Gather 层

- 初始示例代码
- axis
- mode（要求 TensorRT>=8.2）
  - Gather DEFAULT 模式
  - Gather ELEMENT 模式
  - Gather ND 模式与 num_elementwise_dims 参数

初始示例代码

```
import numpy as np
from cuda import cudart
import tensorrt as trt

nIn, cIn, hIn, wIn = 1, 3, 4, 5  # 输入张量 NCHW
lenIndex = 3
data0 = np.arange(cIn).reshape(cIn, 1, 1) * 100 + np.arange(hIn).reshape(1, hIn, 1) * 10 +
np.arange(wIn).reshape(1, 1, wIn)
data0 = data0.reshape(nIn, cIn, hIn, wIn).astype(np.float32)  # 输入数据
data1 = np.array([1, 0, 2], dtype=np.int32)  # 下标数据

np.set_printoptions(precision=8, linewidth=200, suppress=True)
cudart.cudaDeviceSynchronize()

logger = trt.Logger(trt.Logger.ERROR)
builder = trt.Builder(logger)
network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
config = builder.create_builder_config()
config.max_workspace_size = 1 << 30
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))
inputT1 = network.add_input('inputT1', trt.DataType.INT32, (len(data1), ))
#------------------------------------------------------------ --------------------# 替换部分
gatherLayer = network.add_gather(inputT0, inputT1, 1)
#------------------------------------------------------------ --------------------# 替换部分
network.mark_output(gatherLayer.get_output(0))
engineString = builder.build_serialized_network(network, config)
engine = trt.Runtime(logger).deserialize_cuda_engine(engineString)
context = engine.create_execution_context()
_, stream = cudart.cudaStreamCreate()

inputH0 = np.ascontiguousarray(data0.reshape(-1))
inputH1 = np.ascontiguousarray(data1.reshape(-1))
outputH0 = np.empty(context.get_binding_shape(2), dtype=trt.nptype(engine.get_binding_dtype(2)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, inputD1 = cudart.cudaMallocAsync(inputH1.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
cudart.cudaMemcpyAsync(inputD1, inputH1.ctypes.data, inputH1.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(inputD1), int(outputD0)], stream)
```

```
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data0.shape)
print(data0)
print("inputH0 :", data1.shape)
print(data1)
print("outputH0:", outputH0.shape)
print(outputH0)

cudart.cudaStreamDestroy(stream)
cudart.cudaFree(inputD0)
cudart.cudaFree(inputD1)
cudart.cudaFree(outputD0)
```

- 输入张量 0 形状 (1,3,4,5)，百位表示 C 维编号，十位表示 H 维编号，个位表示 W 维编号

$$\left[\left[\begin{bmatrix} 0. & 1. & 2. & 3. & 4. \\ 10. & 11. & 12. & 13. & 14. \\ 20. & 21. & 22. & 23. & 24. \\ 30. & 31. & 32. & 33. & 34. \end{bmatrix} \begin{bmatrix} 100. & 101. & 102. & 103. & 104. \\ 110. & 111. & 112. & 113. & 114. \\ 120. & 121. & 122. & 123. & 124. \\ 130. & 131. & 132. & 133. & 134. \end{bmatrix} \begin{bmatrix} 200. & 201. & 202. & 203. & 204. \\ 210. & 211. & 212. & 213. & 214. \\ 220. & 221. & 222. & 223. & 224. \\ 230. & 231. & 232. & 233. & 234. \end{bmatrix}\right]\right]$$

