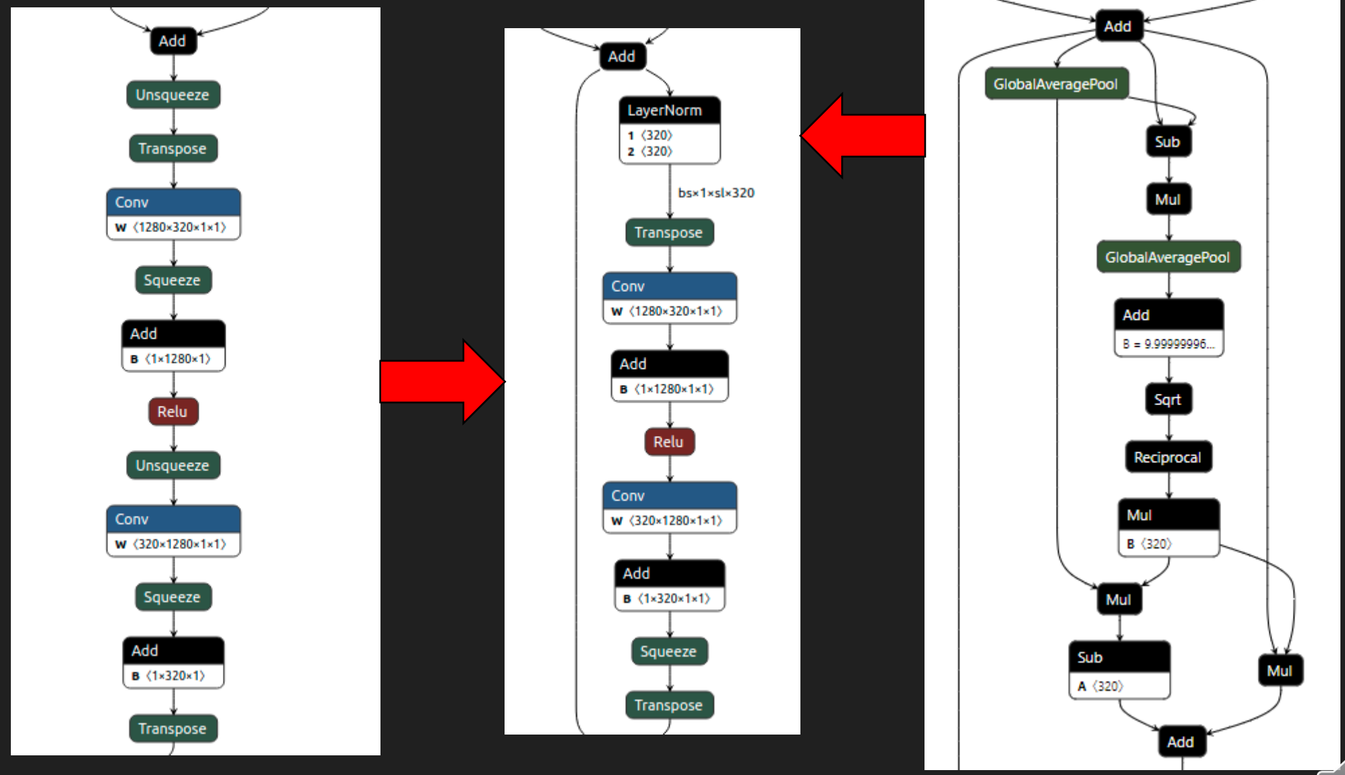
• 开发辅助工具

- 希望解决的问题
  - 不想写脚本来跑 TensorRT　　用命令行搞定
  - 怎么进行简单的推理性能测试?　　测量延迟、吞吐量等
  - 网络结构可视化?
  - 计算图上有些节点阻碍 TensorRT 自动优化　　手工调整以便 TensorRT 能够处理
  - 怎么处理 TensorRT 不支持的网络结构?　　手工调整以便 TensorRT 能够处理
  - 怎么检验 TensorRT 上计算结果正确性 / 精度?　　同时在原框架和 TensorRT 上运行
  - 怎么找出计算错误 / 精度不足的层?　　模型逐层比较
  - 怎么进行简单的计算图优化?　　手工调整
  - 怎样找出最耗时的层?　　找到热点集中优化

• onnx-graphsurgeon

- 需要手工修改网络的情形?
- 冗余节点
- 阻碍 TensorRT 融合的节点组合
- 可以手工模块化的节点

- 更多范例：10-BestPractice（以后陆续更新）



• onnx-graphsurgeon

- ONNX 模型的编辑器，包含 python API（下面简称 ogs）
- 功能：
  - 修改计算图：图属性 / 节点 / 张量 / 节点和张量的连接 / 权重
  - 修改子图：添加 / 删除 / 替换 / 隔离
  - 优化计算图：常量折叠 / 拓扑排序 / 去除无用层
  - 功能和 API 上有别于 onnx 库
- 下载和参考文档
  - pip install nvidia-pyindex onnx-graphsurgeon
  - https://github.com/NVIDIA/TensorRT/tree/master/tools/onnx-graphsurgeon/examples
  - https://docs.nvidia.com/deeplearning/tensorrt/onnx-graphsurgeon/docs/index.html

onnx-graphsurgeon

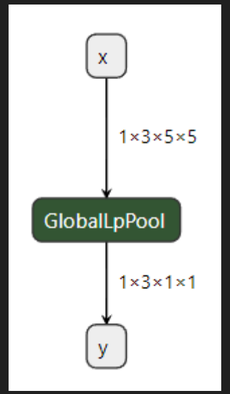
- 范例代码
  - 08-Tool/OnnxGraphSurgeon，运行 test.sh
  - 共有 9 个例子，包含创建模型、隔离子图、替换节点、常量折叠、删除节点、shape 操作

• onnx-graphsurgeon

➢ 01-CreateModel

➢ 使用 ogs 创建 onnx 格式的模型

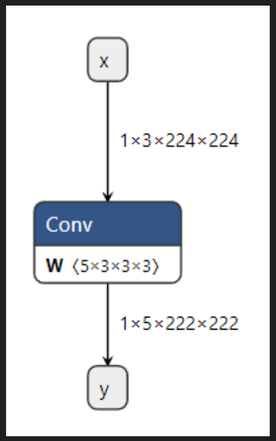
```python
import onnx
import onnx_graphsurgeon as gs
import numpy as np

x = gs.Variable(name="x", dtype=np.float32, shape=[1, 3, 5, 5])
y = gs.Variable(name="y", dtype=np.float32, shape=[1, 3, 1, 1])
node = gs.Node(op="GlobalLpPool", attrs={"p": 2}, inputs=[x], outputs=[y])

graph = gs.Graph(nodes=[node], inputs=[x], outputs=[y])
onnx.save(gs.export_onnx(graph), "01-CreateModel.onnx")
```



➢ 02-CreateModelWithInitializer

➢ 使用 ogs 创建 onnx 格式的模型，具有带权重的节点

```python
import onnx
import onnx_graphsurgeon as gs
import numpy as np

x = gs.Variable(name="x", dtype=np.float32, shape=(1, 3, 224, 224))
y = gs.Variable(name="y", dtype=np.float32, shape=(1, 5, 222, 222))
w = gs.Constant(name="w", values=np.ones(shape=(5, 3, 3, 3), dtype=np.float32))

# w 将被当做 initializer 合并到 Conv 节点中
node = gs.Node(op="Conv", inputs=[x, w], outputs=[y])

graph = gs.Graph(nodes=[node], inputs=[x], outputs=[y])
onnx.save(gs.export_onnx(graph), "02-CeateModelWithInitializer.onnx")
```

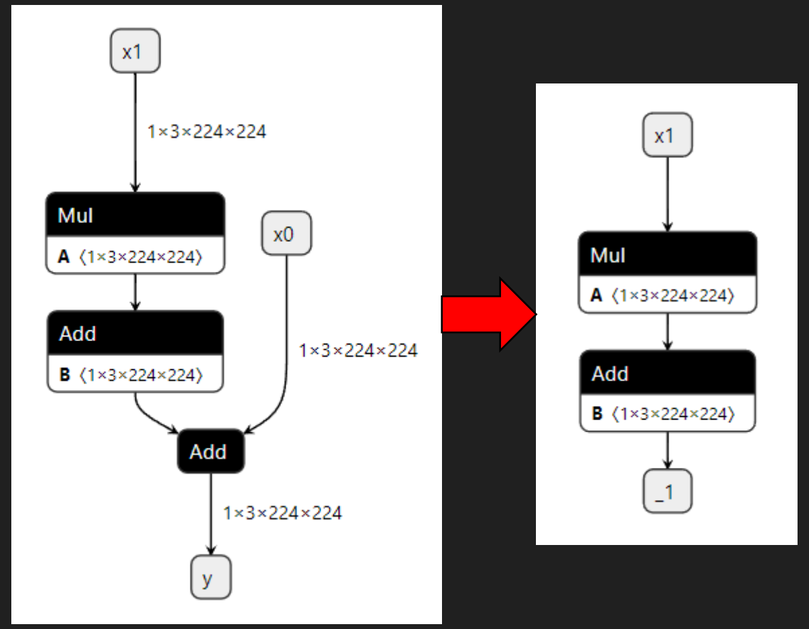


• onnx-graphsurgeon

➢ 03-IsolateSubgraph

➢ 从计算图中取一个子图出来

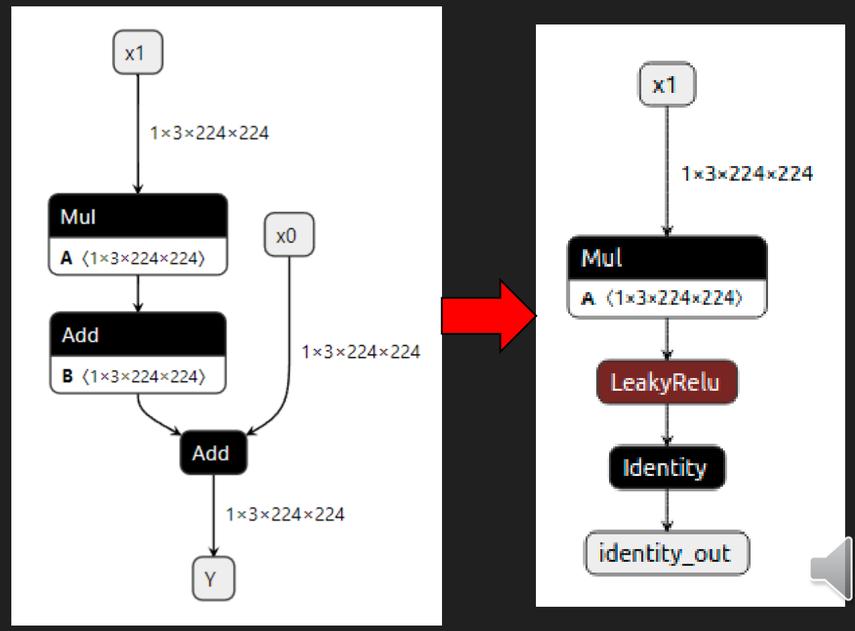
```python
graph = gs.import_onnx(onnx.load("03-IsolateSubgraph_0.onnx"))
tensors = graph.tensors()

graph.inputs = [tensors["x1"].to_variable(dtype=np.float32)]
graph.outputs = [tensors["_1"].to_variable(dtype=np.float32)]

graph.cleanup()
onnx.save(gs.export_onnx(graph), "03-IsolateSubgraph_1.onnx")
```



➢ 04-ModifyModel

➢ 替换并插入一个新节点

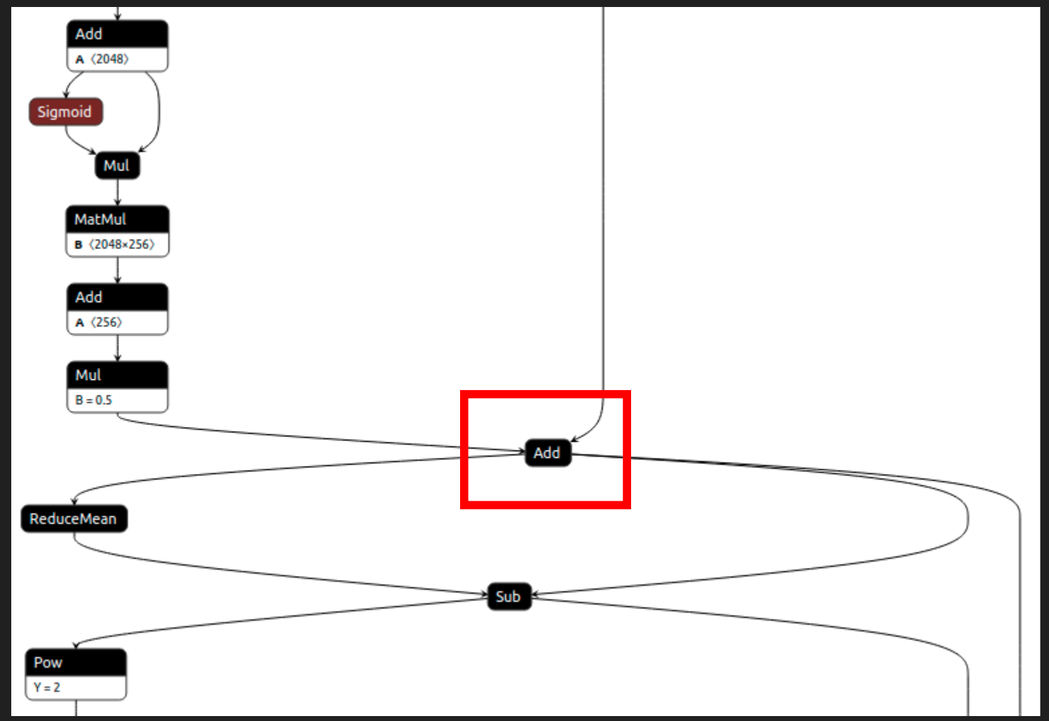
```python
graph = gs.import_onnx(onnx.load("04-ModifyModel_0.onnx"))

# 找到第一个 Add 节点，去掉加数 "b"
first_add = [node for node in graph.nodes if node.op == "Add"][0]
first_add.inputs = [inp for inp in first_add.inputs if inp.name != "b"]

# 将第一个 Add 节点替换为 LeakyRelu 节点
first_add.op = "LeakyRelu"
first_add.attrs["alpha"] = 0.02

# 在第一个 Add 节点后插入一个 Identity 节点
identity_out = gs.Variable("identity_out", dtype=np.float32)
identity = gs.Node(op="Identity", inputs=first_add.outputs, outputs=[identity_out])
graph.nodes.append(identity)

# 修改计算图输入输出
graph.inputs = [graph.inputs[1]]
graph.outputs = [identity_out]

graph.cleanup().toposort()
onnx.save(gs.export_onnx(graph), "04-ModifyModel_1.onnx")
```



• onnx-graphsurgeon

➢ Ogs 的 Node 和 Variable

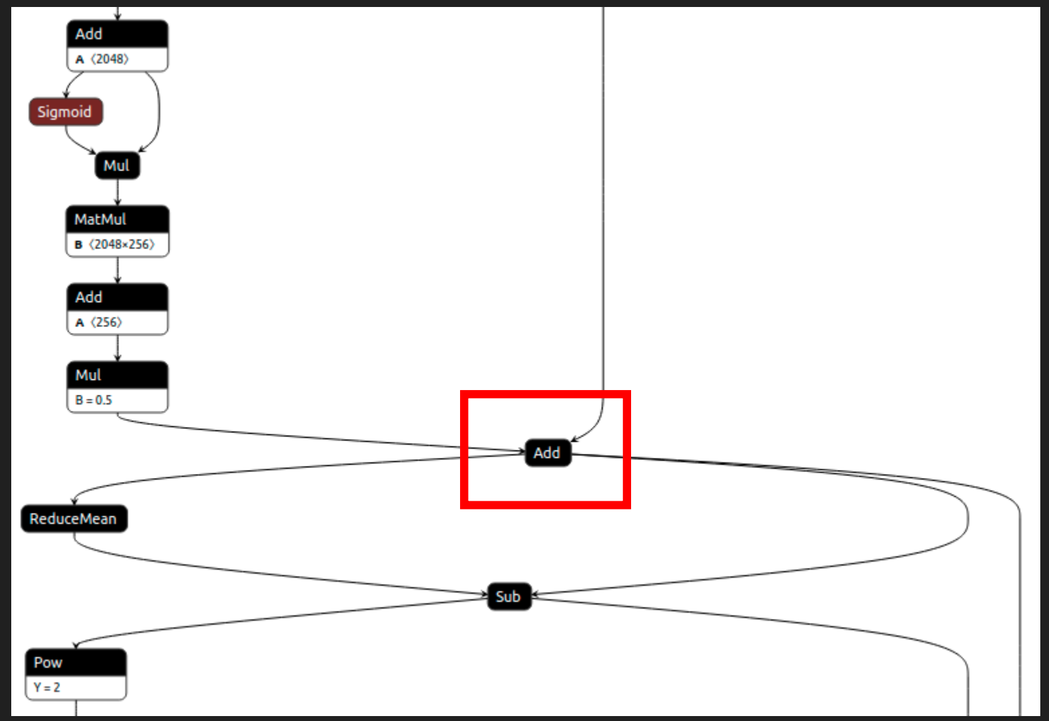
```
node
Add_104 (Add)
    Inputs: [Variable (587): (shape=None, dtype=float32), Variable (634): (shape=None, dtype=float32)]
    Outputs:[Variable (635): (shape=None, dtype=float32)]

node.inputs
[Variable (587): (shape=None, dtype=float32), Variable (634): (shape=None, dtype=float32)]

node.outputs
[Variable (635): (shape=None, dtype=float32)]

node.i(0)
Mul_64 (Mul)
    Inputs: [Variable (585): (shape=None, dtype=float32)Variable (586): (shape=[], dtype=float32)]
    Outputs:[Variable (587): (shape=None, dtype=float32)]

node.i(1)
Mul_103 (Mul)
    Inputs: [Variable (632): (shape=None, dtype=float32), Variable (633): (shape=[], dtype=float32)]
    Outputs:[Variable (634): (shape=None, dtype=float32)]

node.o(0)
ReduceMean_105 (ReduceMean)
    Inputs: [Variable (635): (shape=None, dtype=float32)]
    Outputs:[Variable (636): (shape=None, dtype=float32)]
    Attributes: OrderedDict([('axes', [-1])])

node.o(1)
Sub_106 (Sub)
    Inputs: [Variable (635): (shape=None, dtype=float32), Variable (636): (shape=None, dtype=float32)]
    Outputs:[Variable (637): (shape=None, dtype=float32)]

node.o(2)
Add_180 (Add)
    Inputs: [Variable (635): (shape=None, dtype=float32), Variable (757): (shape=None, dtype=float32)]
    Outputs: [Variable (758): (shape=None, dtype=float32)]
```



• onnx-graphsurgeon

➢ Ogs 的 Node 和 Variable

```
38  node.inputs[0].inputs
39 [Mul_64 (Mul)
40      Inputs: [Variable (585): (shape=None, dtype=float32), Variable (586): (shape=[], dtype=float32)]
41      Outputs: [Variable (587): (shape=None, dtype=float32)]
42  ]
43
44  node.outputs[0].outputs
45 [
46      ReduceMean_105 (ReduceMean)
47          Inputs: [Variable (635): (shape=None, dtype=float32)]
48          Outputs:[Variable (636): (shape=None, dtype=float32)]
49          Attributes: OrderedDict([('axes', [-1])]),
50
51      Sub_106 (Sub)
52          Inputs: [Variable (635): (shape=None, dtype=float32), Variable (636): (shape=None, dtype=float32)]
53          Outputs:[Variable (637): (shape=None, dtype=float32)],
54
55      Add_180 (Add)
56          Inputs: [Variable (635): (shape=None, dtype=float32), Variable (757): (shape=None, dtype=float32)]
57          Outputs:[Variable (758): (shape=None, dtype=float32)]
58  ]
59
60  node.inputs[0].i(0)
61  Variable (585): (shape=None, dtype=float32)
62
63  node.inputs[0].i(1)
64  Variable (586): (shape=[], dtype=float32)
65
66  node.outputs[0].o(0)
67  Variable (636): (shape=None, dtype=float32)
68
69  node.outputs[0].o(1)
70  Variable (637): (shape=None, dtype=float32)
71
72  node.outputs[0].o(2)
73  Variable (758): (shape=None, dtype=float32)
```

