NVIDIA TensorRT

API Migration Guide | NVIDIA Docs

DA-11734-001 _v10.4.0 | September 2024

Table of Contents

Chapter 1. Python

1.1. Python API Changes

Table 1. Allocating Buffers and Using a Name-Based Engine API

TensorRT 8.x

```
def allocate_buffers(self, engine):
    '''
    Allocates all buffers required for
 an engine, i.e. host/device inputs/
outputs.
    '''
    inputs = []
    outputs = []
    bindings = []
    stream = cuda.Stream()


    # binding is the name of input/
output
    for binding in engine:
        size =
 trt.volume(engine.get_binding_shape(bindi
 * engine.max_batch_size
        dtype =
 trt.nptype(engine.get_binding_dtype(bindi


        # Allocate host and device
 buffers
        host_mem =
 cuda.pagelocked_empty(size, dtype)
 # page-locked memory buffer (won't
 swapped to disk)
        device_mem =
 cuda.mem_alloc(host_mem.nbytes)

        # Append the device buffer
 address to device bindings.
        # When cast to int, it's a
 linear index into the context's memory
 (like memory address).
        bindings.append(int(device_mem))
```

TensorRT 10.0

```
def allocate_buffers(self, engine):
    '''
    Allocates all buffers required for
 an engine, i.e. host/device inputs/
outputs.
    '''
    inputs = []
    outputs = []
    bindings = []
    stream = cuda.Stream()

    for i in
 range(engine.num_io_tensors):
        tensor_name =
 engine.get_tensor_name(i)
        size =
 trt.volume(engine.get_tensor_shape(tensor_name))
        dtype =
 trt.nptype(engine.get_tensor_dtype(tensor_name))

        # Allocate host and device
 buffers
        host_mem =
 cuda.pagelocked_empty(size, dtype)
 # page-locked memory buffer (won't
 swapped to disk)
        device_mem =
 cuda.mem_alloc(host_mem.nbytes)

        # Append the device buffer
 address to device bindings.
        # When cast to int, it's a
 linear index into the context's memory
 (like memory address).
        bindings.append(int(device_mem))

      # Append to the appropriate
 input/output list.
```

TensorRT 8.x	TensorRT 10.0

`        # Append to the appropriate input/output list.`
`        if engine.binding_is_input(binding):`
 	
`inputs.append(self.HostDeviceMem(host_mem, device_mem))`
`        else:`
 	
`outputs.append(self.HostDeviceMem(host_mem, device_mem))`
 	
`    return inputs, outputs, bindings, stream`
	`        if engine.get_tensor_mode(tensor_name) == trt.TensorIOMode.INPUT:`
	
	`inputs.append(self.HostDeviceMem(host_mem, device_mem))`
	`        else:`
	
	`outputs.append(self.HostDeviceMem(host_mem, device_mem))`
	
	`    return inputs, outputs, bindings, stream`

Table 2. Transition from `enqueueV2` to `enqueueV3` for Python

TensorRT 8.x	TensorRT 10.0

`# Allocate device memory for inputs.`
`d_inputs = [cuda.mem_alloc(input_nbytes) for binding in range(input_num)]`
 	
`# Allocate device memory for outputs.`
`h_output = cuda.pagelocked_empty(output_nbytes, dtype=np.float32)`
`d_output = cuda.mem_alloc(h_output.nbytes)`
 	
`# Transfer data from host to device.`
`cuda.memcpy_htod_async(d_inputs[0], input_a, stream)`
`cuda.memcpy_htod_async(d_inputs[1], input_b, stream)`
`cuda.memcpy_htod_async(d_inputs[2], input_c, stream)`
 	
`# Run inference`
`context.execute_async_v2(bindings=[int(d_inp for d_inp in d_inputs] + [int(d_output)], stream_handle=stream.handle)`
 	
`# Synchronize the stream`
`stream.synchronize()`
	`# Allocate device memory for inputs.`
	`d_inputs = [cuda.mem_alloc(input_nbytes) for binding in range(input_num)]`
	
	`# Allocate device memory for outputs.`
	`h_output = cuda.pagelocked_empty(output_nbytes, dtype=np.float32)`
	`d_output = cuda.mem_alloc(h_output.nbytes)`
	
	`# Transfer data from host to device.`
	`cuda.memcpy_htod_async(d_inputs[0], input_a, stream)`
	`cuda.memcpy_htod_async(d_inputs[1], input_b, stream)`
	`cuda.memcpy_htod_async(d_inputs[2], input_c, stream)`
	