- 输入张量 1 形状 (3,)，取元素的索引张量

$$\begin{bmatrix} 1 & 0 & 2 \end{bmatrix}$$

- 输出张量形状 (1,3,4,5)，在次高维上按照下标张量重排顺序

$$\left[\begin{bmatrix} 100. & 101. & 102. & 103. & 104. \\ 110. & 111. & 112. & 113. & 114. \\ 120. & 121. & 122. & 123. & 124. \\ 130. & 131. & 132. & 133. & 134. \end{bmatrix} \begin{bmatrix} 0. & 1. & 2. & 3. & 4. \\ 10. & 11. & 12. & 13. & 14. \\ 20. & 21. & 22. & 23. & 24. \\ 30. & 31. & 32. & 33. & 34. \end{bmatrix} \begin{bmatrix} 200. & 201. & 202. & 203. & 204. \\ 210. & 211. & 212. & 213. & 214. \\ 220. & 221. & 222. & 223. & 224. \\ 230. & 231. & 232. & 233. & 234. \end{bmatrix}\right]$$

- TensorRT>=8.2 后需要设置 config.max_wokspace_size，否则报错

```
[TRT] [W] Skipping tactic 0 due to Myelin error: myelinTargetSetPropertyMemorySize called with invalid
memory size (0).
[TRT] [E] 10: [optimizer.cpp::computeCosts::2011] Error Code 10: Internal Error (Could not find any
implementation for node {ForeignNode[(Unnamed Layer* 0) [Gather]]}.)
```

---

axis

```
gatherLayer = network.add_gather(inputT0, inputT1, 1)
gatherLayer.axis = 0  # 重设操作的维度编号，默认值 1
```

- 指定 axis=0（在最高维上按照下标张量重排顺序），输出张量形状 (3,3,4,5)

$$\begin{bmatrix} \begin{bmatrix} \begin{bmatrix} 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \end{bmatrix} \begin{bmatrix} 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \end{bmatrix} \begin{bmatrix} 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} \begin{bmatrix} 0. & 1. & 2. & 3. & 4. \\ 10. & 11. & 12. & 13. & 14. \\ 20. & 21. & 22. & 23. & 24. \\ 30. & 31. & 32. & 33. & 34. \end{bmatrix} \begin{bmatrix} 100. & 101. & 102. & 103. & 104. \\ 110. & 111. & 112. & 113. & 114. \\ 120. & 121. & 122. & 123. & 124. \\ 130. & 131. & 132. & 133. & 134. \end{bmatrix} \begin{bmatrix} 200. & 201. & 202. & 203. & 204. \\ 210. & 211. & 212. & 213. & 214. \\ 220. & 221. & 222. & 223. & 224. \\ 230. & 231. & 232. & 233. & 234. \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} \begin{bmatrix} 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \end{bmatrix} \begin{bmatrix} 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \end{bmatrix} \begin{bmatrix} 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

- 指定 axis=1（在次高维上按照下标张量重排顺序），输出张量形状 (1,3,4,5)，结果与初始示例代码相同
- 指定 axis=2（在季高维上按照下标张量重排顺序），输出张量形状 (1,3,3,5)

$$\begin{bmatrix} \begin{bmatrix} \begin{bmatrix} 10. & 11. & 12. & 13. & 14. \\ 0. & 1. & 2. & 3. & 4. \\ 20. & 21. & 22. & 23. & 24. \end{bmatrix} \begin{bmatrix} 110. & 111. & 112. & 113. & 114. \\ 100. & 101. & 102. & 103. & 104. \\ 120. & 121. & 122. & 123. & 124. \end{bmatrix} \begin{bmatrix} 210. & 211. & 212. & 213. & 214. \\ 200. & 201. & 202. & 203. & 204. \\ 220. & 221. & 222. & 223. & 224. \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

- 指定 axis=3（在叔高维上按照下标张量重排顺序），输出张量形状 (1,3,4,3)

$$\begin{bmatrix} \begin{bmatrix} \begin{bmatrix} 1. & 0. & 2. \\ 11. & 10. & 12. \\ 21. & 20. & 22. \\ 31. & 30. & 32. \end{bmatrix} \begin{bmatrix} 101. & 100. & 102. \\ 111. & 110. & 112. \\ 121. & 120. & 122. \\ 131. & 130. & 132. \end{bmatrix} \begin{bmatrix} 201. & 200. & 202. \\ 211. & 210. & 212. \\ 221. & 220. & 222. \\ 231. & 230. & 232. \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