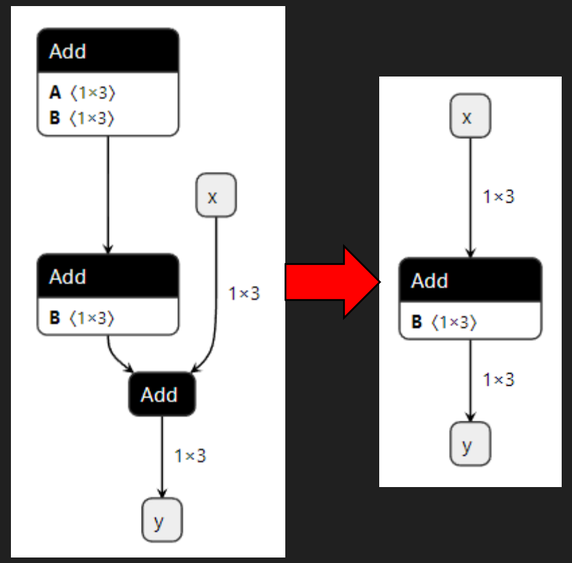


• onnx-graphsurgeon

➢ 05-FoldModel

➢ 常量折叠（一句话的事，用 onnx-simplifier 或 polygraphy 也可以做到）

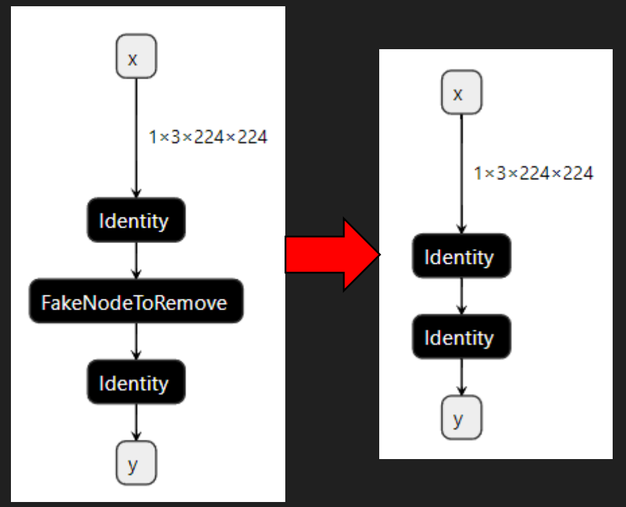
```python
model = onnx.shape_inference.infer_shapes(onnx.load("model.onnx"))  # 带有形状推理的计算图

graph = gs.import_onnx(model)

graph.fold_constants().cleanup()
onnx.save(gs.export_onnx(graph), "05-FoldModel_1.onnx")
```



➢ 06-RemoveNode

➢ 删除节点

```python
graph = gs.import_onnx(onnx.load("06-RemoveNode_0.onnx"))
# 找到欲删除的节点
fake_node = [node for node in graph.nodes if node.op == "FakeNodeToRemove"][0]
# 获取欲删除节点的输入节点（母节点）
inp_node = fake_node.i()
# 将母节点的输出张量赋为欲删除节点的输出张量
inp_node.outputs = fake_node.outputs
fake_node.outputs.clear()

graph.cleanup()
onnx.save(gs.export_onnx(graph), "06-RemoveNode_1.onnx")
```

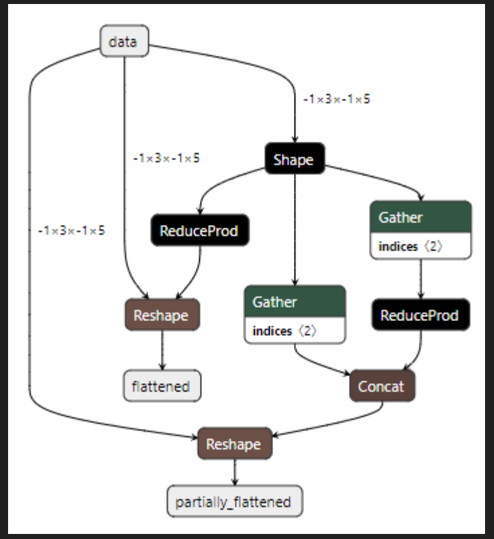


• onnx-graphsurgeon

- 07-BuildModelWithAPI 和 08-RepaceModel
  - 使用 *@gs.Graph.register* 来搭建、包装和调整计算图（用得不多）

- 09-ShapeOperation
  - 与 Dynamic Shape 有关的操作

```python
graph = gs.Graph()

graph.inputs = [gs.Variable(name="data", dtype=np.float32, \
                            shape=(gs.Tensor.DYNAMIC, 3, gs.Tensor.DYNAMIC, 5))]  # shape=(A,3,B,5)

input_shape = graph.shape(graph.inputs[0])  # value=(A,3,B,5), shape=(4,)

volume = graph.reduce_prod(input_shape, axes=[0])  # value=(A*3*B*5), shape=()
flattened = graph.reshape(graph.inputs[0], volume)  # shape=(A*3*B*5,)

NC = graph.gather(input_shape, indices=[0, 1])  # value=(A,3), shape=(2,)
HW = graph.gather(input_shape, indices=[2, 3])  # value=(B,5), shape=(2,)
new_shape = graph.concat([NC, graph.reduce_prod(HW, axes=[0])])  # value=(A,3,B*5), shape=(3,)
partially_flattened = graph.reshape(graph.inputs[0], new_shape)  # shape=(A,3,B*5)

flattened.name = "flattened"
flattened.dtype = np.float32
partially_flattened.name = "partially_flattened"
partially_flattened.dtype = np.float32

graph.outputs = [flattened, partially_flattened]
onnx.save(gs.export_onnx(graph), "09-ShapeOperation.onnx")
```



开发辅助工具

- 希望解决的问题
  - 不想写脚本来跑 TensorRT — 用命令行搞定
  - 怎么进行简单的推理性能测试? — 测量延迟、吞吐量等
  - 网络结构可视化?
  - 计算图上有些节点阻碍 TensorRT 自动优化 — 手工调整以便 TensorRT 能够处理
  - 怎么处理 TensorRT 不支持的网络结构? — 手工调整以便 TensorRT 能够处理
  - 怎么检验 TensorRT 上计算结果正确性 / 精度? — 同时在原框架和 TensorRT 上运行
  - 怎么找出计算错误 / 精度不足的层? — 模型逐层比较
  - 怎么进行简单的计算图优化? — 手工调整
  - 怎样找出最耗时的层? — 找到热点集中优化

polygraphy

- 深度学习模型调试器，包含 python API 和命令行工具（这里只介绍命令行）
- 功能：
  - 使用多种后端运行推理计算，包括 TensorRT，onnxruntime，TensorFlow
  - 比较不同后端的逐层计算结果
  - 由模型文件生成 TensorRT 引擎并序列化为 .plan
  - 查看模型网络的逐层信息
  - 修改 Onnx 模型，如提取子图，计算图化简
  - 分析 Onnx 转 TensorRT 失败原因，将原计算图中可以 / 不可以转 TensorRT 的子图分割保存
  - 隔离 TensorRT 中的错误 tactic
- 下载和参考文档：
  - pip install polygraphy
  - https://docs.nvidia.com/deeplearning/tensorrt/polygraphy/docs/index.html
  - https://www.nvidia.com/en-us/on-demand/session/gtcspring21-s31695/（onnx-graphsurgeon 和 polygraphy 的一个视频教程）

• polygraphy

- Run 模式
  - 范例代码

    08-Tool/polygraphy/runExample，运行 ./command.sh
  - 首先生成一个 .onnx 文件（同前面基于 MNIST 的模型）
  - 其次使用 polygraphy 生成一个 FP16 的 TRT 引擎，并对比使用 onnxruntime 和 TensorRT 的计算结果
  - 然后使用 polygraphy 生成一个 FP32 的 TRT 引擎，将网络中所有层都标记为输出，并对比使用 onnxruntime 和 TensorRT 的计算结果（逐层结果对比）
  - 最后（右图中没有）使用 polygraphy 生成一个“用于生成 FP32 的 TRT 引擎”的 python 脚本，可用于理解 polygraphy 的 API 和原理，这里不展开

```
clear

rm ./*.pb ./*.onnx ./*.plan ./result-*.txt

# 从 TensorFlow 创建一个 .onnx 用来做 polygraphy 的输入文件
python getOnnxModel.py

# 01 用上面的 .onnx 构建一个 TensorRT 引擎，使用 FP16精度，同时在 onnxruntime 和 TensorRT 中运行，对比结果
polygraphy run model.onnx \
    --onnxrt --trt \
    --workspace 1000000000 \
    --save-engine=model-FP16.plan \
    --atol 1e-3 --rtol 1e-3 \
    --fp16 \
    --verbose \
    --trt-min-shapes 'x:0:[1,1,28,28]' \
    --trt-opt-shapes 'x:0:[4,1,28,28]' \
    --trt-max-shapes 'x:0:[16,1,28,28]' \
    --input-shapes   'x:0:[4,1,28,28]' \
    > result-run-FP16.txt

# 注意参数名和格式跟 trtexec 不一样，多个形状之间用空格分隔，如：
# --trt-max-shapes 'input0:[16,320,256]' 'input1:[16, 320]' 'input2:[16]'

# 02 用上面的 .onnx 构建一个 TensorRT 引擎，输出所有层的计算结果作对比
polygraphy run model.onnx \
    --onnxrt --trt \
    --workspace 1000000000 \
    --save-engine=model-FP32-MarkAll.plan \
    --atol 1e-3 --rtol 1e-3 \
    --verbose \
    --onnx-outputs mark all \
    --trt-outputs mark all \
    --trt-min-shapes 'x:0:[1,1,28,28]' \
    --trt-opt-shapes 'x:0:[4,1,28,28]' \
    --trt-max-shapes 'x:0:[16,1,28,28]' \
    --input-shapes   'x:0:[4,1,28,28]'
    > result-run-FP32-MarkAll.txt
```

polygraphy

- Run 模式
  - 输出结果示例

```
[I]     Comparing Output: 'z:0' (dtype=int32, shape=(4,)) with 'z:0' (dtype=int64, shape=(4,)) | Tolerance: [abs=0.001, rel=0.001] | Checking elemwise error
[I]         trt-runner-N0-03/14/22-08:20:47: z:0 | Stats: mean=6.25, std-dev=3.0311, var=9.1875, median=8, min=1 at (2,), max=8 at (0,), avg-magnitude=6.25
[V]             ---- Histogram ----
                Bin Range   |  Num Elems | Visualization
                (1   , 1.7) |          1 | ############
                (1.7, 2.4)  |          0 |
                (2.4, 3.1)  |          0 |
                (3.1, 3.8)  |          0 |
                (3.8, 4.5)  |          0 |
                (4.5, 5.2)  |          0 |
                (5.2, 5.9)  |          0 |
                (5.9, 6.6)  |          0 |
                (6.6, 7.3)  |          0 |
                (7.3, 8   ) |          3 | ########################################
[I]         onnxrt-runner-N0-03/14/22-08:20:47: z:0 | Stats: mean=6.25, std-dev=3.0311, var=9.1875, median=8, min=1 at (2,), max=8 at (0,), avg-magnitude=6.25
[V]             ---- Histogram ----
                Bin Range   |  Num Elems | Visualization
                (1   , 1.7) |          1 | ############
                (1.7, 2.4)  |          0 |
                (2.4, 3.1)  |          0 |
                (3.1, 3.8)  |          0 |
                (3.8, 4.5)  |          0 |
                (4.5, 5.2)  |          0 |
                (5.2, 5.9)  |          0 |
                (5.9, 6.6)  |          0 |
                (6.6, 7.3)  |          0 |
                (7.3, 8   ) |          3 | ########################################
[I]         Error Metrics: z:0
[I]             Minimum Required Tolerance: elemwise error | [abs=0] OR [rel=0] (requirements may be lower if both abs/rel tolerances are set)
[I]             Absolute Difference | Stats: mean=0, std-dev=0, var=0, median=0, min=0 at (0,), max=0 at (0,), avg-magnitude=0
```

TRT 计算结果统计

Onnxruntime 计算结果统计

两者结果之差的统计

• polygraphy

- Run 模式常用选项
  - --model-type 输入模型文件类型，可用 frozen, keras, ckpt, onnx, engine 等
  - --verbose 打印详细日志
  - --trt --tf –onnxrt 选用的后端（可以指定多个）
  - --input-shapes "x:[4,1,28,28]" "y:[4,8]" 做推理时的实际数据形状，注意跟 trtexec 格式不同
  - --ckpt "./model.ckpt" --save-pb "./model.pb" --freeze-graph TensorFlow相关的模型在载入和保存
  - --save-onnx ".model.onnx" --opset 13 指定导出 Onnx 文件名和算子集编号
  - --shape-inference 启用 Onnx 的形状推理功能
  - --trt-min-shapes 'x:[1,1,28,28]' 'y:[1,8]' --trt-opt-shapes 'x:[4,1,28,28]' 'y:[4,8]' --trt-max-shapes 'x:[16,1,28,28]' 'y:[16,8]'
    设定 TensorRT Dynamic Shape 的输入性状范围
  - --fp16 --int8 --tf32 --sparse-weights 指定 TensorRT 中网络计算精度和稀疏性
  - --workspace 1G 指定 TensorRT 工作空间大小，可用 M 或者 G 做后缀
  - --save-engine "./model.plan" 指定 TensorRT 导出的 .plan 文件名
  - --rtol 1e-6 --atol 1e-6 --validate 比较计算结果时的相对误差/绝对误差上限，并检查 NaN 和 INF

• polygraphy

- inspect 模式
  - 范例代码

    08-Tool/polygraphy/inspectExample，运行 ./command.sh
  - 首先生成一个 .onnx 文件（同前面基于 MNIST 的模型）
  - 其次使用 polygraphy 导出 .onnx 详细信息（计算图属性、逐层属性、权重信息等）
  - 然后生成一个 FP32 的 TRT 引擎，使用 polygraphy 导出 .plan 详细信息（网络属性、引擎binding信息，显存需求等）
  - 使用 polygraphy 判断一个 .onnx 是否完整被 TensorRT 支持
  - 故意生成一个 TensorRT 不完整支持的 .onnx（含一个 NonZero 节点），再次做上述判断

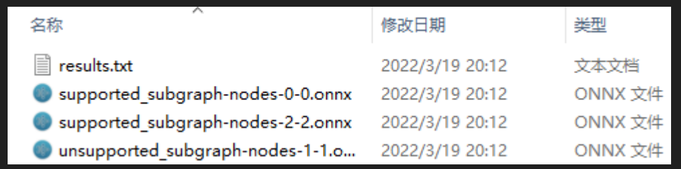
```
clear

rm ./*.pb ./*.onnx ./*.plan ./result-*.txt

# 从 TensorFlow 创建一个 .onnx 用来做 polygraphy 的输入文件
python getOnnxModel.py

# 导出上面 .onnx 的详细信息
polygraphy inspect model model.onnx \
    --mode=full \
    > result-inspectOnnxModel.txt

# 用上面 .onnx 生成一个 .plan 及其相应的 tactics 用于后续分析
polygraphy run model.onnx \
    --trt \
    --workspace 1000000000 \
    --save-engine="./model.plan" \
    --save-inputs="./model-input.txt" \
    --save-outputs="./model-output.txt" \
    --silent \
    --trt-min-shapes 'x:0:[1,1,28,28]' \
    --trt-opt-shapes 'x:0:[4,1,28,28]' \
    --trt-max-shapes 'x:0:[16,1,28,28]' \
    --input-shapes   'x:0:[4,1,28,28]'

# 导出上面 .plan 的详细信息（要求 TensorRT >= 8.2）
polygraphy inspect model model.plan \
    --mode=full \
    > result-inspectPlanModel.txt

# 确认 TensorRT 是否完全原生支持该 .onnx
polygraphy inspect capability model.onnx
# 输出信息：[I] Graph is fully supported by TensorRT; Will not generate subgraphs.

# 生成一个含有 TensorRT 不原生支持的 .onnx，再次用 inspect capability 来确认
python getOnnxModel-NonZero.py

polygraphy inspect capability model-NonZero.onnx > result-NonZero.txt
# 产生目录 .results，包含网络分析信息和支持的子图(.onnx)、不支持的子图(.onnx)
```

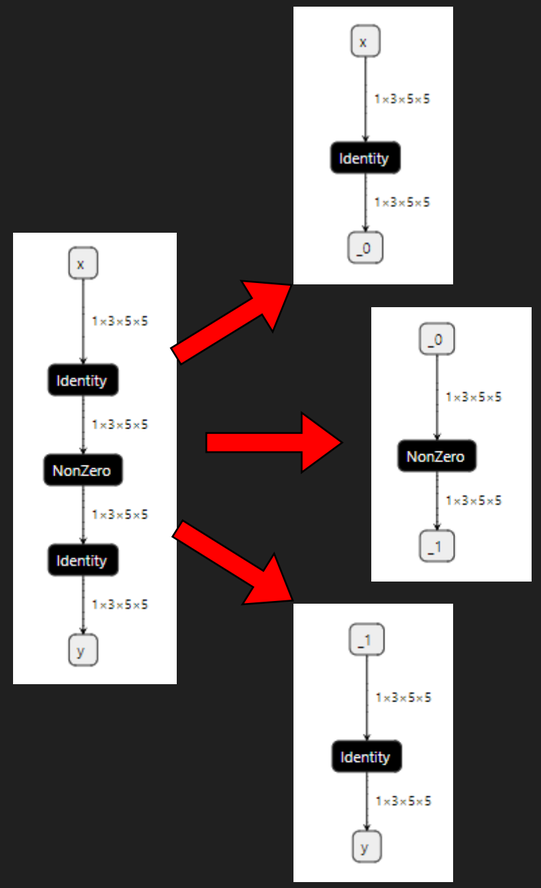
• polygraphy

➢ inspect 模式

➢ 输出结果示例

```
[I] ==== ONNX Model ====
    Name: tf2onnx | Opset: 9

    ---- 1 Graph Input(s) ----
    {x:0 [dtype=float32, shape=('unk__30', 1, 28, 28)]}

    ---- 1 Graph Output(s) ----
    {z:0 [dtype=int64, shape=('unk__31',)]}

    ---- 9 Initializer(s) ----
    Initializer | w4/read:0 [dtype=float32, shape=[1024, 10]] | Values:
        [[ 0.03731047 -0.05782586 -0.12556368 ... -0.08612765 -0.06061816
          -0.08246689]
         [ 0.19088742 -0.06377193 -0.12876913 ...  0.06157729 -0.05597464
          -0.04897792]
         [-0.08788037  0.04984109 -0.03226187 ...  0.09335365  0.03903114
           0.00912305]
         ...
         [-0.07114442 -0.18008882  0.1481347  ... -0.12616785  0.13713098
          -0.05504219]
         [-0.04465632 -0.07094491 -0.04796916 ...  0.02845419 -0.03283151
           0.07002489]
         [-0.12010998  0.02466527  0.00236313 ... -0.09452797  0.07460572
           0.02989185]]

    ...

    ---- 15 Node(s) ----
    Node 0    | Conv__25 [Op: Conv]
        {x:0 [dtype=float32, shape=('unk__30', 1, 28, 28)],
         Initializer | w1/read:0 [dtype=float32, shape=(32, 1, 5, 5)],
         Initializer | b1/read:0 [dtype=float32, shape=(32,)]}
         -> {Conv__25:0}
        ---- Attributes ----
        Conv__25.dilations = [1, 1]
        Conv__25.strides = [1, 1]
        Conv__25.kernel_shape = [5, 5]
        Conv__25.pads = [2, 2, 2, 2]
        Conv__25.group = 1

    ...
```



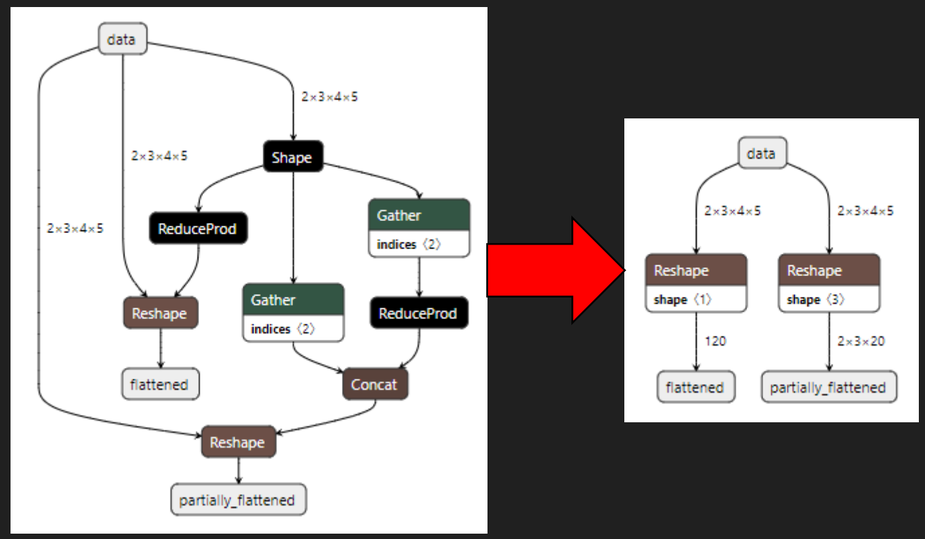


• polygraphy

- surgeon 模式
  - 范例代码

    08-Tool/polygraphy/surgeonExample，运行 ./command.sh
  - 首先生成一个 .onnx 文件（一个 shape 操作相关的网络）
  - 其次使用 polygraphy 优化其计算图

```
clear

rm ./*.pb ./*.onnx ./*.plan ./result-*.txt

# 从 TensorFlow 创建一个 .onnx 用来做 polygraphy 的输入文件
python getShapeOperateOnnxModel.py

# 01 对该模型做图优化
polygraphy surgeon sanitize model.onnx \
    --fold-constant \
    -o model-foldConstant.onnx \
    > result-surgeon.txt
```