	`# Setup tensor address`
	`bindings = [int(d_inputs[i]) for i in range(3)] + [int(d_output)]`
	
	`for i in range(engine.num_io_tensors):`
	
	`context.set_tensor_address(engine.get_tensor_name(i), bindings[i])`
	
	`# Run inference`
	`context.execute_async_v3(stream_handle=stream.handle)`
	
	`# Synchronize the stream`
	`stream.synchronize()`

Table 3. Engine Building, use only `build_serialized_network`

TensorRT 8.x	TensorRT 10.0
`engine_bytes = None` `try:`	`engine_bytes = self.builder.build_serialized_network(self.network, self.config)`

TensorRT 8.x	TensorRT 10.0

`    engine_bytes = self.builder.build_serialized_network(sel self.config)`
`except AttributeError:`
`    engine = self.builder.build_engine(self.network, self.config)`
`    engine_bytes = engine.serialize()`
`    del engine`
`assert engine_bytes`
	`if engine_bytes is None:`
	`    log.error("Failed to create engine")`
	`    sys.exit(1)`

1.2. Added Python APIs

Types

- `APILanguage`
- `ExecutionContextAllocationStrategy`
- `IGpuAsyncAllocator`
- `InterfaceInfo`
- `IPluginResource`
- `IPluginV3`
- `IStreamReader`
- `IVersionedInterface`

Methods and Properties

- `ICudaEngine.is_debug_tensor()`
- `ICudaEngine.minimum_weight_streaming_budget`
- `ICudaEngine.streamable_weights_size`
- `ICudaEngine.weight_streaming_budget`
- `IExecutionContext.get_debug_listener()`
- `IExecutionContext.get_debug_state()`
- `IExecutionContext.set_all_tensors_debug_state()`
- `IExecutionContext.set_debug_listener()`
- `IExecutionContext.set_tensor_debug_state()`
- `IExecutionContext.update_device_memory_size_for_shapes()`
- `IGpuAllocator.allocate_async()`
- `IGpuAllocator.deallocate_async()`
- `INetworkDefinition.add_plugin_v3()`
- `INetworkDefinition.is_debug_tensor()`

- `INetworkDefinition.mark_debug()`
- `INetworkDefinition.unmark_debug()`
- `IPluginRegistry.acquire_plugin_resource()`
- `IPluginRegistry.all_creators`
- `IPluginRegistry.deregister_creator()`
- `IPluginRegistry.get_creator()`
- `IPluginRegistry.register_creator()`
- `IPluginRegistry.release_plugin_resource()`

1.3. Removed Python APIs

Table 4. Removed Python APIs and their Suggested Superseded API

Python API	Superseded API
`BuilderFlag.ENABLE_TACTIC_HEURISTIC`	Builder optimization level 2
`BuilderFlag.STRICT_TYPES`	Use all three flags: 1. `BuilderFlag.DIRECT_IO` 2. `BuilderFlag.PREFER_PRECISION_CONSTRAINTS` 3. `BuilderFlag.REJECT_EMPTY_ALGORITHMS`
1. `EngineCapability.DEFAULT` 2. `EngineCapability.kSAFE_DLA` 3. `EngineCapability.SAFE_GPU`	1. `EngineCapability.STANDARD` 2. `EngineCapability.DLA_STANDALONE` 3. `EngineCapability.SAFETY`
`IAlgorithmIOInfo.tensor_format`	The strides, data type, and vectorization information is sufficient to uniquely identify tensor formats.
`IBuilder.max_batch_size`	Implicit batch is no longer supported.
`IBuilderConfig.max_workspace_size`	1. `IBuilderConfig.set_memory_pool_limit()` with `MemoryPoolType.WORKSPACE` 2. `IBuilderConfig.get_memory_pool_limit()` with `MemoryPoolType.WORKSPACE`
`IBuilderConfig.min_timing_iterations`	`IBuilderConfig.avg_timing_iterations`
1. `ICudaEngine.binding_is_input()` 2. `ICudaEngine.get_binding_bytes_per_component()` 3. `ICudaEngine.get_binding_components_per_element()`	1. `ICudaEngine.get_tensor_mode()` 2. `ICudaEngine.get_tensor_bytes_per_component()` 3. `ICudaEngine.get_tensor_components_per_element()`