---

mode（要求 TensorRT>=8.2）

DEFAULT 模式

```python
import numpy as np
from cuda import cudart
import tensorrt as trt

nIn, cIn, hIn, wIn = 1, 3, 4, 5
lenIndex = 3
data0 = np.arange(cIn).reshape(cIn, 1, 1) * 100 + np.arange(hIn).reshape(1, hIn, 1) * 10 +
np.arange(wIn).reshape(1, 1, wIn)
data0 = data0.reshape(nIn, cIn, hIn, wIn).astype(np.float32)
data1 = np.array([[0, 1, 2], [0, 2, -1]], dtype=np.int32)

np.set_printoptions(precision=8, linewidth=200, suppress=True)
cudart.cudaDeviceSynchronize()

logger = trt.Logger(trt.Logger.ERROR)
builder = trt.Builder(logger)
network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
config = builder.create_builder_config()
config.max_workspace_size = 1 << 30
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))
inputT1 = network.add_input('inputT1', trt.DataType.INT32, data1.shape)
gatherLayer = network.add_gather(inputT0, inputT1, 1)
gatherLayer.mode = trt.GatherMode.ND
#gatherLayer.num_elementwise_dims = 0
network.mark_output(gatherLayer.get_output(0))
engineString = builder.build_serialized_network(network, config)
```

```
engine = trt.Runtime(logger).deserialize_cuda_engine(engineString)
context = engine.create_execution_context()
_, stream = cudart.cudaStreamCreate()

inputH0 = np.ascontiguousarray(data0.reshape(-1))
inputH1 = np.ascontiguousarray(data1.reshape(-1))
outputH0 = np.empty(context.get_binding_shape(2), dtype=trt.nptype(engine.get_binding_dtype(2)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, inputD1 = cudart.cudaMallocAsync(inputH1.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
cudart.cudaMemcpyAsync(inputD1, inputH1.ctypes.data, inputH1.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(inputD1), int(outputD0)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data0.shape)
print(data0)
print("inputH0 :", data1.shape)
print(data1)
print("outputH0:", outputH0.shape)
print(outputH0)

cudart.cudaStreamDestroy(stream)
cudart.cudaFree(inputD0)
cudart.cudaFree(outputD0)
```

- 输入张量 0 与初始示例代码相同
- 输入张量 1 形状 (3,2)

$$\begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 2 & 1 \end{bmatrix}$$

- 指定 mode=trt.GatherMode.DEFAUT，操作维度 axis=2，输出张量形状(1,3,3,2,5)，按索引张量抽取指定维上所有元素

$$\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}10. & 11. & 12. & 13. & 14. \\ 0. & 1. & 2. & 3. & 4.\end{bmatrix} & \begin{bmatrix}0. & 1. & 2. & 3. & 4. \\ 20. & 21. & 22. & 23. & 24.\end{bmatrix} & \begin{bmatrix}20. & 21. & 22. & 23. & 24. \\ 10. & 11. & 12. & 13. & 14.\end{bmatrix}\end{bmatrix} & & \end{bmatrix} \\ \begin{bmatrix}\begin{bmatrix}110. & 111. & 112. & 113. & 114. \\ 100. & 101. & 102. & 103. & 104.\end{bmatrix} & \begin{bmatrix}100. & 101. & 102. & 103. & 104. \\ 120. & 121. & 122. & 123. & 124.\end{bmatrix} & \begin{bmatrix}120. & 121. & 122. & 123. & 124. \\ 110. & 111. & 112. & 113. & 114.\end{bmatrix}\end{bmatrix} \\ \begin{bmatrix}\begin{bmatrix}210. & 211. & 212. & 213. & 214. \\ 200. & 201. & 202. & 203. & 204.\end{bmatrix} & \begin{bmatrix}200. & 201. & 202. & 203. & 204. \\ 220. & 221. & 222. & 223. & 224.\end{bmatrix} & \begin{bmatrix}220. & 221. & 222. & 223. & 224. \\ 210. & 211. & 212. & 213. & 214.\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