• polygraphy

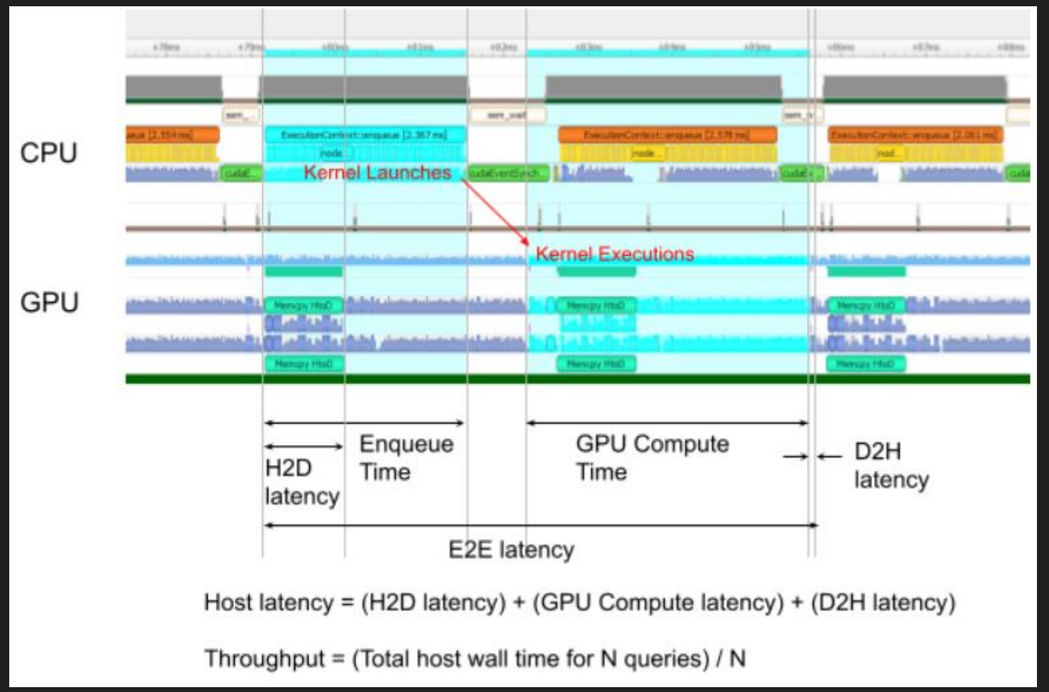
- 其他模式　　*You can never have too many tools*
  - convert 模式
    - 基本同 run 模式
    - 范例代码：08-Tool\Polygraphy\convertExample
  - debug 模式
    - 检查模型转 TensorRT 时的错误并分离出一个可以运行的最大子图
    - 范例代码：08-Tool\Polygraphy\debugExample
  - data 模式
    - 调整和分析用于运行模型的输入输出、权重数据
  - template 模式
    - 用于生成 polygraphy 的 python 脚本，使用脚本对模型进行调整
    - 范例代码：08-Tool\Polygraphy\templateExample

• 开发辅助工具

- 希望解决的问题
  - 不想写脚本来跑 TensorRT — 用命令行搞定
  - 怎么进行简单的推理性能测试? — 测量延迟、吞吐量等
  - 网络结构可视化?
  - 计算图上有些节点阻碍 TensorRT 自动优化 — 手工调整以便 TensorRT 能够处理
  - 怎么处理 TensorRT 不支持的网络结构? — 手工调整以便 TensorRT 能够处理
  - 怎么检验 TensorRT 上计算结果正确性 / 精度? — 同时在原框架和 TensorRT 上运行
  - 怎么找出计算错误 / 精度不足的层? — 模型逐层比较
  - 怎么进行简单的计算图优化? — 手工调整
  - 怎样找出最耗时的层? — 找到热点集中优化

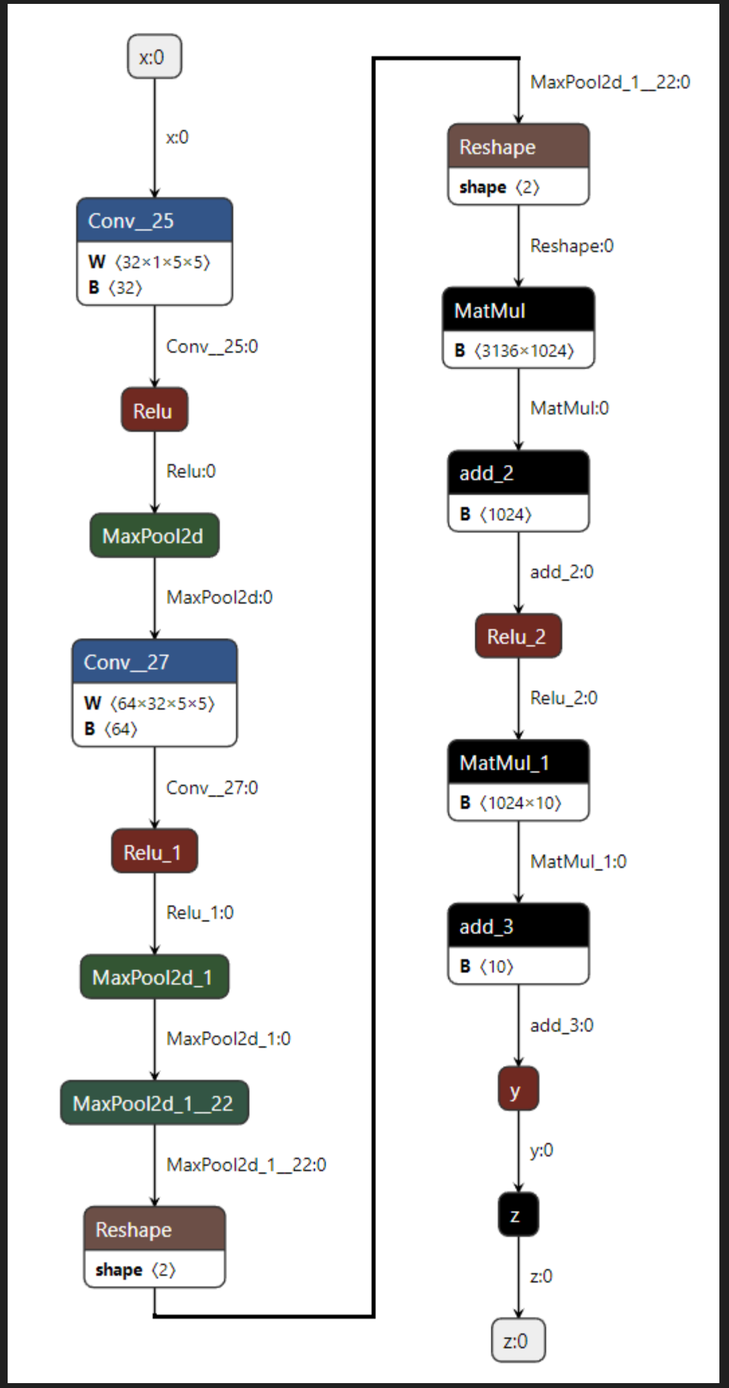
nsight systems

- 性能调试器
  - 随 CUDA 安装或独立安装，位于 /usr/local/cuda/bin/ 下的 nsys 和 nsys-ui
  - 替代旧工具 nvprof 和 nvvp
  - 首先命令行运行 nsys profile XXX，获得 .qdrep 或 .qdrep-nsys 文件
  - 然后打开 nsys-ui，将上述文件拖入即可观察 timeline
- 建议
  - 只计量运行阶段，而不分析构建期
  - 构建期打开 profiling 以便获得关于 Layer 的更多信息
    - builder_config.profilling_verbosity = trt.ProfilingVerbosity.DETAILED
    - 范例代码 09-Advance\ProfilingVerbosity
  - 可以搭配 trtexec 使用：nsys profile -o myProfile trtexec --loadEngine=model.plan --warmUp=0 --duration=0 --iterations=50
  - 也可以配合自己的 script 使用：nsys profile -o myProfile python myScript.py
  - 务必将 nsys 更新到最新版，nsys 不能前向兼容

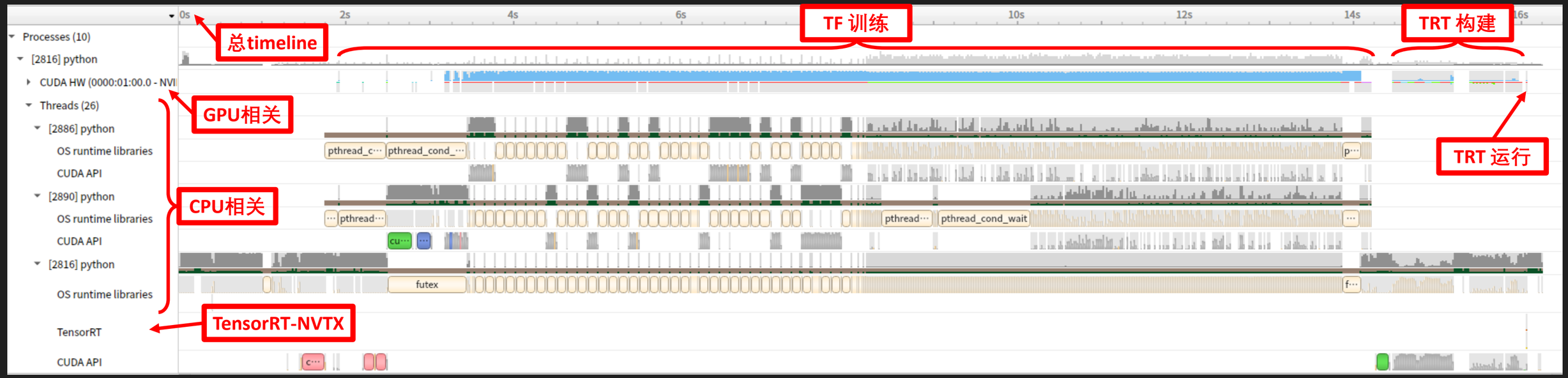


• nsight systems

- 范例代码
  - 08-Tool\NsightSystem，运行 ./command.sh
  - 范例在 TensorFlow 上训练一个 MNIST 手写识别模型，然后在 TensorRT 中用 API 重新搭建该模型，并加载训练好的权重，最后进行 30 次推理
- 下载和参考文档
  - https://developer.nvidia.com/nsight-systems
  - https://docs.nvidia.com/nsight-systems/UserGuide/index.html

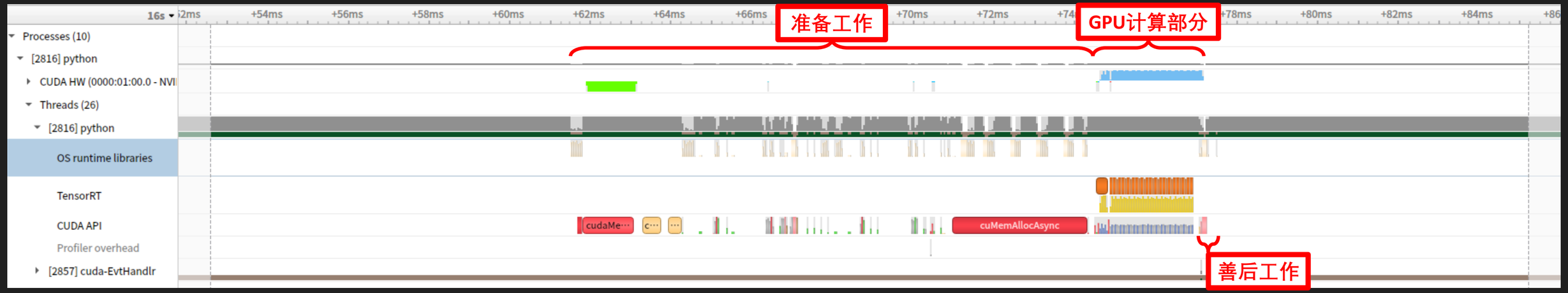


• nsight systems



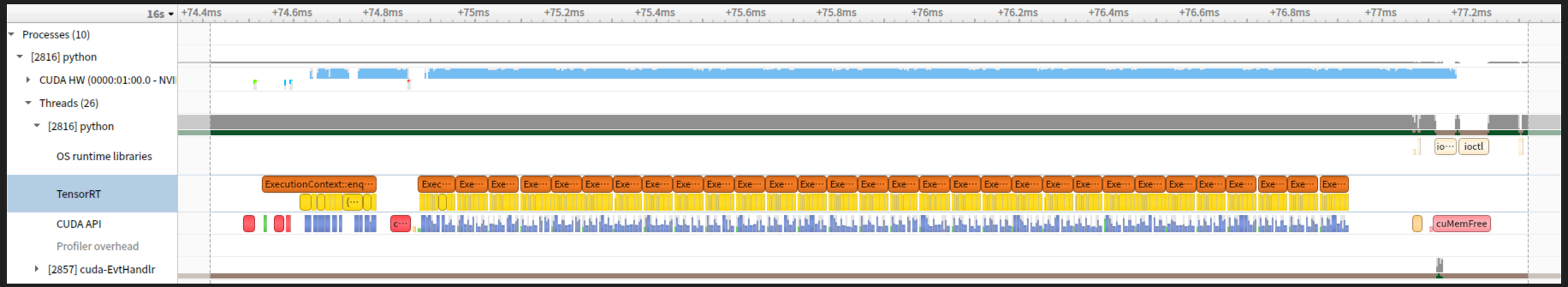
- 整个 timeline
- 建议使用 nsys 只计量 TensoRT 运行部分的时间（上面 timeline 包含 TF 训练、TRT 构建和 TRT 运行）

• nsight systems



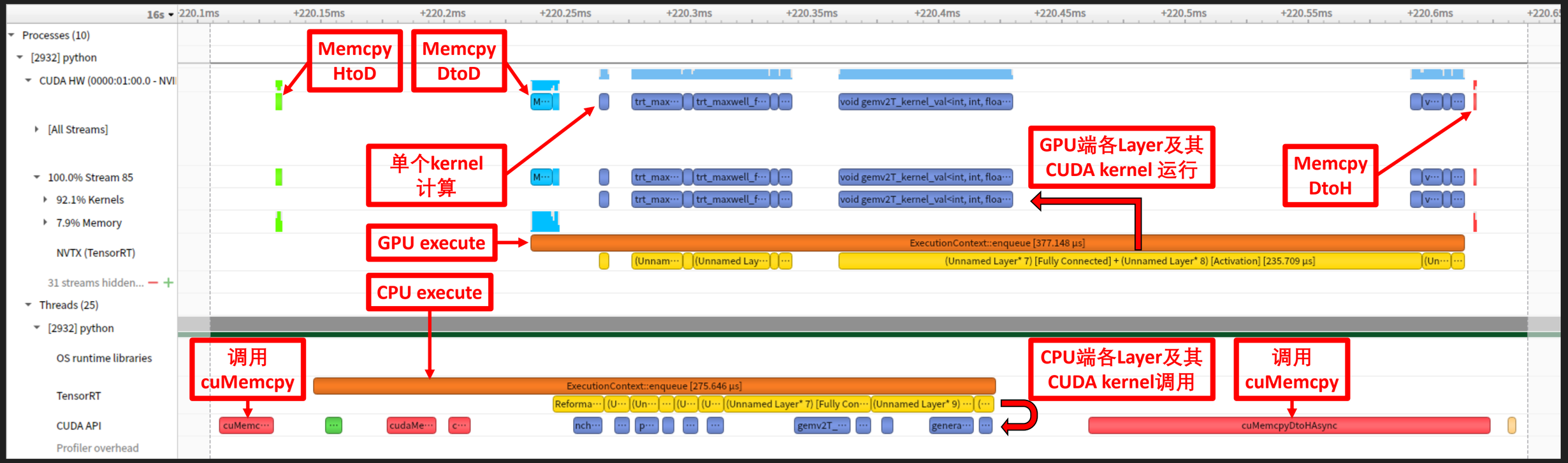
➢ TensorRT 运行阶段的 timeline

• nsight systems



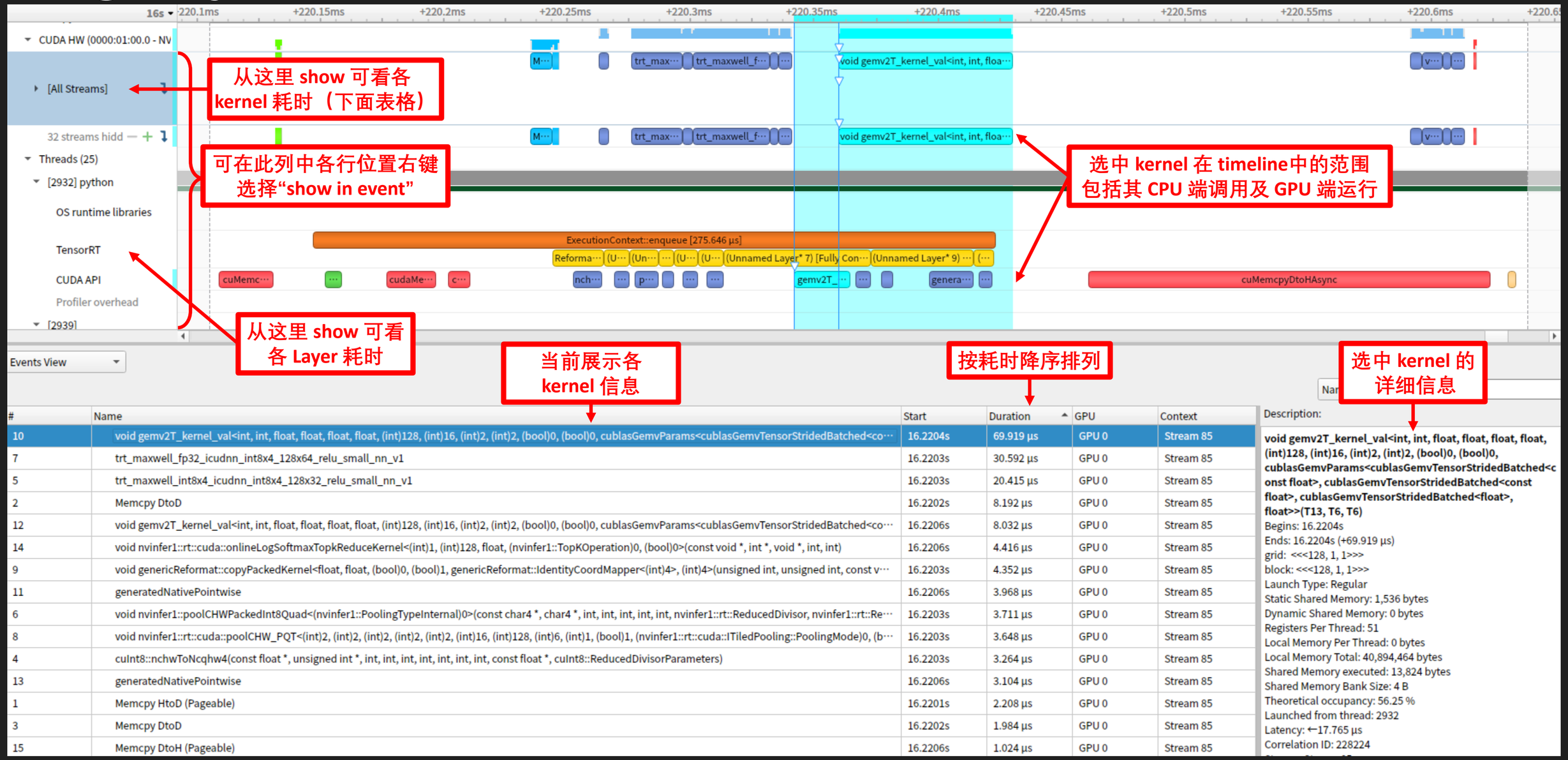
- GPU计算部分的 timeline
- 一共有 31 次 ExecutionContext::enqueue（橘黄色），对应 31 次 context.execute 调用
- 下面的黄色矩形对应网络中各层，蓝色矩形为相应的 cuda kernel 的调用
- 首次推理（warming up）为完整推理过程，前后包含一些 memcpy，后续 30 次只做计算不做 memcpy

• nsight systems



- 单次 inference 的 timeline，展开 GPU部分
- 9 个黄矩形代表网络的 9 个层：
  - Reformat、Conv+ReLU、Pooling、Conv+ReLU、Pooling、Shuffle、FullyConnected+ReLU、 FullyConnected+ReLU、softmax+TopK
- CPU 端调用 cuda API 与 GPU 端实际执行 kernel 之间存在滞后