Python API	Superseded API
4. `ICudaEngine.get_binding_dtype()`	4. `ICudaEngine.get_tensor_dtype()`
5. `ICudaEngine.get_binding_format()`	5. `ICudaEngine.get_tensor_format()`
6. `ICudaEngine.get_binding_format_desc()`	6. `ICudaEngine.get_tensor_format_desc()`
7. `ICudaEngine.get_binding_index()`	7. No name-based equivalent replacement
8. `ICudaEngine.get_binding_name()`	8. No name-based equivalent replacement
9. `ICudaEngine.get_binding_shape()`	9. `ICudaEngine.get_tensor_shape()`
10. `ICudaEngine.get_binding_vectorized_dim()`	10. `ICudaEngine.get_tensor_vectorized_dim()`
11. `ICudaEngine.get_location()`	11. `ITensor.location`
12. `ICudaEngine.get_profile_shape()`	12. `ICudaEngine.get_tensor_profile_shape()`
13. `ICudaEngine.get_profile_shape_input()`	13. `ICudaEngine.get_tensor_profile_values()`
14. `ICudaEngine.has_implicit_batch_dimension()`	14. Implicit batch is no longer supported
15. `ICudaEngine.is_execution_binding()`	15. No name-based equivalent replacement
16. `ICudaEngine.is_shape_binding()`	16. `ICudaEngine.is_shape_inference_io()`
17. `ICudaEngine.max_batch_size()`	17. Implicit batch is no longer supported
18. `ICudaEngine.num_bindings()`	18. `ICudaEngine.num_io_tensors()`
1. `IExecutionContext.get_binding_shape()`	1. `IExecutionContext.get_tensor_shape()`
2. `IExecutionContext.get_strides()`	2. `IExecutionContext.get_tensor_strides()`
3. `IExecutionContext.set_binding_shape()`	3. `IExecutionContext.set_input_shape()`
`IFullyConnectedLayer`	`IMatrixMultiplyLayer`
1. `INetworkDefinition.add_convolution()`	1. `INetworkDefinition.add_convolution_nd()`
2. `INetworkDefinition.add_deconvolution()`	2. `INetworkDefinition.add_deconvolution_nd()`
3. `INetworkDefinition.add_fully_connected()`	3. `INetworkDefinition.add_matrix_multiply()`
4. `INetworkDefinition.add_padding()`	4. `INetworkDefinition.add_padding_nd()`
5. `INetworkDefinition.add_pooling()`	5. `INetworkDefinition.add_pooling_nd()`
6. `INetworkDefinition.add_rnn_v2()`	6. `INetworkDefinition.add_loop()`
7. `INetworkDefinition.has_explicit_precision`	7. Explicit precision support is removed in 10.0
8. `INetworkDefinition.has_implicit_batch_dimension`	8. Implicit batch is no longer supported
`IRNNv2Layer`	`ILoop`
1. `NetworkDefinitionCreationFlag.EXPLICIT_BATCH`	Support is removed in 10.0
2. `NetworkDefinitionCreationFlag.EXPLICIT_PRECISION`	
1. `PaddingMode.CAFFE_ROUND_DOWN`	Caffe is not supported since 9.0

Python API	Superseded API
2. `PaddingMode.CAFFE_ROUND_UP`	
1. `PreviewFeature.DISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805` 2. `PreviewFeature.FASTER_DYNAMIC_SHAPES_0805`	1. External tactics are always disabled for core code 2. This flag is on by default
1. `ProfilingVerbosity.DEFAULT` 2. `ProfilingVerbosity.VERBOSE`	1. `ProfilingVerbosity.LAYER_NAMES_ONLY` 2. `ProfilingVerbosity.DETAILED`
`ResizeMode`	Use `InterpolationMode`, alias is removed
`SampleMode.DEFAULT`	`SampleMode.STRICT_BOUNDS`
`SliceMode`	Use `SampleMode`, alias is removed

Chapter 2. C++

2.1. C++ API Changes

Table 5. Transition from `enqueueV2` to `enqueueV3` for C++

TensorRT 8.x	TensorRT 10.0

`// Create RAII buffer manager object.`
`samplesCommon::BufferManager`
` buffers(mEngine);`
 	
`auto context =`
` SampleUniquePtr<nvinfer1::IExecutionConte`
`>createExecutionContext());`
`if (!context)`
`{`
`    return false;`
`}`
 	
`// Pick a random digit to try to infer.`
`srand(time(NULL));`
`int32_t const digit = rand() % 10;`
 	