- 含义：参考 Onnx Gather 算子
  - 数据张量形状 $data[d_0, d_1, \ldots, d_{r-1}]$（$dim = r$），索引张量形状 $index[a_0, a_1, \ldots, a_{q-1}]$（$dim = q$），指定 $axis = p$（$0 \le p < r$），则
  - 输出张量形状 $output[d_0, d_1, \ldots, d_{p-1}, a_0, a_1, \ldots, a_{q-1}, d_{p+1}, d_{p+2}, \ldots, d_{r-1}]$（$dim = r + q - 1$，$p = 0$ 时以 a_0 开头）
  - 注意输出张量形状中没有了 d_p 这一维，相当于把 data 的这一维扩展成 index 的维度。对于 index 的每一个元素 i，都要抽取 d_p 维上的 i 个元素作为输出
  - 命循环变量 i_j 满足 $0 \le i_j < a_j$，则计算过程可以写作（numpy 语法，等号左边的 i_* 和等号右边的 index[...] 均位于 d_p 这一维）：$output[:, :, \ldots, :, i_0, i_1, \ldots, i_{q-1}, :, :, \ldots, :] = data[:, :, \ldots, :, index[i_0, i_1, \ldots, i_{q-1}], :, :, \ldots, :]$
  - 对于上面的示例代码，就是：output[:,:,i_0,i_1,:] = inputT0[:,:,index[i_0,i_1],:]，其中 $0 \le i_0 < 3, 0 \le i_1 < 2$

ELEMENT 模式

```
import numpy as np
from cuda import cudart
import tensorrt as trt

nIn, cIn, hIn, wIn = 1, 3, 4, 5
lenIndex = 3
data0 = np.arange(cIn).reshape(cIn, 1, 1) * 100 + np.arange(hIn).reshape(1, hIn, 1) * 10 +
np.arange(wIn).reshape(1, 1, wIn)
data0 = data0.reshape(nIn, cIn, hIn, wIn).astype(np.float32)
data1 = np.zeros(data0.shape, dtype=np.int32)

np.random.seed(97)
axis = 2
# 使用随机排列
for i in range(data0.shape[0]):
    for j in range(data0.shape[1]):
        for k in range(data0.shape[3]):
            data1[i, j, :, k] = np.random.permutation(range(data0.shape[2]))
'''# 使用随机数也可以
for i in range(data0.shape[0]):
    for j in range(data0.shape[1]):
        for k in range(data0.shape[3]):
            data1[i,j,:,k] = [ np.random.randint(0,data0.shape[2]) for i in range(data0.shape[2]) ]
'''

np.set_printoptions(precision=8, linewidth=200, suppress=True)
cudart.cudaDeviceSynchronize()

logger = trt.Logger(trt.Logger.ERROR)
builder = trt.Builder(logger)
network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
config = builder.create_builder_config()
config.max_workspace_size = 1 << 30
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))
inputT1 = network.add_input('inputT1', trt.DataType.INT32, (nIn, cIn, hIn, wIn))
gatherLayer = network.add_gather(inputT0, inputT1, 1)
gatherLayer.mode = trt.GatherMode.ELEMENT
gatherLayer.axis = 2
network.mark_output(gatherLayer.get_output(0))
engineString = builder.build_serialized_network(network, config)
engine = trt.Runtime(logger).deserialize_cuda_engine(engineString)
context = engine.create_execution_context()
_, stream = cudart.cudaStreamCreate()

inputH0 = np.ascontiguousarray(data0.reshape(-1))
inputH1 = np.ascontiguousarray(data1.reshape(-1))
outputH0 = np.empty(context.get_binding_shape(2), dtype=trt.nptype(engine.get_binding_dtype(2)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, inputD1 = cudart.cudaMallocAsync(inputH1.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
cudart.cudaMemcpyAsync(inputD1, inputH1.ctypes.data, inputH1.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(inputD1), int(outputD0)], stream)
```

```
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data0.shape)
print(data0)
print("inputH0 :", data1.shape)
print(data1)
print("outputH0:", outputH0.shape)
print(outputH0)

cudart.cudaStreamDestroy(stream)
cudart.cudaFree(inputD0)
cudart.cudaFree(outputD0)
```