• nsight systems

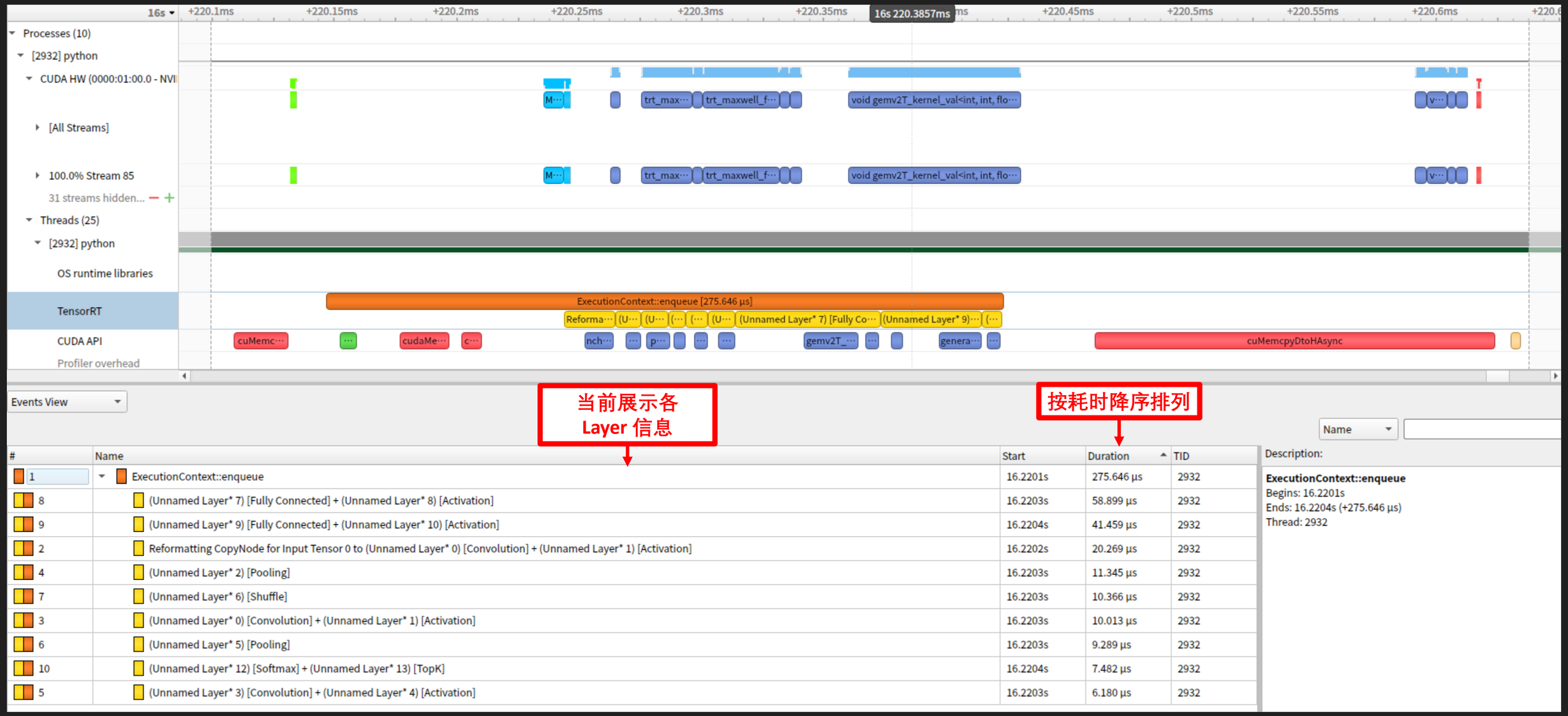


#	Name	Start	Duration	GPU	Context
10	void gemv2T_kernel_val<int, int, float, float, float, float, (int)128, (int)16, (int)2, (int)2, (bool)0, (bool)0, cublasGemvParams<cublasGemvTensorStridedBatched<co···	16.2204s	69.919 µs	GPU 0	Stream 85
7	trt_maxwell_fp32_icudnn_int8x4_128x64_relu_small_nn_v1	16.2203s	30.592 µs	GPU 0	Stream 85
5	trt_maxwell_int8x4_icudnn_int8x4_128x32_relu_small_nn_v1	16.2203s	20.415 µs	GPU 0	Stream 85
2	Memcpy DtoD	16.2202s	8.192 µs	GPU 0	Stream 85
12	void gemv2T_kernel_val<int, int, float, float, float, float, (int)128, (int)16, (int)2, (int)2, (bool)0, (bool)0, cublasGemvParams<cublasGemvTensorStridedBatched<co···	16.2206s	8.032 µs	GPU 0	Stream 85
14	void nvinfer1::rt::cuda::onlineLogSoftmaxTopkReduceKernel<(int)1, (int)128, float, (nvinfer1::TopKOperation)0, (bool)0>(const void *, int *, void *, int, int)	16.2206s	4.416 µs	GPU 0	Stream 85
9	void genericReformat::copyPackedKernel<float, float, (bool)0, (bool)1, genericReformat::IdentityCoordMapper<(int)4>, (int)4>(unsigned int, unsigned int, const v···	16.2203s	4.352 µs	GPU 0	Stream 85
11	generatedNativePointwise	16.2206s	3.968 µs	GPU 0	Stream 85
6	void nvinfer1::poolCHWPackedInt8Quad<(nvinfer1::PoolingTypeInternal)0>(const char4 *, char4 *, int, int, int, int, int, nvinfer1::rt::ReducedDivisor, nvinfer1::rt::Re···	16.2203s	3.711 µs	GPU 0	Stream 85
8	void nvinfer1::rt::cuda::poolCHW_PQT<(int)2, (int)2, (int)2, (int)2, (int)2, (int)16, (int)128, (int)6, (int)1, (bool)1, (nvinfer1::rt::cuda::ITiledPooling::PoolingMode)0, (b···	16.2203s	3.648 µs	GPU 0	Stream 85
4	cuInt8::nchwToNcqhw4(const float *, unsigned int *, int, int, int, int, int, int, int, const float *, cuInt8::ReducedDivisorParameters)	16.2203s	3.264 µs	GPU 0	Stream 85
13	generatedNativePointwise	16.2206s	3.104 µs	GPU 0	Stream 85
1	Memcpy HtoD (Pageable)	16.2201s	2.208 µs	GPU 0	Stream 85
3	Memcpy DtoD	16.2202s	1.984 µs	GPU 0	Stream 85
15	Memcpy DtoH (Pageable)	16.2206s	1.024 µs	GPU 0	Stream 85

void gemv2T_kernel_val<int, int, float, float, float, float, (int)128, (int)16, (int)2, (int)2, (bool)0, (bool)0, cublasGemvParams<cublasGemvTensorStridedBatched<const float>, cublasGemvTensorStridedBatched<const float>, cublasGemvTensorStridedBatched<float>, float>>(T13, T6, T6)
Begins: 16.2204s
Ends: 16.2204s (+69.919 µs)
grid: <<<128, 1, 1>>>
block: <<<128, 1, 1>>>
Launch Type: Regular
Static Shared Memory: 1,536 bytes
Dynamic Shared Memory: 0 bytes
Registers Per Thread: 51
Local Memory Per Thread: 0 bytes
Local Memory Total: 40,894,464 bytes
Shared Memory executed: 13,824 bytes
Shared Memory Bank Size: 4 B
Theoretical occupancy: 56.25 %
Launched from thread: 2932
Latency: ←17.765 µs
Correlation ID: 228224

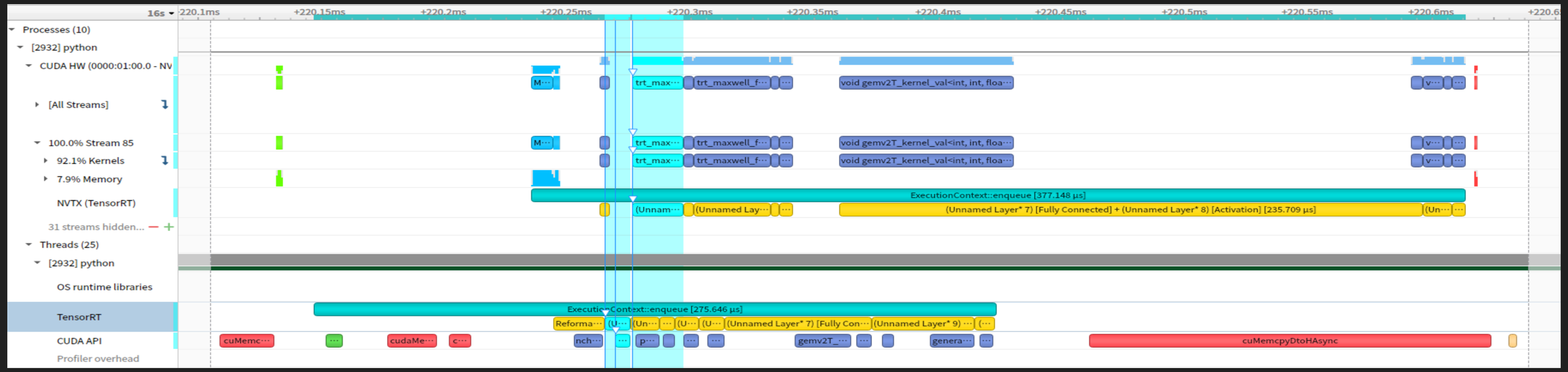
➢ 列出 GPU kernel 运行情况（在 [All Streams] 那一行右键选择 “show in events”）

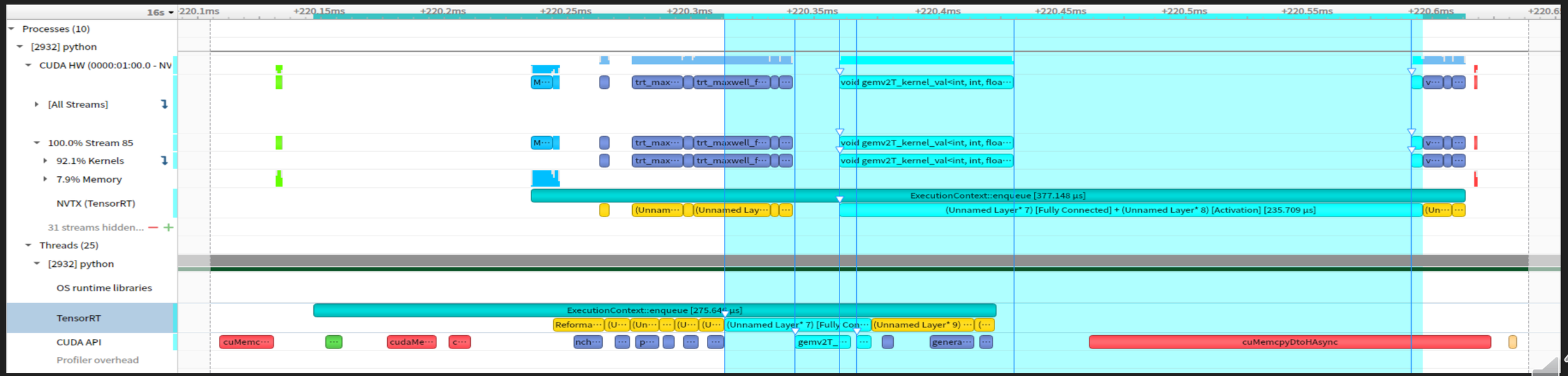
• nsight systems



- 列出 TensorRT 的 NVTX（在 TensorRT 那一行右键选择 “show in events”）

- **nsight systems**





Part 3

TensorRT Plugin

- Plugin 简介
- 使用 Plugin 的简单例子
- TensorRT 与 Plugin 的交互
- 关键 API
- 结合使用 Parser 和 Plugin
- 高级话题
- 使用 Plugin 的案例

• Plugin 简介

- 功能
  - 以 .so 的形式插入到网络中实现某些算子
  - 实现 TensorRT 不原生支持的层或结构
  - 替换性能不足的层或结构
  - 手动合并 TensorRT 没有自动融合的层或结构
- 限制条件
  - 自己实现 CUDA C++ kernel，为结果精度和性能负责
  - Plugin 与其他 Layer 之间无法 fusing
  - 可能在 Plugin 节点前后插入 reformatting 节点，增加开销
- 建议
  - 先尝试原生 Layer 的组合保证计算正确性
  - 尝试 TensorRT 自带的 Plugin 是否满足要求
  - 还是不满意，自己写！

- TensorRT 自带的 Plugin 及使用说明
  - https://github.com/NVIDIA/TensorRT/tree/main/plugin

Plugin	Name	Plugin	Name
batchTilePlugin	BatchTilePlugin_TRT	leakyReluPlugin	LReLU_TRT
batchedNMSPlugin	BatchedNMS_TRT	multilevelCropAndResizePlugin	MultilevelCropAndResize_TRT
batchedNMSDynamicPlugin	BatchedNMSDynamic_TRT	multilevelProposeROI	MultilevelProposeROI_TRT
bertQKVToContextPlugin	CustomQKVToContextPluginDynamic	nmsPlugin	NMS_TRT
coordConvACPlugin	CoordConvAC	normalizePlugin	Normalize_TRT
cropAndResizePlugin	CropAndResize	nvFasterRCNN	RPROI_TRT
decodeBbox3DPlugin	DecodeBbox3DPlugin	pillarScatterPlugin	PillarScatterPlugin
detectionLayerPlugin	DetectionLayer_TRT	priorBoxPlugin	PriorBox_TRT
efficientNMSPlugin	EfficientNMS_TRT	proposalLayerPlugin	ProposalLayer_TRT
efficientNMSONNXPlugin	EfficientNMS_ONNX_TRT	proposalPlugin	Proposal
embLayerNormPlugin	CustomEmbLayerNormPluginDynamic	pyramidROIAlignPlugin	PyramidROIAlign_TRT
fcPlugin	CustomFCPluginDynamic	regionPlugin	Region_TRT
flattenConcat	FlattenConcat_TRT	reorgPlugin	Reorg_TRT
geluPlugin	CustomGeluPluginDynamic	resizeNearestPlugin	ResizeNearest_TRT
generateDetectionPlugin	GenerateDetection_TRT	scatterPlugin	ScatterND
gridAnchorPlugin	GridAnchor_TRT	skipLayerNormPlugin	CustomSkipLayerNormPluginDynamic
gridAnchorRectPlugin	GridAnchorRect_TRT	specialSlicePlugin	SpecialSlice_TRT
groupNormalizationPlugin	GroupNormalizationPlugin	splitPlugin	Split
instanceNormalizationPlugin	InstanceNormalization_TRT	voxelGeneratorPlugin	VoxelGeneratorPlugin

• Plugin 关键问题

- 怎么从头开始实现一个 Plugin? 要写哪些类和函数
- 怎么把 Plugin 接到 TensorRT 网络中去? 要怎么包装 kernel 以便 TensorRT 识别
- TensorRT 怎么参与 Plugin 的资源管理? 两者之间要交换些什么信息
- Plugin 能不能序列化到 .plan 中去?
- Plugin 有些什么扩展性? FP16/INT8，Dynamic Shape，data-dependent-shape，...
- Plugin 与原生 Layer 相比性能怎么样?

Plugin 的简单例子

- 范例代码
  - 05-Plugin\usePluginV2DynamicExt
  - 功能：给输入张量所有元素加上同一个值

- 实现步骤
  - 继承 IPluginV2DynamicExt 类实现一个Plugin 类
  - 继承 IPluginCreator 类实现一个 PluginCreator 类
  - 实现用于计算的 CUDA C++ kernel
  - 将 Plugin 编译为 .so 保存
  - 在 TenorRT 中加载和使用 Plugin

```cpp
namespace nvinfer1
{
class AddScalarPlugin : public IPluginV2DynamicExt
{
private:
    // ... // 一些自定义的数据

public:
    // .. // 重写部分成员函数
};

class AddScalarPluginCreator : public IPluginCreator
{
private:
    // ... // 一些自定义的数据

public:
    // .. // 重写部分成员函数

};

} // namespace nvinfer1
```

```cpp
__global__ void addScalarKernel(const float *input,
                                float *output,
                                const float scalar,
                                const int nElement)
{
    const int index = blockIdx.x * blockDim.x + threadIdx.x;
    if (index >= nElement)
        return;

    float _1      = input[index];
    float _2      = _1 + scalar;
    output[index] = _2;
}
```

```python
trt.init_libnvinfer_plugins(logger, '')
ctypes.cdll.LoadLibrary('AddScalar.so')

for creator in trt.get_plugin_registry().plugin_creator_list:
    if creator.name == 'AddScalar':
        parameterList = []
        parameterList.append(trt.PluginField("scalar", np.float32(scalar), trt.PluginFieldType.FLOAT32))
        plugin = creator.create_plugin(creator.name, trt.PluginFieldCollection(parameterList))

pluginLayer = network.add_plugin_v2([inputT0], plugin)
```

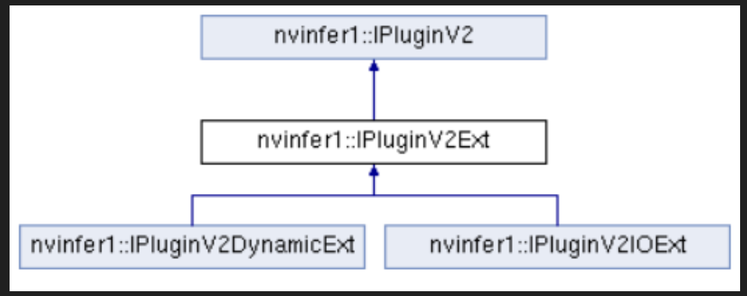
Plugin 与 TensorRT 的交互

- 构建期
  - TensorRT 向 Plugin 传输参数和权重
  - Plugin 向 TensorRT 报告其输入输出张量信息，包括数量、形状（Shape）、数据类型（DataType）和数据排布（Layout）组合
  - Plugin 向 TensorRT 报告其需要的 workspace 大小
  - TensorRT 尝试各种允许的组合，选择性能最佳的输入输出组合（可能在 Plugin 前后插入 reformat 节点）
  - Plugin 不参与层 fusing

- 运行期
  - TensorRT 为 Plugin 提供输入输出张量的地址，workspace 的地址，以及所在的 stream

• Plugin 的类型

- ~~IPluginV1~~
- IPluginV2 （支持单一 input / output 格式）
- IPluginV2Ext （支持单一 input 格式和混合 output 格式）
- IPluginV2IOExt （支持混合 input / output 格式，Implicit Batch 模式）
- **IPluginV2DynamicExt** （支持混合 input / output 格式，Dynamic Shape 模式）



- 范例代码
  - 05-Plugin/usePluginV2IOExt 和 05-Plugin/usePluginV2DynamicExt，使用两种类实现同一个 Plugin，并搭建网络进行推理

• Plugin 的关键 API

- 以 IPluginV2DynamicExt 类为例

- 构造函数
  - 接受来自 PluginCreator 的参数
  - 做一些构建期的初始化工作

```cpp
CuBLASGemmPlugin::CuBLASGemmPlugin(const std::string &name,
                                   Weights weight,
                                   int k,
                                   int n,
                                   bool needDeepCopy = false):// 传入各种参数
    name_(name),
    bOwnWeight_(needDeepCopy),
    nK_(k), nN_(n)
{
    WHERE_AM_I()
    assert(weight.type == DataType::kFLOAT);    // 检查传入的参数
    assert(weight.values != nullptr);
    assert(weight.count == k * n);

    weight_.type  = DataType::kFLOAT;           // 设置 Plugin 内部变量
    weight_.count = weight.count;
    if (needDeepCopy)
    {
        size_t size    = sizeof(float) * weight.count;
        weight_.values = malloc(size);
        memcpy(reinterpret_cast<char *>(const_cast<void *>(weight_.values)), weight.values, size);
    }
    else
    {
        weight_.values = weight.values;
    }

    CHECK(cublasCreate(&handle_));              // 创建某些需要的构件
}
```

• Plugin 的关键 API

- getOutputDimensions
  - 向 TensorRT 报告每个输出张量的形状
  - Dynamic Shape 模式输入输出张量形状不一定是构建期常量，需要用表达式来计算
  - outputIndex 只索引输出张量
  - 范例代码
    - 设该 Plugin 输入两个张量，输出三个张量
    - 某次 inference 输入张量形状分别为 [2,3,5] 和 [7,9]
    - 输出张量形状分别为 [2,11,3,5+7] 和 [7,9] 和 []

```
1   DimsExprs getOutputDimensions(int32_t outputIndex,
2                                 const DimsExprs *inputs,
3                                 int32_t nbInputs,
4                                 IExprBuilder &exprBuilder) noexcept
5   {
6       DimsExprs ret;
7
8       switch (outputIndex)
9       {
10      case 0:
11          ret.nbDims = 4;
12          ret.d[0]   = inputs[0].d[0];
13          ret.d[1]   = exprBuilder.constant(11);
14          ret.d[2]   = inputs[0].d[1];
15          ret.d[3]   = exprBuilder.operation(DimensionOperation::kSUM,
16                                             *inputs[0].d[2],
17                                             *inputs[1].d[0]);
18          return ret;
19
20      case 1:
21          return ipnuts[1];
22
23      case 2:
24          ret.nbDims = 0;
25          return ret;
26
27      default:  // should NOT be here!
28          return inputs[0];
29      }
30  }
```

Plugin 的关键 API

- supportsFormatCombination
  - Plugin 可以同时支持多种数据类型（DataType）和数据排布（LayerOut）的输入输出张量组合
  - TensorRT 深度优先遍历各 pos 尝试各种组合，该函数返回“Plugin 是否支持当前尝试的组合”
  - pos 同时索引输入和输出张量
  - 第 k 号张量的条件可以基于 0~k-1 号张量的条件
  - 打开 FP16 模式时才会尝试含 kHALF 的组合
  - 打开 INT8 模式时才会尝试含 kINT8 的组合
  - 范例代码
    - 设该 Plugin 输入两个张量，输出一个张量
    - 要求两个输入张量同时为 Int 或 Float，输出张量总是 Int
  - 建议
    - 保证性能的前提下， 多实现一些 format 的组合， 消除 TenosrRT 自动插入 reformat 节点的开销

```cpp
bool supportsFormatCombination(int32_t pos,
                               const PluginTensorDesc *inOut,
                               int32_t nbInputs,
                               int32_t nbOutputs) noexcept
{
    WHERE_AM_I()
    if (inOut[pos].format != TensorFormat::kLINEAR)
        return false;

    switch (pos)
    {
    case 0:
        return inOut[0].type == DataType::kFLOAT || inOut[0].type == DataType::kINT32;
    case 1:
        return inOut[1].type == inOut[0].type;
    case 2:
        return inOut[2].type == DataType::kINT32;
    default: // should NOT be here!
        return false;
    }
    return false;
}
```

Plugin 的关键 API

- configurePlugin
  - 在推理前将调用该成员函数
  - Dynamic Shape 模式中，每当输入数据形状发生变化（调用 context.set_binding_shape）时，该成员函数被调用
  - 构建期调用时 in/out 张量形状中含有 -1
  - 运行期调用时 in/out 张量形状为真实绑定的形状

```
configurePlugin(const DynamicPluginTensorDesc *in,
                int32_t nbInputs,
                const DynamicPluginTensorDesc *out,
                int32_t nbOutputs) noexcept override;
{
    m_.height = in[0].desc.dims.d[1];
    m_.width  = in[0].desc.dims.d[2];
}
```