`// Read the input data into the managed`
` buffers.`
`// There should be just 1 input tensor.`
`ASSERT(mParams.inputTensorNames.size()`
` == 1);`
 	
`if (!processInput(buffers,`
` mParams.inputTensorNames[0], digit))`
`{`
`    return false;`
`}`
`// Create CUDA stream for the execution`
` of this inference.`
`cudaStream_t stream;`
`CHECK(cudaStreamCreate(&stream));`
	`// Create RAII buffer manager object.`
	`samplesCommon::BufferManager`
	` buffers(mEngine);`
	
	`auto context =`
	` SampleUniquePtr<nvinfer1::IExecutionContext>(mEngine-`
	`>createExecutionContext());`
	`if (!context)`
	`{`
	`    return false;`
	`}`
	
	`for (int32_t i = 0, e = mEngine-`
	`>getNbIOTensors(); i < e; i++)`
	`{`
	`    auto const name = mEngine-`
	`>getIOTensorName(i);`
	`    context->setTensorAddress(name,`
	` buffers.getDeviceBuffer(name));`
	`}`
	
	`// Pick a random digit to try to infer.`
	`srand(time(NULL));`
	`int32_t const digit = rand() % 10;`
	
	`// Read the input data into the managed`
	` buffers.`
	`// There should be just 1 input tensor.`
	`ASSERT(mParams.inputTensorNames.size()`
	` == 1);`
	
	`if (!processInput(buffers,`
	` mParams.inputTensorNames[0], digit))`
	`{`
	`    return false;`
	`}`
	`// Create CUDA stream for the execution`
	` of this inference.`

TensorRT 8.x	TensorRT 10.0

`// Asynchronously copy data from host input buffers to device input buffers`
`buffers.copyInputToDeviceAsync(stream);`
 	
`// Asynchronously enqueue the inference work`
 	
`if (!context->enqueueV2(buffers.getDeviceBindings().data(), stream, nullptr))`
`{`
`    return false;`
`}`
`// Asynchronously copy data from device output buffers to host output buffers.`
`buffers.copyOutputToHostAsync(stream);`
 	
`// Wait for the work in the stream to complete.`
`CHECK(cudaStreamSynchronize(stream));`
 	
`// Release stream.`
`CHECK(cudaStreamDestroy(stream));`
	`cudaStream_t stream;`
	`CHECK(cudaStreamCreate(&stream));`
	
	`// Asynchronously copy data from host input buffers to device input buffers`
	`buffers.copyInputToDeviceAsync(stream);`
	
	`// Asynchronously enqueue the inference workif (!context->enqueueV3(stream))`
	`{`
	`    return false;`
	`}`
	`// Asynchronously copy data from device output buffers to host output buffers.`
	`buffers.copyOutputToHostAsync(stream);`
	
	`// Wait for the work in the stream to complete.`
	`CHECK(cudaStreamSynchronize(stream));`
	
	`// Release stream.`
	`CHECK(cudaStreamDestroy(stream));`

2.2. 64-Bit Dimension Changes

The dimensions held by Dims changed from `int32_t` to `int64_t`. However, in TensorRT 10.0, TensorRT will generally reject networks that use dimensions exceeding the range of `int32_t`. The tensor type returned by `IShapeLayer` is now `DataType::kINT64`. Use `ICastLayer` to cast the result to the tensor of type `DataType::kINT32` if 32-bit dimensions are required.

Inspect code that bitwise copies to and from Dims to ensure it is correct for `int64_t` dimensions.

2.3. Added C++ APIs

Enums

- `ActivationType::kGELU_ERF`
- `ActivationType::kGELU_TANH`
- `BuilderFlag::kREFIT_IDENTICAL`
- `BuilderFlag::kSTRIP_PLAN`
- `BuilderFlag::kWEIGHT_STREAMING`
- `Datatype::kINT4`
- `LayerType::kPLUGIN_V3`

Types

- APILanguage
- Dims64
- ExecutionContextAllocationStrategy
- IGpuAsyncAllocator
- InterfaceInfo
- IPluginResource
- IPluginV3
- IStreamReader
- IVersionedInterface