- 输入张量 0 与初始示例代码相同
- 输入张量 1 形状 (1,3,4,5)

$$\left[\left[\begin{bmatrix}0&2&3&0&2\\3&3&2&3&3\\1&0&0&1&1\\2&1&1&2&0\end{bmatrix}\begin{bmatrix}1&0&3&3&2\\2&1&2&1&1\\3&2&0&2&0\\0&3&1&0&3\end{bmatrix}\begin{bmatrix}0&2&2&2&1\\3&1&0&3&3\\2&3&3&0&2\\1&0&1&1&0\end{bmatrix}\right]\right]$$

- 指定 mode=trt.GatherMode.ELEMENT，操作维度 axis=2，输出张量形状 (1,3,4,5)，按索引张量抽取指定位置上单个元素，表现为十位数按照输入张量 1 的次序作排列

$$\left[\left[\begin{bmatrix}0.&21.&32.&3.&24.\\30.&31.&22.&33.&34.\\10.&1.&2.&13.&14.\\20.&11.&12.&23.&4.\end{bmatrix}\begin{bmatrix}110.&101.&132.&133.&124.\\120.&111.&122.&113.&114.\\130.&121.&102.&123.&104.\\100.&131.&112.&103.&134.\end{bmatrix}\begin{bmatrix}200.&221.&222.&223.&214.\\230.&211.&202.&233.&234.\\220.&231.&232.&203.&224.\\210.&201.&212.&213.&204.\end{bmatrix}\right]\right]$$

- 含义：参考 Onnx GatherElements 算子
  - 数据张量、索引张量、输出张量形状相同（$dim = r$），$data[d_0, d_1, \dots, d_{r-1}], index[d_0, d_1, \dots, d_{r-1}]$，指定 $axis = p$（$0 \le p < r$），则
  - 输出张量形状 $output[d_0, d_1, \dots, d_{r-1}]$
  - 命循环变量 i_j 满足 $0 \le i_j < d_j$，则计算过程可以写作（numpy 语法，等号左边的 i 和等号右边的 index[...] 均位于 d_p 这一维）：
    $output[i_0, i_1, \dots, i_{p-1}, i_p, i_{p+1}, \dots, i_{r-1}] = data[i_0, i_1, \dots, i_{p-1}, index[i_0, i_1, \dots, i_{p-1}, i_p, i_{p+1}, \dots, i_{r-1}], i_{p+1}, \dots, i_{r-1}]$
  - 对于上面的示例代码，就是：output[:,:,i,:] = inputT0[:,:,index[:,:,i,:],:]，其中 $0 \le i < 4$