- getWorkspaceSize
  - 向 TensorRT 报告中间计算结果的存储空间
  - 由 TensorRT 管理，参与显存优化

```
#define ALIGN_SIZE      256
#define ALIGN1024(x)    (((x) + (ALIGN_SIZE) - 1) / (ALIGN_SIZE) * (ALIGN_SIZE))

size_t getWorkspaceSize(const PluginTensorDesc *inputs,
                        int32_t nbInputs,
                        const PluginTensorDesc *outputs,
                        int32_t nbOutputs) const noexcept override;
    {
        return ALIGN(sizeof(int) * inputs[0].dims.d[0] * (m_.height + 2) * (m_.width + 2)) +
               ALIGN(sizeof(char8) * input[0].dims.d[0] * (m_.height + 2) * (m_.width + 2)) +
               ALIGN(sizeof(int) * input[0].dims.d[0]) +
               ALIGN(sizeof(int) * input[0].dims.d[0]);
    }
```

• Plugin 的关键 API

- enqueue
  - 调用 CUDA C++ kernel 计算的地方
  - 可以根据输入张量的不同形状、数据类型等条件选择不同 kernel 执行计算
  - 不要在 enqueue 中使用 cudaMalloc* 等函数

```cpp
int32_t enqueue(const PluginTensorDesc* inputDesc,
                const PluginTensorDesc* outputDesc,
                const void* const* inputs,
                void* const* outputs,
                void* workspace,
                cudaStream_t stream) noexcept
{
    const int nBlock = inputDesc[0].dims.d[0] * inputDesc[0].dims.d[1];
    const int nValuePerBlock = inputDesc[0].dims.d[2];
    if (inputDesc[0].type == DataType::kFLOAT)
    {
        (layerNormKernel<float,256>) <<<nBlock, nValuePerBlock, 0, stream>>>
            ((float *)inputs[0],
                (float *)inputs[1],
                (float *)inputs[2],
                (float *)outputs[0]);
    }
    else
    {
        (layerNormKernel<half,256>) <<<nBlock, nValuePerBlock, 0, stream>>>
            ((half *)inputs[0],
                (float *)inputs[1],
                (float *)inputs[2],
                (half *)outputs[0]);
    }
    return 0;
}
```

- enqueue 在 TensorRT7 和 TensorRT8 中函数声明不同，升级 TRT 版本时注意调整

```cpp
int32_t enqueue(int32_t batchSize, const void * const * inputs, void *       * outputs, void * workspace, cudaStream_t stream)          override;  // TensorRT7
int32_t enqueue(int32_t batchSize, const void * const * inputs, void * const * outputs, void * workspace, cudaStream_t stream) noexcept override;  // TensorRT8
```

• Plugin 的关键 API

- 资源管理函数
  - initialize（engine 创建时被调用，用于初始化 Plugin 层）
  - terminate (engine 销毁时被调用，用于释放 initialize 函数申请的资源)
  - clone（创建多个 context ，可以与源对象共享本 engine 的资源）
  - attachToContext（申请使用 context 独占的 cudnn 或 cublas 资源）
  - detachFromContext（销毁 context 独占的 cudnn 或 cublas 资源）
  - destroy（当 context/engine 销毁时被调用）

- 范例代码
  - 05-Plugin/cuBLASPlugin，在 Plugin 中使用 cuBLAS 并通过引用计数来共享权重

```
1  CuBLASGemmPlugin(..., bool needDeepCopy = false): ...
2  {
3      ...
4      if (needDeepCopy)
5      {
6          size_t size    = sizeof(float) * weight.count;
7          weight_.values = malloc(size);
8          memcpy(reinterpret_cast<char *>(const_cast<void *>(weight_.values)), weight.values, size);
9      }
10     else
11     {
12         weight_.values = weight.values;
13     }
14
15     CHECK(cublasCreate(&handle_));
16 }
17
18 IPluginV2DynamicExt *CuBLASGemmPlugin::clone() const noexcept
19 {
20     CuBLASGemmPlugin *p = new CuBLASGemmPlugin(name_, weight_, nK_, nN_, false);
21     p->setPluginNamespace(namespace_.c_str());
22     p->pGPUWeight_ = this->pGPUWeight_;
23     return p;
24 }
25
26 int32_t CuBLASGemmPlugin::initialize() noexcept
27 {
28     size_t size = sizeof(float) * weight_.count;
29     CHECK(cudaMalloc((void **)&pGPUWeight_, size));
30     CHECK(cudaMemcpy(pGPUWeight_, weight_.values, size, cudaMemcpyHostToDevice));
31     return 0;
32 }
33
34 void CuBLASGemmPlugin::terminate() noexcept
35 {
36     CHECK(cudaFree(pGPUWeight_));
37 }
38
39 void CuBLASGemmPlugin::destroy() noexcept
40 {
41     if (bOwnWeight_)
42     {
43         free(const_cast<void *>(weight_.values));
44     }
45     CHECK(cublasDestroy(handle_));
46 }
```

• Plugin 的关键 API

- 序列化和反序列化
- 序列化（Plugin 负责）
  - getSerializationSize（报告序列化需要的空间大小，单位 Byte）
  - serialize（将Plugin 数据序列化到给定的 buffer 中）
- 反序列化（PluginCreator 负责）
  - deserializePlugin（把序列化的 buffer 传给 Plugin 的构造函数）
  - Plugin 构造函数（从 buffer 中读取数据并完成 Plugin 构造）

```cpp
size_t CuBLASGemmPlugin::getSerializationSize() const noexcept
{
    return sizeof(nK_) + sizeof(nN_) + sizeof(float) * weight_.count;
}

void CuBLASGemmPlugin::serialize(void *buffer) const noexcept
{
    char * data   = reinterpret_cast<char *>(buffer);
    size_t offset = 0;
    memcpy(data + offset, &nK_, sizeof(nK_));
    offset += sizeof(nK_);
    memcpy(data + offset, &nN_, sizeof(nN_));
    offset += sizeof(nN_);
    size_t size = sizeof(float) * nK_ * nN_;
    memcpy(data + offset, weight_.values, size);
}

IPluginV2 *CuBLASGemmPluginCreator::deserializePlugin(const char *name,
                                                      const void *serialData,
                                                      size_t serialLength) noexcept
{
    return new CuBLASGemmPlugin(name, serialData, serialLength);
}
```

```cpp
CuBLASGemmPlugin::CuBLASGemmPlugin(const std::string &name,
                                   const void *buffer,
                                   size_t length): ...
{
    const char *data   = reinterpret_cast<const char *>(buffer);
    size_t      offset = 0;
    memcpy(&nK_, data + offset, sizeof(nK_));
    offset += sizeof(nK_);
    memcpy(&nN_, data + offset, sizeof(nN_));
    offset += sizeof(nN_);

    weight_.type   = DataType::kFLOAT;
    weight_.count  = nK_ * nN_;
    size_t size    = sizeof(float) * nK_ * nN_;
    weight_.values = malloc(size);
    memcpy(reinterpret_cast<char *>(const_cast<void *>(weight_.values)), data + offset, size);
}
```

• Plugin 的关键 API

- PluginCreator 部分
  - createPlugin（依照传入参数调用 Plugin 构造函数）

- 注册 PluginCreator
  - REGISTER_TENSORRT_PLUGIN(CuBLASGemmPluginCreator);

```
1  IPluginV2 *createPlugin(const char *name, const PluginFieldCollection *fc) noexcept
2  {
3      int     k, n;
4      Weights w;
5      for (int i = 0; i < fc->nbFields; i++)
6      {
7          PluginField field = fc->fields[i];
8          std::string field_name(field.name);
9
10         if (field_name.compare("weight") == 0)
11         {
12             w.values = field.data;
13             w.count  = field.length;
14             w.type   = DataType::kFLOAT;
15             continue;
16         }
17         if (field_name.compare("k") == 0)
18         {
19             k = *reinterpret_cast<const int *>(field.data);
20         }
21         if (field_name.compare("n") == 0)
22         {
23             n = *reinterpret_cast<const int *>(field.data);
24         }
25     }
26     return new CuBLASGemmPlugin(name, w, k, n, true);
27 }
```

• Plugin 的关键 API

- Version 和 namespace 相关
  - 序列化时 TensorRT 会将 Plugin 的名字（Name）、类型（Type）、版本号（Version）、命名空间（Namespace）信息写入 engine
  - 通常不用修改

```
1   static const char *PLUGIN_NAME {"CuBLASGemm"};
2   static const char *PLUGIN_VERSION {"1"};
3
4   void CuBLASGemmPlugin::setPluginNamespace(const char *pluginNamespace) noexcept
5   {
6       namespace_ = pluginNamespace;
7   }
8   const char *CuBLASGemmPlugin::getPluginNamespace() const noexcept
9   {
10      return namespace_.c_str();
11  }
12
13  const char *CuBLASGemmPlugin::getPluginType() const noexcept
14  {
15      return PLUGIN_NAME;
16  }
17
18  const char *CuBLASGemmPlugin::getPluginVersion() const noexcept
19  {
20      return PLUGIN_VERSION;
21  }
22
23  void CuBLASGemmPluginCreator::setPluginNamespace(const char *pluginNamespace) noexcept
24  {
25      WHERE_AM_I()
26      namespace_ = pluginNamespace;
27  }
28
29  const char *CuBLASGemmPluginCreator::getPluginNamespace() const noexcept
30  {
31      WHERE_AM_I()
32      return namespace_.c_str();
33  }
34
35  const char *CuBLASGemmPluginCreator::getPluginName() const noexcept
36  {
37      WHERE_AM_I()
38      return PLUGIN_NAME;
39  }
40  const char *CuBLASGemmPluginCreator::getPluginVersion() const noexcept
41  {
42      WHERE_AM_I()
43      return PLUGIN_VERSION;
44  }
```

TensorRT 中导入 Plugin

- 基本步骤
  - 加载编译好的 .so
  - 在 Plugin Registry 里找到需要的 Plugin
  - 通过 Plugin Creator 构造需要的 Plugin
  - 将 Plugin 插入网络中（搭建网络情景）
  - Parser 自动识别（通过 Parser 加载模型场景）

```python
logger = trt.Logger(trt.Logger.ERROR)
trt.init_libnvinfer_plugins(logger, '')
ctypes.cdll.LoadLibrary("./CuBLASGemmPlugin.so")

...

weight = np.random.rand(k * n).astype(np.float32).reshape(k, n) * 2 - 1
plugin = None

for c in trt.get_plugin_registry().plugin_creator_list:
    if c.name == 'CuBLASGemm':
        parameterList = []
        parameterList.append(trt.PluginField("weight", np.float32(weight), trt.PluginFieldType.FLOAT32))
        parameterList.append(trt.PluginField("k", np.int32(weight.shape[0]), trt.PluginFieldType.INT32))
        parameterList.append(trt.PluginField("n", np.int32(weight.shape[1]), trt.PluginFieldType.INT32))
        plugin = c.create_plugin(c.name, trt.PluginFieldCollection(parameterList))
        break

pluginLayer = network.add_plugin_v2([inputTensor], plugin)
```

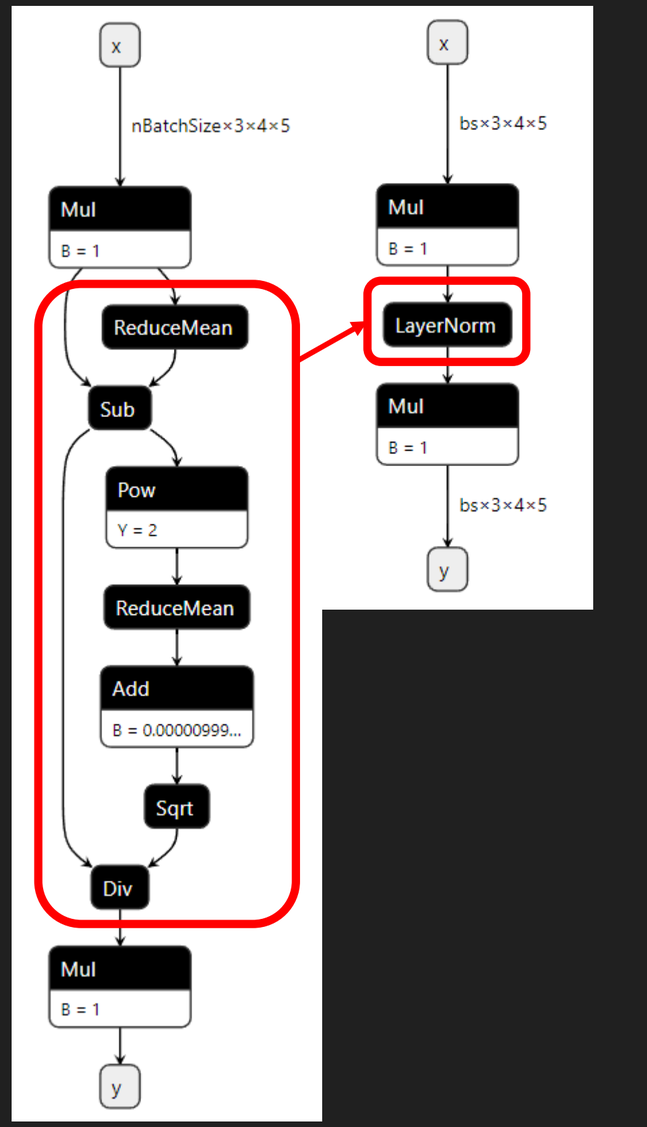
```python
logger = trt.Logger(trt.Logger.ERROR)
trt.init_libnvinfer_plugins(logger, '')
ctypes.cdll.LoadLibrary(soFile)

...

parser = trt.OnnxParser(network, logger)
with open(onnxFile, 'rb') as model:
    parser.parse(model.read())
```

• 结合使用 Parser 和 Plugin

- 范例代码
  - 06-PluginAndParser 目录中 TensorFlow-addScalar 和 pyTorch-LayerNorm
  - 实现 AddScalar Plugin / LayerNormalization Plugin 并替换原本 .onnx 中的相应部分

- 基本流程
  - Netron 分析原始 .onnx 文件中需要替换的模块
  - 使用 onnx-graphsurgeon 修改 .onnx 替换新节点
  - 实现相应的 Plugin 并做好单元测试
  - 在 TensorRT 中加载修改后的 .onnx 和 Plugin
  - 对比加载前后的计算精度和性能表现

- 加载 Plugin 失败时的报错信息
  - In node 2 (importFallbackPluginImporter): UNSUPPORTED_NODE: Assertion failed: creator && "Plugin not found, are the plugin name, version, and namespace correct?"

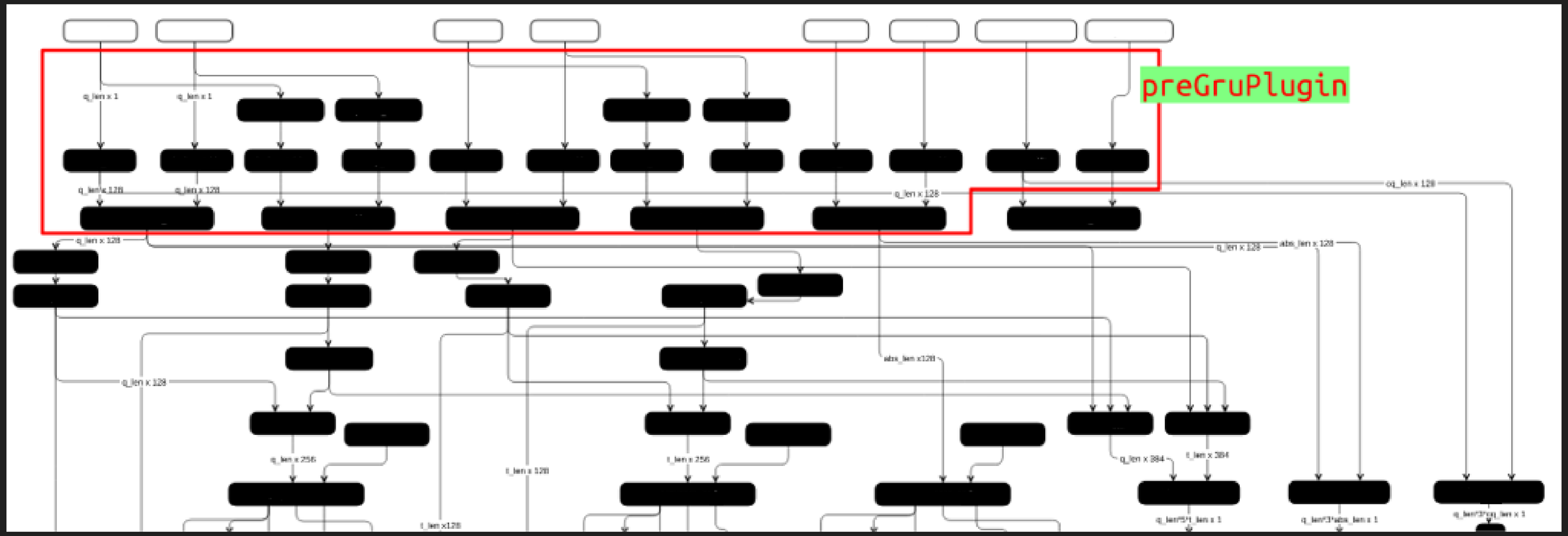


高级话题—— Plugin 的扩展性

- Plugin 与 FP16/Int8
  - 为了支持 FP16 模式，Plugin 要允许 float16 的输入输出张量类型，并实现 float16 的 CUDA C++ kernel
  - 为了支持 Int8 模式下的 calibration 过程，Plugin 需要实现 FP32 的支持，否则要手工指定所有输入输出张量的 dynamic_range
  - Int8 模式下，Plugin 内部张量的 dynamic_range 也要手工指定

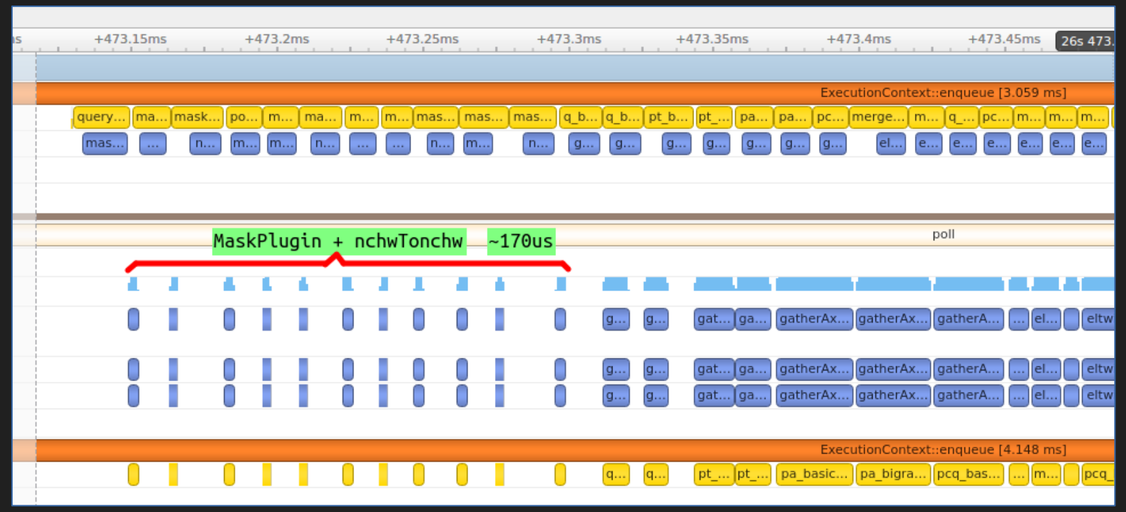
• 使用 Plugin 的例子

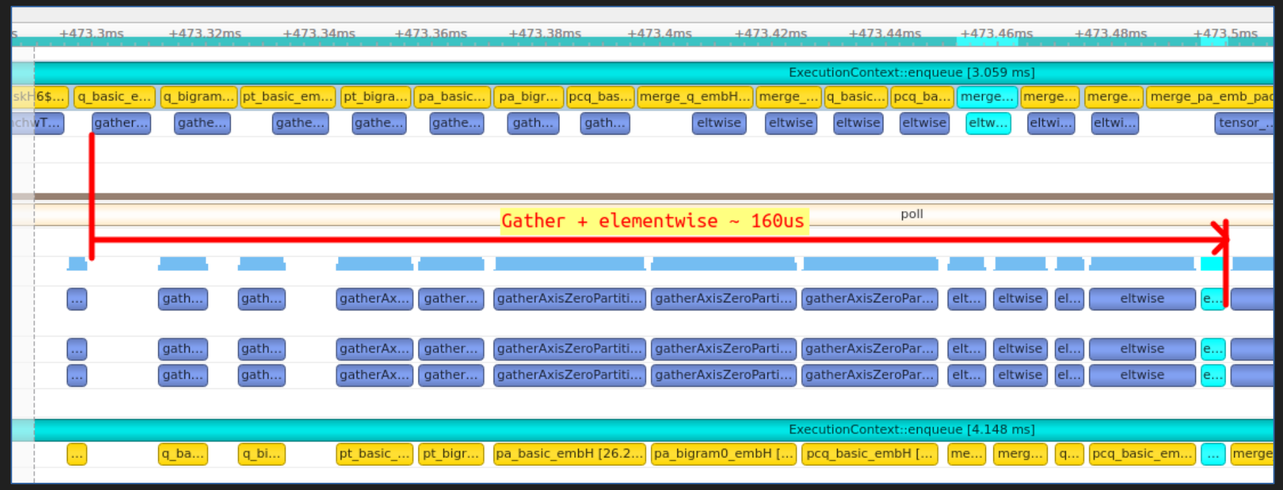
➢ 1、整合零散算子和 memory bound 的操作

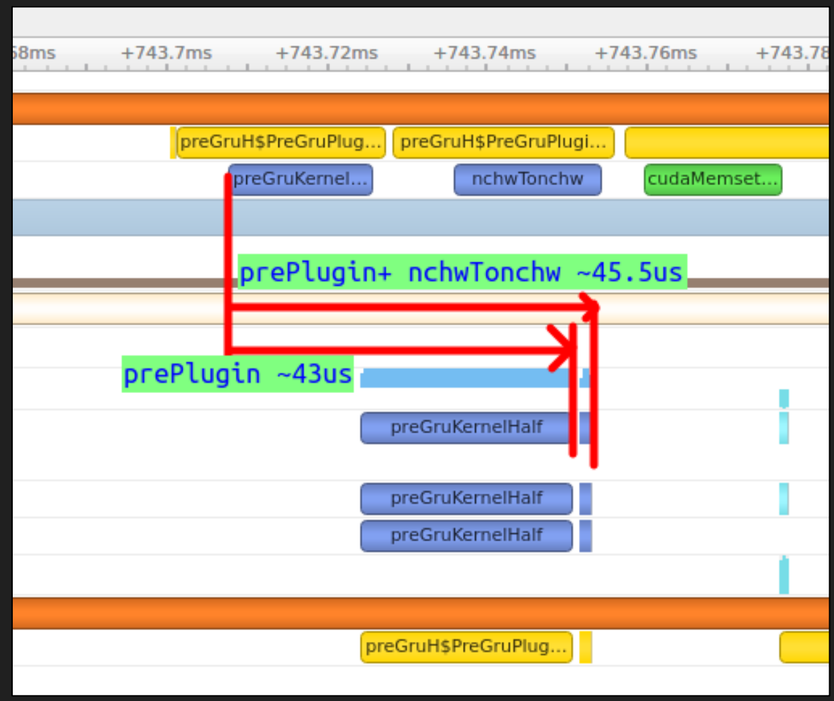


• 使用 Plugin 的例子

➢ 1、整合零散算子和 memory bound 的操作

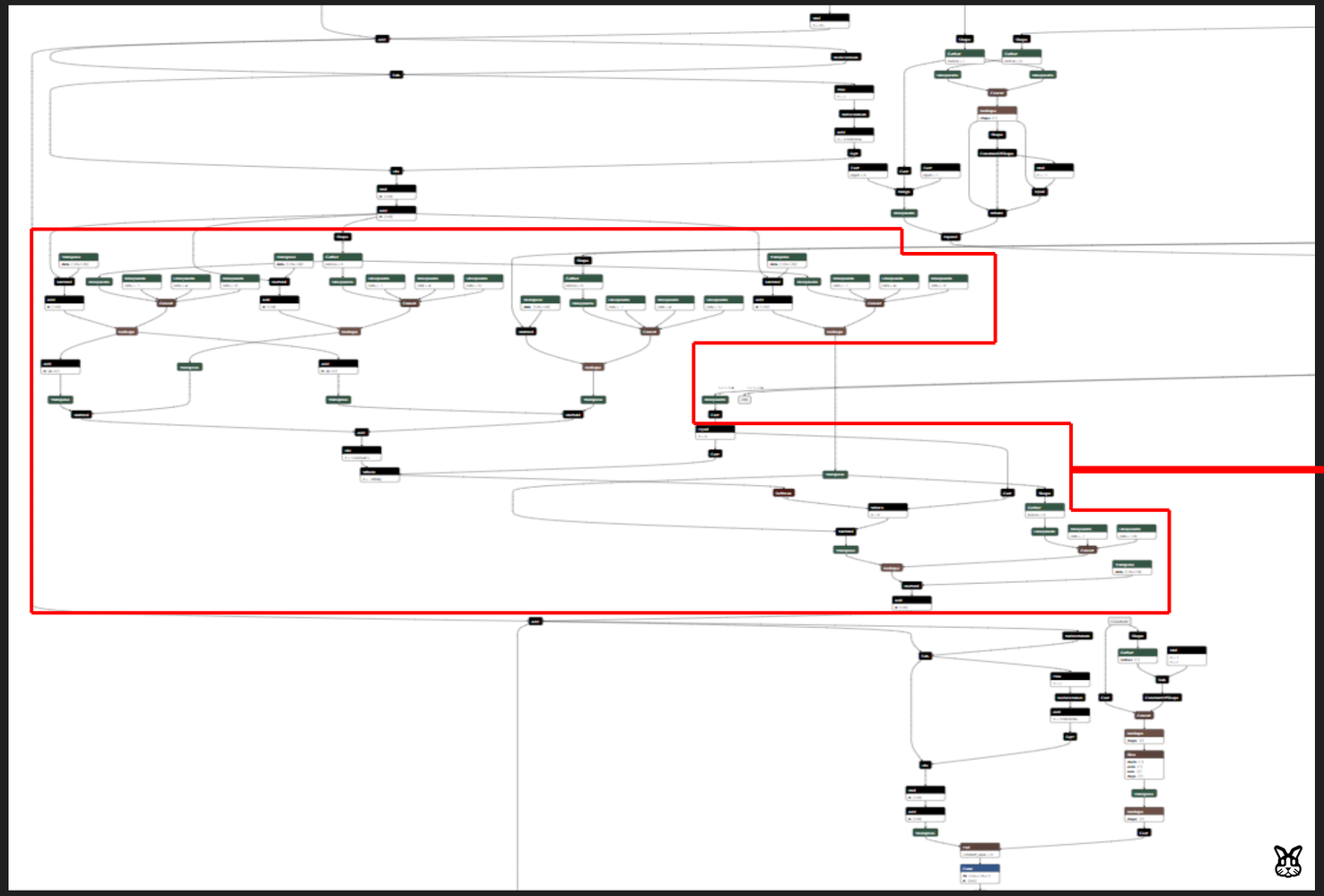


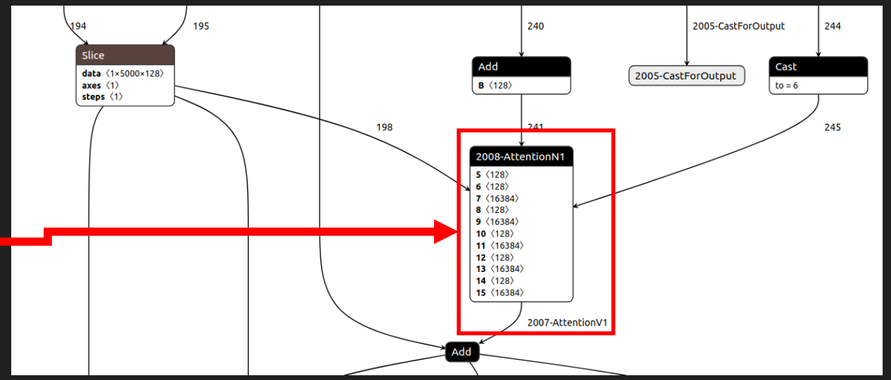




• 使用 Plugin 的例子

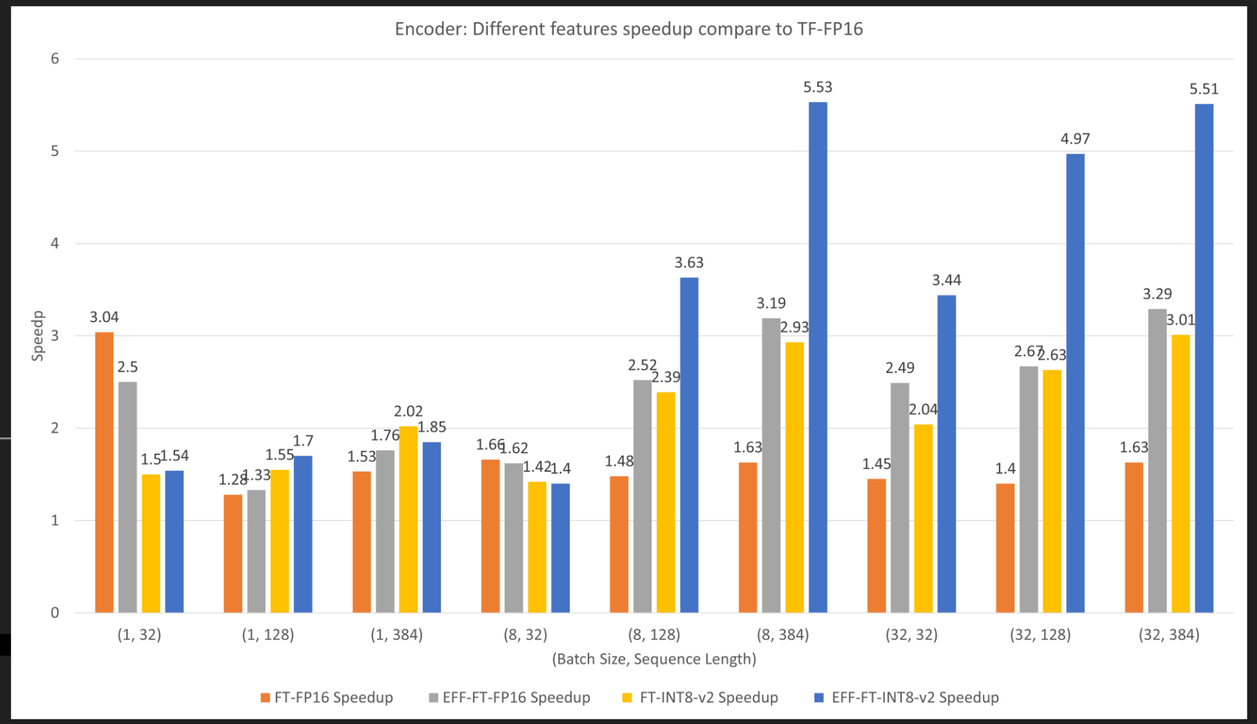
➢ 2、整合 Self-Attention 结构





• 使用 Plugin 的例子

➢ 3、融入其他高度优化的 CUDA kernel




```cpp
int T5EncoderPlugin::enqueue(const PluginTensorDesc *inputDesc,
                             const PluginTensorDesc *outputDesc,
                             const void *const *     inputs,
                             void *const *           outputs,
                             void *                  workspace,
                             cudaStream_t            stream) noexcept
{
    m_.batch_size = inputDesc[0].dims.d[0];
    m_.seq_len    = inputDesc[0].dims.d[1];

    cublasSetStream(cublasHandle_, stream);
    pCublasWrapper_->setStream(stream);

    std::unordered_map<std::string, Tensor> inputTensor {
        {"input_ids", Tensor {MEMORY_GPU, TYPE_INT32, std::vector<size_t> {m_.batch_size, m_.seq_len}, (int *)inputs[0]}},
        {"sequence_length", Tensor {MEMORY_GPU, TYPE_INT32, std::vector<size_t> {m_.batch_size}, (int *)inputs[1]}}};
    if (m_.useFP16)
    {
        std::unordered_map<std::string, Tensor> outputTensor {
            {"output_hidden_state",
             Tensor {MEMORY_GPU, TYPE_FP16, std::vector<size_t> {m_.batch_size, m_.seq_len, (size_t)(m_.head_num * m_.size_per_head)}, (half *)outputs[0]}}};
        pT5EncoderHalf_->setStream(stream);
        pT5EncoderHalf_->forward(&outputTensor, &inputTensor, pT5EncoderWeightHalf_);
    }
    else
    {
        std::unordered_map<std::string, Tensor> outputTensor {
            {"output_hidden_state",
             Tensor {MEMORY_GPU, TYPE_FP32, std::vector<size_t> {m_.batch_size, m_.seq_len, (size_t)(m_.head_num * m_.size_per_head)}, (float *)outputs[0]}}};
        pT5EncoderFloat_->setStream(stream);
        pT5EncoderFloat_->forward(&outputTensor, &inputTensor, pT5EncoderWeightFloat_);
    }
    return 0;
}
```

- https://github.com/NVIDIA/FasterTransformer

Part 4

高级话题

- 多 Optimization Profile
- 多 Context 与多 Stream
- CUDA Graph
- Timing Cache
- Algorithm Selector
- Refit
- Tactic Source
- 更多工具

高级话题

- 希望解决的问题
  - Dynamic Shape 模式在 min-opt-max 跨度较大时性能下降?
  - 怎样重叠计算和数据拷贝的时间，增加GPU利用率?
  - 怎样使一个 engine 供多个线程使用?
  - 怎么样优化 Kernel 的调用，减少 Launch Bound 的发生?
  - engine 构建时间太长，怎么节约多次构建的时间?
  - 某些 Layer 的算法导致较大的误差，能不能屏蔽掉该选择?
  - 想更新模型的权重，但又不想重新构建 engine?
  - 构建期/运行期显存占用过大，怎么减少?

高级话题：多 OptimizationProfile

- Dynamic Shape 模式在 min-opt-max 跨度较大时性能下降
- 解决方法：造多个 OptimizationProfile
- 范例代码
  - 09-Advance\MultiProfile
- 要点
  - 缩小每个 Profile 范围，方便 TensorRT 自动优化
  - 推理时，根据数据形状选择相应 Profile
  - 注意输入输出数据的绑定位置
- 间接作用
  - 多 Context 的基础工作
  - 增加显存占用、引擎尺寸和 .plan 尺寸

```python
profile0 = builder.create_optimization_profile()
profile1 = builder.create_optimization_profile()

inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, [-1, -1,- 1, -1])

profile0.set_shape(inputT0.name, (1, 1,  32,  48), (4, 2, 120, 180), ( 8,  4, 320,  480))
profile1.set_shape(inputT0.name, (1, 1, 560, 840), (4, 3, 640, 960), (16, 16, 1024, 1536))
config.add_optimization_profile(profile0)
config.add_optimization_profile(profile1)

...

# 使用 Profile 0
context.set_optimization_profile_async(0, stream)
cudart.cudaStreamSynchronize(stream)
#context.active_optimization_profile = 0  # 等价方法，depercated
context.set_binding_shape(0, [nIn, cIn, hIn, wIn])
print("Context binding all? %s" % ("Yes" if context.all_binding_shapes_specified else "No"))
for i in range(engine.num_bindings):
    print(i,
            "Input " if engine.binding_is_input(i) else "Output",
            engine.get_binding_shape(i),
            context.get_binding_shape(i),
            engine.get_binding_name(i))
...
context.execute_v2([int(inputD0), int(outputD0), int(0), int(0)])  # buffer 一共 4 个，放入前两个位置

# 使用 Profile 1
context.set_optimization_profile_async(1, stream)
cudart.cudaStreamSynchronize(stream)
context.set_binding_shape(2, [nIn, cIn, hIn, wIn])
print("Context binding all? %s" % ("Yes" if context.all_binding_shapes_specified else "No"))
for i in range(engine.num_bindings):
    print(i,
            "Input " if engine.binding_is_input(i) else "Output",
            engine.get_binding_shape(i),
            context.get_binding_shape(i),
            engine.get_binding_name(i))
...
context.execute_v2([int(0), int(0), int(inputD1), int(outputD1)])  # buffer 一共 4 个，放入后两个位置
```

• 高级话题：多 OptimizationProfile

- 输入输出绑定位置
  - 多出的 Profile 在绑定名末尾有“[Profile X]”
  - 推理时只需设定一组 profile 内的所有 input binding 就够了

```
Use Profile 0
Context binding all? Yes
0 Input  (-1, -1, -1, -1) (4, 2, 120, 180) inputT0
1 Output (-1, -1, -1, -1) (4, 2, 120, 180) outputT0
2 Input  (-1, -1, -1, -1) (-1, -1, -1, -1) inputT0 [profile 1]
3 Output (-1, -1, -1, -1) (-1, -1, -1, -1) outputT0 [profile 1]
check result: True

Use Profile 1
Context binding all? Yes
0 Input  (-1, -1, -1, -1) (-1, -1, -1, -1) inputT0
1 Output (-1, -1, -1, -1) (-1, -1, -1, -1) outputT0
2 Input  (-1, -1, -1, -1) (4, 3, 640, 960)  inputT0 [profile 1]
3 Output (-1, -1, -1, -1) (4, 3, 640, 960) outputT0 [profile 1]
check result: True
```

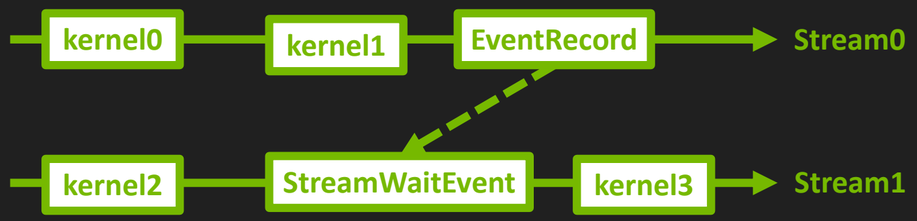
```python
profile0 = builder.create_optimization_profile()
profile1 = builder.create_optimization_profile()

inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, [-1, -1,- 1, -1])

profile0.set_shape(inputT0.name, (1, 1,  32,  48), (4, 2, 120, 180), ( 8,  4, 320,  480))
profile1.set_shape(inputT0.name, (1, 1, 560, 840), (4, 3, 640, 960), (16, 16, 1024, 1536))
config.add_optimization_profile(profile0)
config.add_optimization_profile(profile1)

...

# 使用 Profile 0
context.set_optimization_profile_async(0, stream)
cudart.cudaStreamSynchronize(stream)
#context.active_optimization_profile = 0  # 等价方法，depercated
context.set_binding_shape(0, [nIn, cIn, hIn, wIn])
print("Context binding all? %s" % ("Yes" if context.all_binding_shapes_specified else "No"))
for i in range(engine.num_bindings):
    print(i,
            "Input " if engine.binding_is_input(i) else "Output",
            engine.get_binding_shape(i),
            context.get_binding_shape(i),
            engine.get_binding_name(i))
...
context.execute_v2([int(inputD0), int(outputD0), int(0), int(0)])  # buffer 一共 4 个，放入前两个位置

# 使用 Profile 1
context.set_optimization_profile_async(1, stream)
cudart.cudaStreamSynchronize(stream)
context.set_binding_shape(2, [nIn, cIn, hIn, wIn])
print("Context binding all? %s" % ("Yes" if context.all_binding_shapes_specified else "No"))
for i in range(engine.num_bindings):
    print(i,
            "Input " if engine.binding_is_input(i) else "Output",
            engine.get_binding_shape(i),
            context.get_binding_shape(i),
            engine.get_binding_name(i))
...
context.execute_v2([int(0), int(0), int(inputD1), int(outputD1)])  # buffer 一共 4 个，放入后两个位置
```

高级话题

- 希望解决的问题
  - Dynamic Shape 模式在 min-opt-max 跨度较大时性能下降?
  - 怎样重叠计算和数据拷贝的时间，增加GPU利用率?
  - 怎样使一个 engine 供多个线程使用?
  - 怎么样优化 Kernel 的调用，减少 Launch Bound 的发生?
  - engine 构建时间太长，怎么节约多次构建时的时间?
  - 某些 Layer 的算法导致较大的误差，能不能屏蔽掉该选择?
  - 想更新模型的权重，但又不想重新构建 engine?
  - 构建期/运行期显存占用过大，怎么减少?