ND 模式与 num_elementwise_dims 参数

```
import numpy as np
from cuda import cudart
import tensorrt as trt

nIn, cIn, hIn, wIn = 1, 3, 4, 5
lenIndex = 3
data0 = np.arange(cIn).reshape(cIn, 1, 1) * 100 + np.arange(hIn).reshape(1, hIn, 1) * 10 +
np.arange(wIn).reshape(1, 1, wIn)
data0 = data0.reshape(nIn, cIn, hIn, wIn).astype(np.float32)
data1 = np.array([[0, 1, 2], [0, 2, -1]], dtype=np.int32)

np.set_printoptions(precision=8, linewidth=200, suppress=True)
cudart.cudaDeviceSynchronize()

logger = trt.Logger(trt.Logger.ERROR)
builder = trt.Builder(logger)
```

```
network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
config = builder.create_builder_config()
config.max_workspace_size = 1 << 30
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))
inputT1 = network.add_input('inputT1', trt.DataType.INT32, data1.shape)
gatherLayer = network.add_gather(inputT0, inputT1, 1)
gatherLayer.mode = trt.GatherMode.ND
#gatherLayer.num_elementwise_dims = 0
network.mark_output(gatherLayer.get_output(0))
engineString = builder.build_serialized_network(network, config)
engine = trt.Runtime(logger).deserialize_cuda_engine(engineString)
context = engine.create_execution_context()
_, stream = cudart.cudaStreamCreate()

inputH0 = np.ascontiguousarray(data0.reshape(-1))
inputH1 = np.ascontiguousarray(data1.reshape(-1))
outputH0 = np.empty(context.get_binding_shape(2), dtype=trt.nptype(engine.get_binding_dtype(2)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, inputD1 = cudart.cudaMallocAsync(inputH1.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
cudart.cudaMemcpyAsync(inputD1, inputH1.ctypes.data, inputH1.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(inputD1), int(outputD0)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data0.shape)
print(data0)
print("inputH0 :", data1.shape)
print(data1)
print("outputH0:", outputH0.shape)
print(outputH0)

cudart.cudaStreamDestroy(stream)
cudart.cudaFree(inputD0)
cudart.cudaFree(outputD0)
```

- 指定 mode=trt.GatherMode.ND，不指定 num_elementwise_dims（取默认值 0），输入张量 0 与初始示例代码相同，输入张量 1 形状 (2,3)，输出张量形状 (2,5)。索引张量从最高维开始在数据张量中查找，抽取指定位置上剩余维度的所有元素

$$\text{输入张量}1 = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 2 & -1 \end{bmatrix}$$

$$\text{输出张量} = \begin{bmatrix} 120. & 121. & 122. & 123. & 124. \\ 230. & 231. & 232. & 233. & 234. \end{bmatrix}$$

- 指定 mode=trt.GatherMode.ND，指定 num_elementwise_dims=1，输入张量 0 与初始示例代码相同，输入张量 1 形状 (1,2,3)，输出张量形状 (1,2)。两个输入张量的最高 1 维必须相同，索引张量从次高维开始在数据张量中查找

$$\text{输入张量}1 = \begin{bmatrix} \begin{bmatrix} 0 & 1 & 2 \\ 0 & 2 & -1 \end{bmatrix} \end{bmatrix}$$

$$\text{输出张量} = \begin{bmatrix} 12. & 24. \end{bmatrix}$$

- 指定 mode=trt.GatherMode.ND，指定 num_elementwise_dims=2，输入张量 0 与初始示例代码相同，输入张量 1 形状 (1,3,2)，输出张量形状 (1,3)。两个输入张量的最高 2 维必须相同，索引张量从季高维开始在数据张量中查找

$$
\begin{aligned}
\text{输入张量}1 &= \begin{bmatrix}\begin{bmatrix}2 & 1\\ 3 & 0\\ 1 & 2\end{bmatrix}\end{bmatrix}\\
\text{输出张量} &= [21. \quad 130. \quad 212.]
\end{aligned}
$$

- 指定 mode=trt.GatherMode.ND，指定 num_elementwise_dims=3，输入张量 0 与初始示例代码相同，输入张量 1 形状 (1,3,4,1)，输出张量形状 (1,3,4)。两个输入张量的最高 3 维必须相同，索引张量从叔高维开始在数据张量中查找

$$
\begin{aligned}
\text{输入张量}1 &= \begin{bmatrix}\begin{bmatrix}\begin{bmatrix}0\\0\\0\\0\end{bmatrix} & \begin{bmatrix}2\\2\\2\\2\end{bmatrix} & \begin{bmatrix}1\\1\\1\\1\end{bmatrix}\end{bmatrix}\end{bmatrix}\\
\text{输出张量} &= \begin{bmatrix}\begin{bmatrix}0. & 10. & 20. & 30.\\ 102. & 112. & 122. & 132.\\ 201. & 211. & 221. & 231.\end{bmatrix}\end{bmatrix}
\end{aligned}
$$