• 高级话题：多 Steam

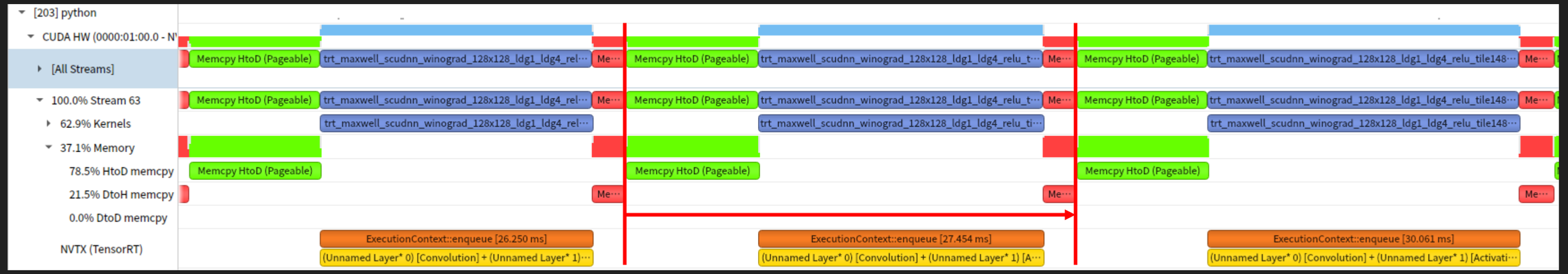
- CUDA 编程经典话题：重叠计算和数据拷贝
- 解决方法：恰当使用 Stream
- 范例代码：
  - 09-Advance\MultiStream
- 要点
  - 使用 CUDA event 和 CUDA stream
  - 使用 pinned-memory
  - 使用 *Async 函数和 context.execute_async_v2 方法



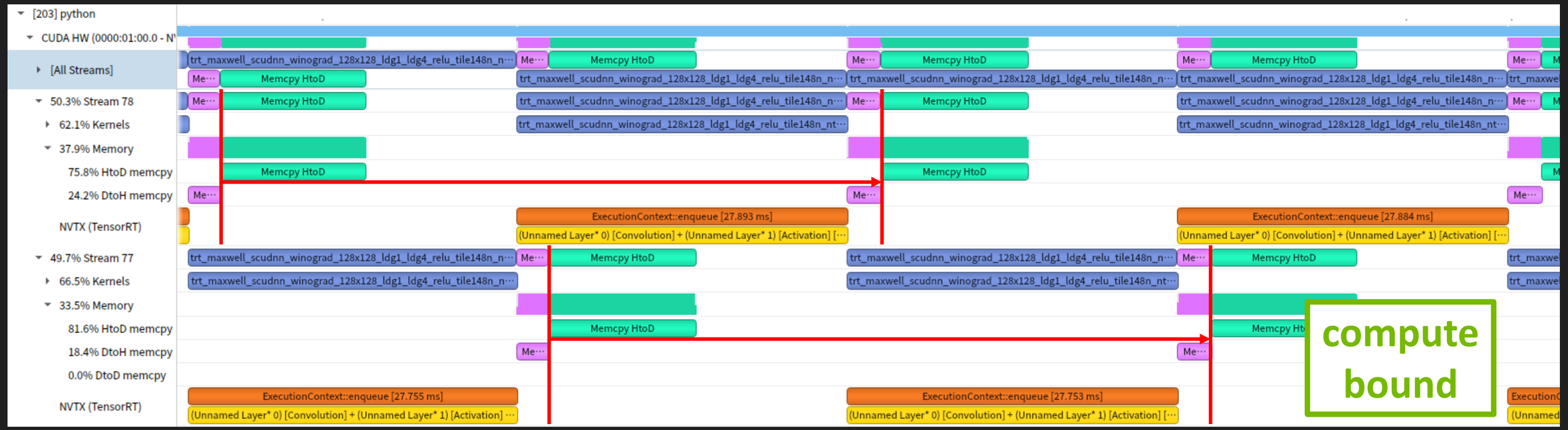

```
1   context = engine.create_execution_context()
2   context.set_binding_shape(0, [nIn, cIn, hIn, wIn])
3   _, stream0 = cudart.cudaStreamCreate()
4   _, stream1 = cudart.cudaStreamCreate()
5   _, event0 = cudart.cudaEventCreate()
6   _, event1 = cudart.cudaEventCreate()
7
8   ...
9
10  cudart.cudaEventRecord(event1, stream1)
11  for i in range(30//2):
12      cudart.cudaMemcpyAsync(inputD0, inputH0, inputSize,
13                             cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream0)
14      cudart.cudaStreamWaitEvent(stream0,event1,cudart.cudaEventWaitDefault)
15      context.execute_async_v2([int(inputD0), int(outputD0)], stream0)
16      cudart.cudaEventRecord(event0,stream0)
17      cudart.cudaMemcpyAsync(outputH0, outputD0, outputSize,
18                             cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream0)
19
20      cudart.cudaMemcpyAsync(inputD1, inputH1, inputSize,
21                             cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream1)
22      cudart.cudaStreamWaitEvent(stream1,event0,cudart.cudaEventWaitDefault)
23      context.execute_async_v2([int(inputD1), int(outputD1)], stream1)
24      cudart.cudaEventRecord(event1,stream1)
25      cudart.cudaMemcpyAsync(outputH1, outputD1, outputSize,
26                             cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream1)
27
28  cudart.cudaEventSynchronize(event1)
```

高级话题：多 Stream

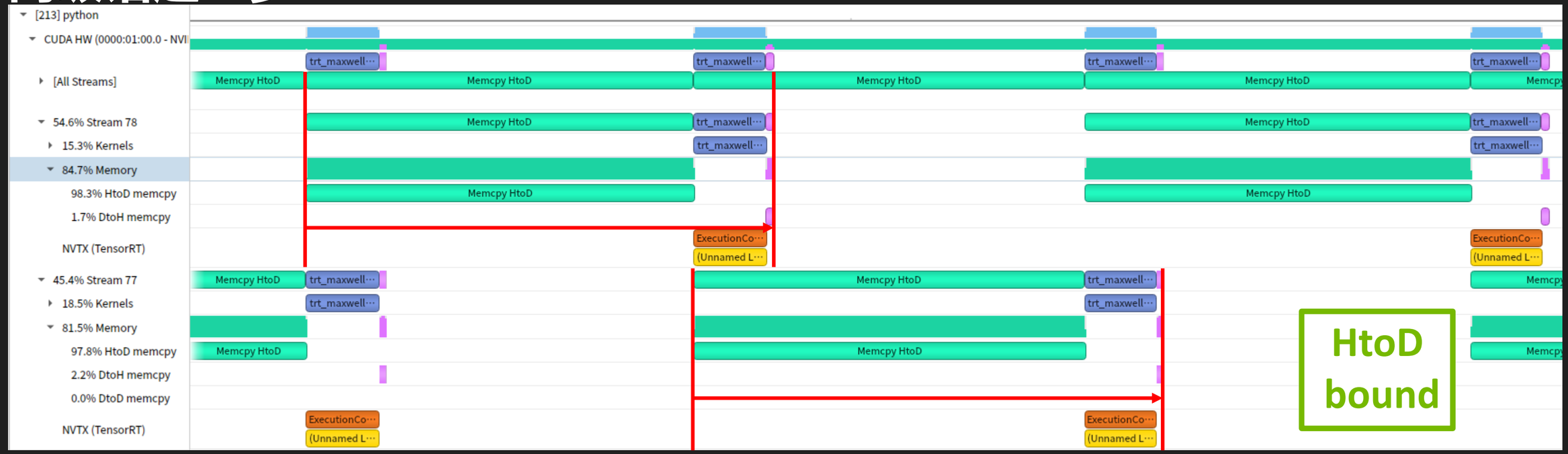
- 使用单 Stream 和同步函数调用

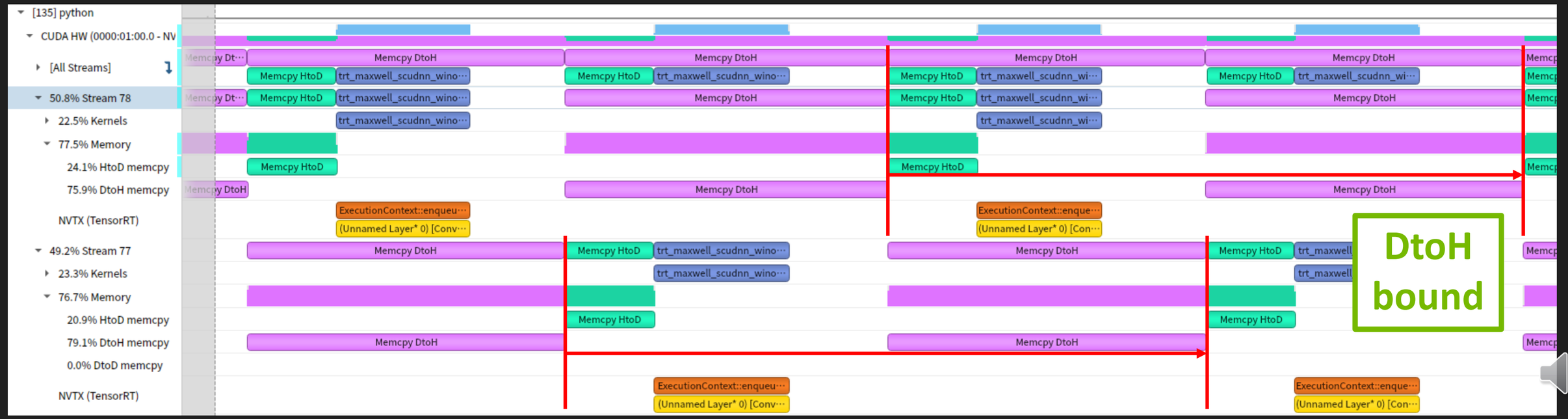


- 使用双 Stream 和异步函数调用



- 高级话题：多 Stream





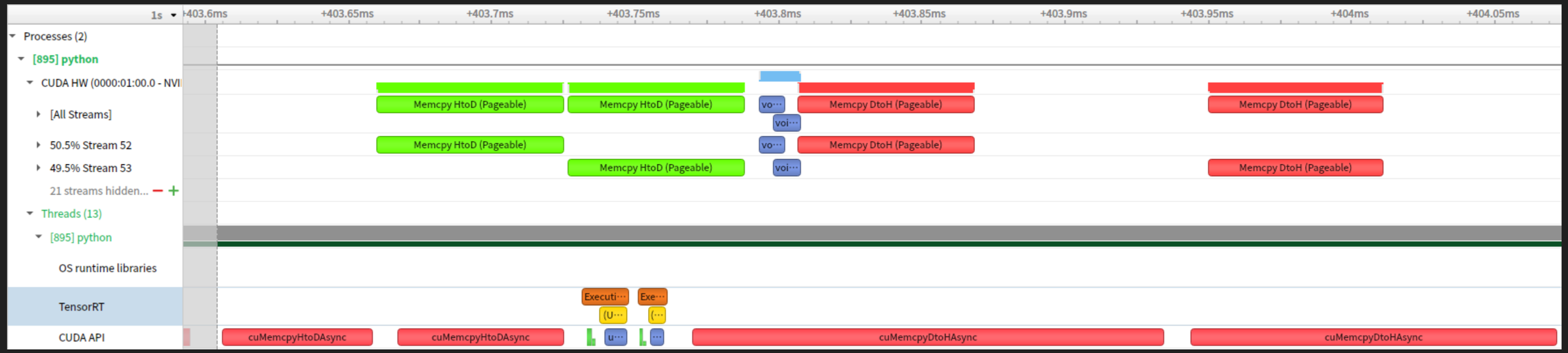
高级话题

- 希望解决的问题
  - Dynamic Shape 模式在 min-opt-max 跨度较大时性能下降?
  - 怎样重叠计算和数据拷贝的时间，增加GPU利用率?
  - 怎样使一个 engine 供多个线程使用?
  - 怎么样优化 Kernel 的调用，减少 Launch Bound 的发生?
  - engine 构建时间太长，怎么节约多次构建时的时间?
  - 某些 Layer 的算法导致较大的误差，能不能屏蔽掉该选择?
  - 想更新模型的权重，但又不想重新构建 engine?
  - 构建期/运行期显存占用过大，怎么减少?

高级话题：多 Context

- 一个Engine 供多个 Context 共用
- 范例代码
  - 09-Advance\MultiContext
- 要点
  - 多 Profile
  - 从 engine 中创建多个 context，各 context 指定不同的 profile

```python
stream0 = cudart.cudaStreamCreate()[1]
stream1 = cudart.cudaStreamCreate()[1]
context0 = engine.create_execution_context()
context1 = engine.create_execution_context()
context0.set_optimization_profile_async(0, stream0)
context1.set_optimization_profile_async(1, stream1)
context0.set_binding_shape(0, [nIn, cIn, hIn, wIn])
context1.set_binding_shape(2, [nIn, cIn, hIn, wIn])
print("Context0 binding all? %s" % (["No", "Yes"][int(context0.all_binding_shapes_specified)]))
print("Context1 binding all? %s" % (["No", "Yes"][int(context1.all_binding_shapes_specified)]))
for i in range(engine.num_bindings):
    print(i,
          "Input " if engine.binding_is_input(i) else "Output",
          engine.get_binding_shape(i),
          context0.get_binding_shape(i),
          context1.get_binding_shape(i))
...
context0.execute_async_v2([int(inputD0), int(outputD0), int(0), int(0)], stream0)
context1.execute_async_v2([int(0), int(0), int(inputD1), int(outputD1)], stream1)
```



高级话题

- 希望解决的问题
  - Dynamic Shape 模式在 min-opt-max 跨度较大时性能下降?
  - 怎样重叠计算和数据拷贝的时间，增加GPU利用率?
  - 怎样使一个 engine 供多个线程使用?
  - 怎么样优化 Kernel 的调用，减少 Launch Bound 的发生?
  - engine 构建时间太长，怎么节约多次构建时的时间?
  - 某些 Layer 的算法导致较大的误差，能不能屏蔽掉该选择?
  - 想更新模型的权重，但又不想重新构建 engine?
  - 构建期/运行期显存占用过大，怎么减少?

高级话题：CUDA Graph

- 利用 CUDA Graph
  - https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#cuda-graphs
- 范例代码
  - 09-Advance\CudaGraph

```python
# 首次捕获 CUDA Graph 并运行
cudart.cudaStreamBeginCapture(stream, cudart.cudaStreamCaptureMode.cudaStreamCaptureModeGlobal)
cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
                       cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(outputD0)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
                       cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
#cudart.cudaStreamSynchronize(stream)  # 不用在 graph 内同步
_, graph = cudart.cudaStreamEndCapture(stream)
_, graphExe, _ = cudart.cudaGraphInstantiate(graph, b"", 0)

cudart.cudaGraphLaunch(graphExe, stream)  # 每次准备好 buffer 后用 cudaGraphLaunch 运行
cudart.cudaStreamSynchronize(stream)

# 输入尺寸改变后，需要首先运行一次推理，然后重新捕获 CUDA Graph，最后再运行
context.set_binding_shape(0, [2, 3, 4])
...
context.execute_async_v2([int(inputD0), int(outputD0)], stream)
...
cudart.cudaStreamSynchronize(stream)

cudart.cudaStreamBeginCapture(stream, cudart.cudaStreamCaptureMode.cudaStreamCaptureModeGlobal)
cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
                       cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(outputD0)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
                       cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
_, graph = cudart.cudaStreamEndCapture(stream)
_, graphExe, _ = cudart.cudaGraphInstantiate(graph, b"", 0)

cudart.cudaGraphLaunch(graphExe, stream)
cudart.cudaStreamSynchronize(stream)
```

- 优点：
  - 降低 CPU Launch cost
  - CUDA 工作流优化
  - 缓解大量 kernel 调用时的 Launch Bound
- 要点
  - 步骤：Graph 定义， Graph 实例化， Graph 执行
  - Dynamic Shape 模式中，实际数据形状发生改变时（调用 context.set_binding_shape），

    要先跑一遍 context.execute 再重新捕获 Graph，最后再实例化和执行 Graph

• 高级话题：CUDA Graph

高级话题

- 希望解决的问题
  - Dynamic Shape 模式在 min-opt-max 跨度较大时性能下降?
  - 怎样重叠计算和数据拷贝的时间，增加GPU利用率?
  - 怎样使一个 engine 供多个线程使用?
  - 怎么样优化 Kernel 的调用，减少 Launch Bound 的发生?
  - engine 构建时间太长，怎么节约多次构建时的时间?
  - 某些 Layer 的算法导致较大的误差，能不能屏蔽掉该选择?
  - 想更新模型的权重，但又不想重新构建 engine?
  - 构建期/运行期显存占用过大，怎么减少?

高级话题：Timing Cache

- 范例代码：
  - 09-Advance\TimingCache

```python
timeCache = b""
if useTimeCache and os.path.isfile(timeCacheFile):
    with open(timeCacheFile, 'rb') as f:
        timeCache = f.read()

...  # 建立 Builder 和 BbuilderConfig

if useTimeCache:
    cache = config.create_timing_cache(timeCache)
    config.set_timing_cache(cache, False)

...  # 建立 Network 和 SerializedNetwork

if useTimeCache and not os.path.isfile(timeCacheFile):
    timeCache = config.get_timing_cache()
    timeCacheString = timeCache.serialize()
    with open(timeCacheFile, 'wb') as f:
        f.write(timeCacheString)
```

- 优点：
  - 优化单次引擎构建时间（模型内多个同参数的算子）
  - 优化多次引擎构建时间（debug、参数更新后重新构建）
  - 优化同环境下多个引擎构建时间（跨 builder 可用）
  - 用于反复生成一模一样的引擎

```python
run(0)  # 不使用 Timing Cache
run(1)  # 创建并保存 Timing Cache
run(1)  # 读取并使用 Timing Cache
```

```
Without timing cache, 1707.987785 ms
With timing cache, 724.868298 ms
Succeeded saving .cache file!
Succeeded getting serialized timing cache!
With timing cache, 32.098293 ms
```

- 要点
  - 类似引擎序列化反序列化，将 Timing Cache 保存出来下次用
  - 类似 .plan，不可跨平台和开发环境使用

• 高级话题：Timing Cache

创建并保存 Timing Cache

```
[V] Applying generic optimizations to the graph for inference.

...

[V] =============== Computing reformatting costs
[V] ************** Autotuning Reformat: Float(784,784,28,1) -> Float(784,1,28,1) *
[V] -------------- Timing Runner: Optimizer Reformat(inputT0 -> <out>) (Reformat)
[V] Tactic: 1002 Time: 0.00512
[V] Tactic: 0 Time: 0.003072
[V] Fastest Tactic: 0 Time: 0.003072
[V] =============== Computing reformatting costs
[V] ************** Autotuning Reformat: Float(25088,1,896,32) -> Float(25088,784,2
[V] -------------- Timing Runner: Optimizer Reformat((Unnamed Layer* 1) [Activatio
[V] Tactic: 1002 Time: 0.01536
[V] Tactic: 0 Time: 0.008896
[V] Fastest Tactic: 0 Time: 0.008896

...

[V] Formats and tactics selection completed in 1.28525 seconds.

...

Layer(CaskConvolution): (Unnamed Layer* 0) [Convolution] + (Unnamed Layer* 1) [Acti
Layer(TiledPooling): (Unnamed Layer* 2) [Pooling], Tactic: 6357249, (Unnamed Layer*
Layer(Reformat): Reformatting CopyNode for Input Tensor 0 to (Unnamed Layer* 3) [Co
Layer(CaskConvolution): (Unnamed Layer* 3) [Convolution] + (Unnamed Layer* 4) [Acti
Layer(Reformat): Reformatting CopyNode for Input Tensor 0 to (Unnamed Layer* 5) [Po
Layer(TiledPooling): (Unnamed Layer* 5) [Pooling], Tactic: 6357249, Reformatted Inp
Layer(Shuffle): (Unnamed Layer* 6) [Shuffle], Tactic: 0, (Unnamed Layer* 5) [Poolin
Layer(CublasConvolution): (Unnamed Layer* 7) [Fully Connected] + (Unnamed Layer* 8)
Layer(CublasConvolution): (Unnamed Layer* 9) [Fully Connected] + (Unnamed Layer* 10
Layer(NoOp): (Unnamed Layer* 11) [Shuffle], Tactic: 0, (Unnamed Layer* 10) [Activat
Layer(LogSoftmaxTopK): (Unnamed Layer* 12) [Softmax] + (Unnamed Layer* 13) [TopK],

...

[V] Deleting timing cache: 31 entries, 0 hits
Succeeded getting serialized engine!
Succeeded saving .cache file!
Succeeded building engine!
```

读取并使用 Timing Cache

```
[V] Loaded 31 timing cache entries.
[V] Applying generic optimizations to the graph for inference.

...

[V] =============== Computing reformatting costs
[V] ************** Autotuning Reformat: Float(784,784,28,1) -> Float(784,1,28,1) *



[V] =============== Computing reformatting costs
[V] ************** Autotuning Reformat: Float(25088,1,896,32) -> Float(25088,784,2

...

[V] Formats and tactics selection completed in 0.014933 seconds.

...

Layer(CaskConvolution): (Unnamed Layer* 0) [Convolution] + (Unnamed Layer* 1) [Acti
Layer(TiledPooling): (Unnamed Layer* 2) [Pooling], Tactic: 6226177, (Unnamed Layer*
Layer(Reformat): Reformatting CopyNode for Input Tensor 0 to (Unnamed Layer* 3) [Co
Layer(CaskConvolution): (Unnamed Layer* 3) [Convolution] + (Unnamed Layer* 4) [Acti
Layer(Reformat): Reformatting CopyNode for Input Tensor 0 to (Unnamed Layer* 5) [Po
Layer(TiledPooling): (Unnamed Layer* 5) [Pooling], Tactic: 6488321, Reformatted Inp
Layer(Shuffle): (Unnamed Layer* 6) [Shuffle], Tactic: 0, (Unnamed Layer* 5) [Poolin
Layer(CublasConvolution): (Unnamed Layer* 7) [Fully Connected] + (Unnamed Layer* 8)
Layer(CublasConvolution): (Unnamed Layer* 9) [Fully Connected] + (Unnamed Layer* 10
Layer(NoOp): (Unnamed Layer* 11) [Shuffle], Tactic: 0, (Unnamed Layer* 10) [Activat
Layer(LogSoftmaxTopK): (Unnamed Layer* 12) [Softmax] + (Unnamed Layer* 13) [TopK],

...

[V] Deleting timing cache: 31 entries, 31 hits
Succeeded getting serialized timing cache!
Succeeded getting serialized engine!
Succeeded building engine!
```

高级话题

- 希望解决的问题
  - Dynamic Shape 模式在 min-opt-max 跨度较大时性能下降?
  - 怎样重叠计算和数据拷贝的时间，增加GPU利用率?
  - 怎样使一个 engine 供多个线程使用?
  - 怎么样优化 Kernel 的调用，减少 Launch Bound 的发生?
  - engine 构建时间太长，怎么节约多次构建时的时间?
  - 某些 Layer 的算法导致较大的误差，能不能精确选择算法?
  - 想更新模型的权重，但又不想重新构建 engine?
  - 构建期/运行期显存占用过大，怎么减少?

高级话题：Algorithm Selector

- 样例代码
  - 09-Advance\AlgorithmSelector

```python
class MyAlgorithmSelector(trt.IAlgorithmSelector):

    def __init__(self, keepAll=True):
        super(MyAlgorithmSelector, self).__init__()
        self.keepAll = keepAll

    def select_algorithms(self, layerAlgorithmContext, layerAlgorithmList):

        # 构建 SerializedNetwork 时被调用
        # 输入一个 Layer（layerAlgorithmContext）和一个算法列表（layerAlgorithmList）
        # 返回一个从 0 开始的索引列表，表示上述算法列表中哪些可以保留

    def report_algorithms(self, modelAlgorithmContext, modelAlgorithmList):

        # 构建 SerializedNetwork 完成后被调用
        # 输入网络所有 Layer（modelAlgorithmContext）和各 Layer 保留的算法列表（modelAlgorithmList）
        # 没有返回

... # 创建 Builder 和BuilderConfig
config.algorithm_selector = MyAlgorithmSelector(True)  # 设置算法选择器
```

- 要点
  - 自己实现一个 MyAlgorithmSelector 类
  - 关键是实现两个成员函数
    - 一个用来挑选特定层的算法
    - 一个用来报告所有层的挑选结果
  - 构建网络时交给 BuilderConfig

- 实际工作流程：
  - 先通过 polygraphy 等工具发现某层的个 Tactic 结果不理想，
  - 构造 Algorithm Selector 屏蔽掉盖层的该 tactic
  - 构建引擎

• 高级话题：Algorithm Selector

```python
class MyAlgorithmSelector(trt.IAlgorithmSelector):

    def __init__(self, keepAll=True):
        super(MyAlgorithmSelector, self).__init__()
        self.keepAll = keepAll

    def select_algorithms(self, layerAlgorithmContext, layerAlgorithmList):
        if self.keepAll:  # 保留全部选择
            return list((range(len(layerAlgorithmList))))

        else:  # 手工筛选算法
            # 选择计算时间最长的算法
            timeList = [algorithm.timing_msec for algorithm in layerAlgorithmList]
            return list(np.argmax(timeList))

            # 选择 workspace 最小的算法
            workspaceSizeList = [ algorithm.workspace_size for algorithm in layerAlgorithmList ]
            return list(np.argmin(workspaceSizeList))

            # 让特定层选择特定算法（用于多次构建一模一样的引擎）
            if layerAlgorithmContext.name == "(Unnamed Layer* 0) [Convolution] + (Unnamed Layer* 1) [Activation]":
                # 最后的数字来自 VERBOSE 日志中信息，代表该层的一种实现
                return [ index for index,algorithm in enumerate(layerAlgorithmList)
                            if algorithm.algorithm_varient.implementation == 2147483648 ]

            return []

    def report_algorithms(self, modelAlgorithmContext, modelAlgorithmList):
        for i in range(len(modelAlgorithmContext)):
            context = modelAlgorithmContext[i]
            algorithm = modelAlgorithmList[i]

            print("Layer%4d:%s" % (i, context.name))
            nInput = context.num_inputs
            nOutput = context.num_outputs
            for j in range(nInput):
                ioInfo = algorithm.get_algorithm_io_info(j)
                print("    Input [%2d]:%s,%s,%s,%s" % (j, context.get_shape(j), ioInfo.dtype, ioInfo.strides, ioInfo.tensor_format))
            for j in range(nOutput):
                ioInfo = algorithm.get_algorithm_io_info(j + nInput)
                print("    Output[%2d]:%s,%s,%s,%s" % (j, context.get_shape(j + nInput), ioInfo.dtype, ioInfo.strides, ioInfo.tensor_format))
            print("    algorithm:[implementation:%d,tactic:%d,timing:%fms,workspace:%dMB]"% \
                  (algorithm.algorithm_varient.implementation,
                   algorithm.algorithm_varient.tactic,
                   algorithm.timing_msec,
                   algorithm.workspace_size))
```

• 高级话题：Algorithm Selector

```
Layer   0:(Unnamed Layer* 0) [Convolution] + (Unnamed Layer* 1) [Activation]
    Input [ 0]:[(1, 1, 28, 28), (1, 1, 28, 28), (2, 1, 28, 28)],DataType.FLOAT,(784, 784, 28, 1),TensorFormat.LINEAR
    Output[ 0]:[(1, 32, 28, 28), (1, 32, 28, 28), (2, 32, 28, 28)],DataType.FLOAT,(25088, 784, 28, 1),TensorFormat.LINEAR
    algorithm:[implementation:2147483649,tactic:9306111,timing:0.000000ms,workspace:0MB]
Layer   1:(Unnamed Layer* 2) [Pooling]
    Input [ 0]:[(1, 32, 28, 28), (1, 32, 28, 28), (2, 32, 28, 28)],DataType.FLOAT,(25088, 784, 28, 1),TensorFormat.LINEAR
    Output[ 0]:[(1, 32, 14, 14), (1, 32, 14, 14), (2, 32, 14, 14)],DataType.FLOAT,(6272, 196, 14, 1),TensorFormat.LINEAR
    algorithm:[implementation:2147483679,tactic:6095105,timing:0.000000ms,workspace:0MB]
Layer   2:(Unnamed Layer* 3) [Convolution] + (Unnamed Layer* 4) [Activation]
    Input [ 0]:[(1, 32, 14, 14), (1, 32, 14, 14), (2, 32, 14, 14)],DataType.FLOAT,(6272, 196, 14, 1),TensorFormat.LINEAR
    Output[ 0]:[(1, 64, 14, 14), (1, 64, 14, 14), (2, 64, 14, 14)],DataType.FLOAT,(12544, 196, 14, 1),TensorFormat.LINEAR
    algorithm:[implementation:2147483649,tactic:8060927,timing:0.000000ms,workspace:0MB]
Layer   3:(Unnamed Layer* 5) [Pooling]
    Input [ 0]:[(1, 64, 14, 14), (1, 64, 14, 14), (2, 64, 14, 14)],DataType.FLOAT,(12544, 196, 14, 1),TensorFormat.LINEAR
    Output[ 0]:[(1, 64, 7, 7), (1, 64, 7, 7), (2, 64, 7, 7)],DataType.FLOAT,(3136, 49, 7, 1),TensorFormat.LINEAR
    algorithm:[implementation:2147483679,tactic:6357249,timing:0.000000ms,workspace:0MB]
Layer   4:(Unnamed Layer* 6) [Shuffle]
    Input [ 0]:[(1, 64, 7, 7), (1, 64, 7, 7), (2, 64, 7, 7)],DataType.FLOAT,(3136, 49, 7, 1),TensorFormat.LINEAR
    Output[ 0]:[(1, 3136, 1, 1), (1, 3136, 1, 1), (2, 3136, 1, 1)],DataType.FLOAT,(3136, 1, 1, 1),TensorFormat.LINEAR
    algorithm:[implementation:2147483661,tactic:0,timing:0.000000ms,workspace:0MB]
Layer   5:(Unnamed Layer* 7) [Fully Connected] + (Unnamed Layer* 8) [Activation]
    Input [ 0]:[(1, 3136, 1, 1), (1, 3136, 1, 1), (2, 3136, 1, 1)],DataType.FLOAT,(3136, 1, 1, 1),TensorFormat.LINEAR
    Output[ 0]:[(1, 1024, 1, 1), (1, 1024, 1, 1), (2, 1024, 1, 1)],DataType.FLOAT,(1024, 1, 1, 1),TensorFormat.LINEAR
    algorithm:[implementation:2147483692,tactic:0,timing:0.000000ms,workspace:0MB]
Layer   6:(Unnamed Layer* 9) [Fully Connected] + (Unnamed Layer* 10) [Activation]
    Input [ 0]:[(1, 1024, 1, 1), (1, 1024, 1, 1), (2, 1024, 1, 1)],DataType.FLOAT,(1024, 1, 1, 1),TensorFormat.LINEAR
    Output[ 0]:[(1, 10, 1, 1), (1, 10, 1, 1), (2, 10, 1, 1)],DataType.FLOAT,(10, 1, 1, 1),TensorFormat.LINEAR
    algorithm:[implementation:2147483692,tactic:1,timing:0.000000ms,workspace:0MB]
Layer   7:(Unnamed Layer* 11) [Shuffle]
    Input [ 0]:[(1, 10, 1, 1), (1, 10, 1, 1), (2, 10, 1, 1)],DataType.FLOAT,(10, 1, 1, 1),TensorFormat.LINEAR
    Output[ 0]:[(1, 10), (1, 10), (2, 10)],DataType.FLOAT,(10, 1),TensorFormat.LINEAR
    algorithm:[implementation:2147483661,tactic:0,timing:0.000000ms,workspace:0MB]
Layer   8:(Unnamed Layer* 12) [Softmax] + (Unnamed Layer* 13) [TopK]
    Input [ 0]:[(1, 10), (1, 10), (2, 10)],DataType.FLOAT,(10, 1),TensorFormat.LINEAR
    Output[ 0]:[(1, 1), (1, 1), (2, 1)],DataType.FLOAT,(1, 1),TensorFormat.LINEAR
    Output[ 1]:[(1, 1), (1, 1), (2, 1)],DataType.INT32,(1, 1),TensorFormat.LINEAR
    algorithm:[implementation:2147483685,tactic:1001,timing:0.000000ms,workspace:0MB]
Succeeded getting serialized engine!
Succeeded building engine!
Bind[ 0]:i[ 0]-> DataType.FLOAT (-1, 1, 28, 28) (1, 1, 28, 28) inputT0
Bind[ 1]:o[ 0]-> DataType.INT32 (-1, 1) (1, 1) (Unnamed Layer* 13) [TopK]_output_2
inputT0
(Unnamed Layer* 13) [TopK]_output_2
```

高级话题

- 希望解决的问题
  - Dynamic Shape 模式在 min-opt-max 跨度较大时性能下降?
  - 怎样重叠计算和数据拷贝的时间，增加GPU利用率?
  - 怎样使一个 engine 供多个线程使用?
  - 怎么样优化 Kernel 的调用，减少 Launch Bound 的发生?
  - engine 构建时间太长，怎么节约多次构建时的时间?
  - 某些 Layer 的算法导致较大的误差，能不能精确选择算法?
  - 想更新模型的权重，但又不想重新构建 engine?
  - 构建期/运行期显存占用过大，怎么减少?

• 高级话题：Refit

- 优点
  - 节约反复构建引擎的时间
  - 强化学习必备

```python
config.flags = 1 << int(trt.BuilderFlag.REFIT)

...  # 已经构建好了 engine

refitter = trt.Refitter(engine, logger)
refitter.set_weights("conv", trt.WeightsRole.KERNEL, weight)
refitter.set_weights("conv", trt.WeightsRole.BIAS, bias)

[missingLayer, weightRole] = refitter.get_missing()
for layer, role in zip(missingLayer, weightRole):
    print("[", layer, "-", role, "]")

if refitter.refit_cuda_engine() == False:
    print("Failed Refitting engine!")
    return
```

- 要点
  - BuilderConfig 中设置相应 Flag
  - 在构建好 engine 的基础上更新权重
  - 更新某层权重后，邻近层可能也需要更新（尽管其值可能不变），如 Convolution 层中的 kernel 和 bias

    [TensorRT] ERROR: 4: [refit.cpp::refitCudaEngine::1769] Error Code 4: Internal Error (missing 1 needed Weights. Call IRefitter::getMissing to get their layer names and roles or IRefitter::getMissingWeights to get their weights names.)

  - 注意权重的排布方式
  - Dynamic Shape 模式暂不支持（未来 TensorRT 版本将添加支持）

    [TRT] [E] 4: [network.cpp::validate::2924] Error Code 4: Internal Error (Refittable networks with dynamic shapes is not supported.)

高级话题

- 希望解决的问题
  - Dynamic Shape 模式在 min-opt-max 跨度较大时性能下降?
  - 怎样重叠计算和数据拷贝的时间，增加GPU利用率?
  - 怎样使一个 engine 供多个线程使用?
  - 怎么样优化 Kernel 的调用，减少 Launch Bound 的发生?
  - engine 构建时间太长，怎么节约多次构建时的时间?
  - 某些 Layer 的算法导致较大的误差，能不能精确选择算法?
  - 想更新模型的权重，但又不想重新构建 engine?
  - 构建期/运行期显存占用过大，怎么减少?

高级话题：Tactic Source

- 要点
  - BuilderConfig 中设置相应 Flag
  - 可以开启或关闭 cuBLAS、cuBLASLt、cuDNN
  - 优点
    - 节约部分内存、显存，减少构建时间
  - 缺点
    - 不能使用某些优化，可能导致性能下降
    - 可能导致构建失败
  - 后续版本中，TensorRT 将彻底断开对外部 Library 依赖

```
config.set_tactic_sources(1 << int(trt.TacticSource.CUBLAS) | \
                          1 << int(trt.TacticSource.CUBLAS_LT) | \
                          1 << int(trt.TacticSource.CUDNN))
#config.set_tactic_sources(1 << int(trt.TacticSource.CUBLAS) | \
#                           1 << int(trt.TacticSource.CUBLAS_LT))
```

• 高级话题：Tactic Source

所有 Tactic

```
[V] Using cublas as a tactic source
[I] [MemUsageChange] Init cuBLAS/cuBLASLt: CPU +269, GPU +112, now: CPU 541, GPU 739 (MiB)
[V] Using cuDNN as a tactic source
[I] [MemUsageChange] Init cuDNN: CPU +112, GPU +46, now: CPU 653, GPU 785 (MiB)
[I] Local timing cache in use. Profiling results in this builder pass will not be stored.

[V] Constructing optimization profile number 0 [1/1].

...

[V] --------------- Timing Runner: (Unnamed Layer* 0) [Convolution] + (Unnamed Layer* 1) [Activation] (CudnnConvolution)
[V] Tactic: 0 Time: 0.013312
[V] Tactic: 1 Time: 0.01328
[V] Tactic: 2 Time: 0.0256
[V] Tactic: 4 Time: 0.031744
[V] Tactic: 5 Time: 0.03072
[V] Fastest Tactic: 1 Time: 0.01328
[V] --------------- Timing Runner: (Unnamed Layer* 0) [Convolution] + (Unnamed Layer* 1) [Activation] (CaskConvolution)
[V] (Unnamed Layer* 0) [Convolution] + (Unnamed Layer* 1) [Activation] Set Tactic Name: maxwell_scudnn_128x32_relu_medium_nn_v1 Tactic: 1062367460111450758
[V] Tactic: 1062367460111450758 Time: 0.014272

...

[V] (Unnamed Layer* 0) [Convolution] + (Unnamed Layer* 1) [Activation] Set Tactic Name: maxwell_scudnn_128x128_relu_small_nn_v1 Tactic: -410470605513481746
[V] Tactic: -410470605513481746 Time: 0.012288
[V] Fastest Tactic: 7144526460361122478 Time: 0.009216

...

[V] --------------- Timing Runner: (Unnamed Layer* 7) [Fully Connected] + (Unnamed Layer* 8) [Activation] (CudaDepthwiseConvolution)
[V] CudaDepthwiseConvolution has no valid tactics for this config, skipping
[V] --------------- Timing Runner: (Unnamed Layer* 7) [Fully Connected] + (Unnamed Layer* 8) [Activation] (FusedConvActConvolution)
[V] FusedConvActConvolution has no valid tactics for this config, skipping
[V] --------------- Timing Runner: (Unnamed Layer* 7) [Fully Connected] + (Unnamed Layer* 8) [Activation] (CudnnConvolution)
[V] Tactic: 0 Time: 0.32768
[V] Tactic: 1 Time: 0.072704
[V] Tactic: 2 Time: 0.400384
[V] Tactic: 4 skipped. Scratch requested: 7426048000, available: 6442450944
[V] Tactic: 5 Time: 6.10406
[I] Some tactics do not have sufficient workspace memory to run. Increasing workspace size may increase performance, please check verbose output.
```

不用cuDNN

少了cuDNN的memory占用

少了cuDNN的测试

```
[V] Using cublas as a tactic source
[I] [MemUsageChange] Init cuBLAS/cuBLASLt: CPU +269, GPU +112, now: CPU 541, GPU 739 (MiB)


[I] Local timing cache in use. Profiling results in this builder pass will not be stored.
[V] CuDNN tactics have been disabled during engine build
[V] Constructing optimization profile number 0 [1/1].

...

[V] --------------- Timing Runner: (Unnamed Layer* 0) [Convolution] + (Unnamed Layer* 1) [Activation] (CudnnConvolution)
[V] CudnnConvolution has no valid tactics for this config, skipping


[V] --------------- Timing Runner: (Unnamed Layer* 0) [Convolution] + (Unnamed Layer* 1) [Activation] (CaskConvolution)
[V] (Unnamed Layer* 0) [Convolution] + (Unnamed Layer* 1) [Activation] Set Tactic Name: maxwell_scudnn_128x32_relu_medium_nn_v1 Tactic: 1062367460111450758
[V] Tactic: 1062367460111450758 Time: 0.01424

...

[V] (Unnamed Layer* 0) [Convolution] + (Unnamed Layer* 1) [Activation] Set Tactic Name: maxwell_scudnn_128x128_relu_small_nn_v1 Tactic: -410470605513481746
[V] Tactic: -410470605513481746 Time: 0.013312
[V] Fastest Tactic: 7144526460361122478 Time: 0.009184

...

[V] --------------- Timing Runner: (Unnamed Layer* 7) [Fully Connected] + (Unnamed Layer* 8) [Activation] (CudaDepthwiseConvolution)
[V] CudaDepthwiseConvolution has no valid tactics for this config, skipping
[V] --------------- Timing Runner: (Unnamed Layer* 7) [Fully Connected] + (Unnamed Layer* 8) [Activation] (FusedConvActConvolution)
[V] FusedConvActConvolution has no valid tactics for this config, skipping
[V] --------------- Timing Runner: (Unnamed Layer* 7) [Fully Connected] + (Unnamed Layer* 8) [Activation] (CudnnConvolution)
[V] CudnnConvolution has no valid tactics for this config, skipping
```

• 高级话题：Tactic Source

所有 Tactic

```
[V] Tactic: 5 Time: 6.10406
[I] Some tactics do not have sufficient workspace memory to run. Increasing workspace size may increase performance, please check verbose output.
[V] Fastest Tactic: 1 Time: 0.072704
[V] Setting workspace to 7426048000enables more tactics for profiling
[V] --------------- Timing Runner: (Unnamed Layer* 7) [Fully Connected] + (Unnamed Layer* 8) [Activation] (CublasConvolution)

...

[V] Fastest Tactic: 6629944304117643200 Time: 0.165888
[V] >>>>>>>>>>>>>>> Chose Runner Type: CaskConvolution Tactic: 6629944304117643200

...

[V] Formats and tactics selection completed in 1.29186 seconds.
[V] After reformat layers: 9 layers

...

[I] [MemUsageStats] Peak memory usage of TRT CPU/GPU memory allocators: CPU 12 MiB, GPU 430 MiB

...

[I] [MemUsageChange] Init cuBLAS/cuBLASLt: CPU +0, GPU +8, now: CPU 953, GPU 927 (MiB)
[V] Using cuDNN as a tactic source
[I] [MemUsageChange] Init cuDNN: CPU +0, GPU +8, now: CPU 953, GPU 935 (MiB)
[V] Engine generation completed in 1.65861 seconds.

...

[I] [MemUsageChange] TensorRT-managed allocation in building engine: CPU +0, GPU +13, now: CPU 0, GPU 13 (MiB)
[I] [MemUsageChange] Init CUDA: CPU +0, GPU +0, now: CPU 965, GPU 887 (MiB)
[I] Loaded engine size: 12 MiB
[V] Using cublas as a tactic source
[I] [MemUsageChange] Init cuBLAS/cuBLASLt: CPU +1, GPU +10, now: CPU 966, GPU 911 (MiB)
[V] Using cuDNN as a tactic source
[I] [MemUsageChange] Init cuDNN: CPU +0, GPU +8, now: CPU 966, GPU 919 (MiB)
[V] Deserialization required 4159 microseconds.
[I] [MemUsageChange] TensorRT-managed allocation in engine deserialization: CPU +0, GPU +12, now: CPU 0, GPU 12 (MiB)
[V] Using cublas as a tactic source
[I] [MemUsageChange] Init cuBLAS/cuBLASLt: CPU +1, GPU +10, now: CPU 966, GPU 911 (MiB)
[V] Using cuDNN as a tactic source
[I] [MemUsageChange] Init cuDNN: CPU +0, GPU +8, now: CPU 966, GPU 919 (MiB)

...
```

不用cuDNN

```
[V] --------------- Timing Runner: (Unnamed Layer* 7) [Fully Connected] + (Unnamed Layer* 8) [Activation] (CublasConvolution)

...

[V] Fastest Tactic: 6629944304117643200 Time: 0.165888
[V] >>>>>>>>>>>>>>> Chose Runner Type: CaskConvolution Tactic: 6629944304117643200

...

[V] Formats and tactics selection completed in 1.03523 seconds.
[V] After reformat layers: 9 layers

...

[I] [MemUsageStats] Peak memory usage of TRT CPU/GPU memory allocators: CPU 12 MiB, GPU 25 MiB

...

[I] [MemUsageChange] Init cuBLAS/cuBLASLt: CPU +0, GPU +8, now: CPU 542, GPU 761 (MiB)

[V] Engine generation completed in 1.28971 seconds.

...

[I] [MemUsageChange] TensorRT-managed allocation in building engine: CPU +0, GPU +13, now: CPU 0, GPU 13 (MiB)
[I] [MemUsageChange] Init CUDA: CPU +0, GPU +0, now: CPU 554, GPU 729 (MiB)
[I] Loaded engine size: 12 MiB
[V] Using cublas as a tactic source
[I] [MemUsageChange] Init cuBLAS/cuBLASLt: CPU +1, GPU +10, now: CPU 555, GPU 753 (MiB)


[V] Deserialization required 2721 microseconds.
[I] [MemUsageChange] TensorRT-managed allocation in engine deserialization: CPU +0, GPU +12, now: CPU 0, GPU 12 (MiB)
[V] Using cublas as a tactic source
[I] [MemUsageChange] Init cuBLAS/cuBLASLt: CPU +0, GPU +10, now: CPU 555, GPU 753 (MiB)


...
```

优化耗时减少

构建期 memory 需求减少

运行期 memory 需求减少

• 高级话题：更多功能

- Profiling
- Profiling Verbosity
- Error Recorder
- GPU Allocator
- Engine Inspector
- Empty Tensor
- …

- 范例代码均在 cookbook 的 09-Advance 中，以后逐步更新完善……


NVIDIA