- 指定 mode=trt.GatherMode.ND，指定 num_elementwise_dims=2，输入张量 0 与初始示例代码相同，输入张量 1 形状 (1,3,1)，输出张量形状 (1,3,5)

$$
\begin{aligned}
\text{输入张量}1 &= [[[2][3][1]]]\\
\text{输出张量} &= \begin{bmatrix}\begin{bmatrix}20. & 21. & 22. & 23. & 24.\\ 130. & 131. & 132. & 133. & 134.\\ 210. & 211. & 212. & 213. & 214.\end{bmatrix}\end{bmatrix}
\end{aligned}
$$

- 含义：参考 Onnx GatherND 算子
  - TensorRT 说明 link
  - 数据张量形状 $data[d_0, d_1, \dots, d_{r-1}]$（$dim = r$），索引张量形状 $index[a_0, a_1, \dots, a_{q-1}]$（$dim = q$），指定 $nElementwiseDim = p$（要求 a_{q-1} 为构建期常量），则
  - 输出张量维度 $dim(output) = q + r - index.shape[-1] - 1 - nElementwiseDim$（以下 $nElementwiseDim$ 简记为 nB，它的含义是"被当成 Batch 维的维度数"）
  - 要求 $nB < min(r, q)$，否则报错。即要求跳过的维数不能超过 $data$ 和 $index$ 维数中的较小者
  - 要求 $data.shape[:nB] = index.shape[:nB]$，否则报错。即要求 $data$ 和 $index$ 的形状的前 nB 维尺寸都相等（都当做 $Batch$ 维）
  - 要求 $index.shape[-1] \le r - nB$，否则报错。即要求"$index$ 跳过 nB 维后的剩余维度数"（真实索引维数）不能超出"$data$ 跳过 nB 维后的剩余维度数"
  - 对于 $index$ 中第 j 维的索引 i_j（$0 \le j < q$） 要求 $-d_j \le index[:, :, \dots, i_j, :, :, \dots, :] \le d_j - 1$，即可以使用负的索引号
  - 命 $N = a_0 * a_1 * \dots * a_{nB-1}$，即 $data$ 和 $index$ 的所有 $Batch$ 维元素数
  - （onnx 文档的解释）If indices_shape[-1] == r-b, since the rank of indices is q, indices can be thought of as N (q-b-1)-dimensional tensors containing 1-D tensors of dimension r-b, where N is an integer equals to the product of 1 and all the elements in the batch dimensions of the indices_shape. Let us think of each such r-b ranked tensor as indices_slice. Each scalar value corresponding to data[0:b-1,indices_slice] is filled into the corresponding location of the (q-b-1)-dimensional tensor to form the output tensor
  - （onnx 文档的解释）If indices_shape[-1] < r-b, since the rank of indices is q, indices can be thought of as N (q-b-1)-dimensional tensor containing 1-D tensors of dimension < r-b. Let us think of each such tensors as indices_slice. Each tensor slice corresponding to data[0:b-1, indices_slice , :] is filled into the corresponding location of the (q-b-1)-dimensional tensor to form the output tensor
- 计算公式：

$$
\begin{aligned}
&output[i_0, i_1, \dots, i_{nB-1}, i_{nB}, i_{nB+1}, \dots, i_{q-2}]\\
&= data[i_0, i_1, \dots, i_{nB-1}, index[i_0, i_1, \dots, i_{nB-1}, i_{nB}, i_{nB+1}, \dots, i_{q-2}]]
\end{aligned}
$$

- 式子中下标当 $0 \le j < nB$ 时 $0 \le i_j < d_j$，当 $nB \le j < q - 1$ 时 $0 \le i_j < a_j$
- 式子中 $output$ 索引的前 nB 项来自公共 $Batch$ 部分（一共 nB 个），以后索引来自 $index$ 跳过 $Batch$ 维后的部分（一共 $q - 2 - (nB - 1)$ 个），两部分总共 $q - 1$ 项
- 式子中 $index$ 只索引了前 $q - 1$ 维，$index[\dots]$ 实际上是个 a_{q-1} 维的张量

- 若 $a_{q-1} = r - nB$（即真实索引维数等于 data 剩余维度数）：
  - 此时由公式计算的输出张量维度 $\mathbf{dim}(output) = q + r - a_{q-1} - 1 - nB = q + r - (r - nB) - 1 - nB = q - 1$
  - 由于此时 $a_{q-1} = r - nB$，所以 $data[\dots]$ 中总索引深度为 $nB + a_{q-1} = nB + (r - nB) = r$，恰好定位到 $data$ 的一个元素上
  - 对于上面的范例代码（考虑 num_elementwise_dims=1 那一组）：
    - $r = 4, q = 3, nB = 1$，$a_{q-1} = 3 = r - nB$，首维 1 被当成 $Batch$ 维，$\mathbf{dim}(output) = q - 1 = 2$
    - output[:,i] = data[:,index[:,i]] = [12,24]，其中
    - index[0,0] = [0,1,2]，所以 output[0,0] = data[0,0,1,2] = 12
    - index[0,1] = [0,2,-1]，所以 output[0,1] = data[0,0,2,-1] = 24
- 若 $a_{q-1} < r - nB$（即真实索引维数小于 data 剩余维度数），记 $nD = r - nB - a_{q-1}$
  - 此时由公式计算的输出张量维度 $\mathbf{dim}(output) = q + r - a_{q-1} - 1 - nB = q + r - (r - nB - nD) - 1 - nB = q - 1 + nD$
  - 由于此时 $a_{q-1} = r - nB - nD$，所以 $data[\dots]$ 中总索引深度为 $nB + a_{q-1} = nB + (r - nB - nD) = r - nD$，每个索引将会定位到 $data$ 末尾 nD 维子张量上
  - 对于上面的范例代码（考虑 num_elementwise_dims=2 的后一组）：
    - $r = 4, q = 3, nB = 2$，$a_{q-1} = 1 < r - nB$，$nD = r - nB - a_{q-1} = 1$，首两维被当成 $Batch$ 维，$\mathbf{dim}(output) = q - 1 + nD = 3$
    - output[:,i] = data[:,i,index[:,i]]，其中
    - index[0,0] = [2]，所以 output[0,0] = data[0,0,2] = [20,21,22,2324]
    - index[0,1] = [3]，所以 output[0,1] = data[0,1,3] = [130,131,132,133,134]
    - index[0,2] = [1]，所以 output[0,2] = data[0,2,1] = [210,211,212,213,214]

- 不满足 index.shape[-1] ≤ r - n*ElementwiseDim 时报错（报错信息的“ -” 写成了 “+”）：

```
[TRT] [E] 1: [gatherNode.cpp::computeGatherNDOutputExtents::110] Error Code 1: Internal Error (invalid
dimension in GatherND indices[-1] > rank(data) + nbElementWiseDims)
```

- 不满足 nB < min(q,r) 时报错：

```
[TRT] [E] 3: (Unnamed Layer* 0) [Gather]: nbElementWiseDims must between 0 and rank(data)-1 inclusive
for GatherMode::kND
```

- 不满足 data.shape[:nB] == index.shape[:nB] 时报错：

```
[TRT] [E] 4: [graphShapeAnalyzer.cpp::processCheck::581] Error Code 4: Internal Error ((Unnamed Layer*
0) [Gather]: dimensions not compatible for Gather with GatherMode = kND)
```