Methods and Properties

- getInferLibBuildVersion
- getInferLibMajorVersion
- getInferLibMinorVersion
- getInferLibPatchVersion
- ICudaEngine::createRefitter
- IcudaEngine::getMinimumWeightStreamingBudget
- IcudaEngine::getStreamableWeightsSize
- ICudaEngine::getWeightStreamingBudget
- IcudaEngine::isDebugTensor
- ICudaEngine::setWeightStreamingBudget
- IExecutionContext::getDebugListener
- IExecutionContext::getTensorDebugState
- IExecutionContext::setAllTensorsDebugState
- IExecutionContext::setDebugListener
- IExecutionContext::setOuputTensorAddress
- IExecutionContext::setTensorDebugState
- IExecutionContext::updateDeviceMemorySizeForShapes
- IGpuAllocator::allocateAsync
- IGpuAllocator::deallocateAsync
- INetworkDefinition::addPluginV3
- INetworkDefinition::isDebugTensor
- INetworkDefinition::markDebug
- INetworkDefinition::unmarkDebug

- `IPluginRegistry::acquirePluginResource`
- `IPluginRegistry::deregisterCreator`
- `IPluginRegistry::getAllCreators`
- `IPluginRegistry::getCreator`
- `IPluginRegistry::registerCreator`
- `IPluginRegistry::releasePluginResource`

2.4. Removed C++ APIs

Table 6. Removed C++ APIs and their Suggested Superseded API

C++ API	Superseded API
`BuilderFlag::kENABLE_TACTIC_HEURISTIC`	Builder optimization level 2
`BuilderFlag::kSTRICT_TYPES`	Use for all three flags: 1. `kREJECT_EMPTY_ALGORITHMS` 2. `kDIRECT_IO` 3. `kPREFER_PRECISION_CONSTRAINTS` Note: When removing enum members (for all enums in this list) we will be changing enumeration in the enum to have sequential numbers.
1. `EngineCapability::kDEFAULT` 2. `EngineCapability::kSAFE_DLA` 3. `EngineCapability::kSAFE_GPU`	1. `EngineCapability::kSTANDARD` 2. `EngineCapability::kDLA_STANDALONE` 3. `EngineCapability::kSAFETY`
`IAlgorithm::getAlgorithmIOInfo()`	`IAlgorithm::getAlgorithmIOInfoByIndex()`
`IAlgorithmIOInfo::getTensorFormat()`	The strides, data type, and vectorization information is sufficient to uniquely identify tensor formats.
1. `IBuilder::buildEngineWithConfig()` 2. `IBuilder::destroy()` 3. `IBuilder::getMaxBatchSize()` 4. `IBuilder::setMaxBatchSize()`	1. `IBuilder::buildSerializedNetwork()` 2. `delete ObjectName` 3. Implicit batch is no longer supported 4. Implicit batch is no longer supported
1. `IBuilderConfig::destroy()`	1. `delete ObjectName`

C++ API	Superseded API
2. `IBuilderConfig::getMaxWorkspaceSize()` 3. `IBuilderConfig::getMinTimingIterations()` 4. `IBuilderConfig::setMaxWorkspaceSize()` 5. `IBuilderConfig::setMinTimingIterations()`	2. `IBuilderConfig::getMemoryPoolLimit()` with `MemoryPoolType::kWORKSPACE` 3. `IBuilderConfig::getAvgTimingIterations()` 4. `IBuilderConfig::setMemoryPoolLimit()` with `MemoryPoolType::kWORKSPACE` 5. `IBuilderConfig::setAvgTimingIterations()`
1. `IConvolutionLayer::getDilation()` 2. `IConvolutionLayer::getKernelSize()` 3. `IConvolutionLayer::getPadding()` 4. `IConvolutionLayer::getStride()` 5. `IConvolutionLayer::setDilation()` 6. `IConvolutionLayer::setKernelSize()` 7. `IConvolutionLayer::setPadding()` 8. `IConvolutionLayer::setStride()`	1. `IConvolutionLayer::getDilationNd()` 2. `IConvolutionLayer::getKernelSizeNd()` 3. `IConvolutionLayer::getPaddingNd()` 4. `IConvolutionLayer::getStrideNd()` 5. `IConvolutionLayer::setDilationNd()` 6. `IConvolutionLayer::setKernelSizeNd()` 7. `IConvolutionLayer::setPaddingNd()` 8. `IConvolutionLayer::setStrideNd()`
1. `ICudaEngine::bindingIsInput()` 2. `ICudaEngine::destroy()` 3. `ICudaEngine::getBindingBytesPerComponent()` 4. `ICudaEngine::getBindingComponentsPerElement()` 5. `ICudaEngine::getBindingDataType()` 6. `ICudaEngine::getBindingDimensions()` 7. `ICudaEngine::getBindingFormat()` 8. `ICudaEngine::getBindingFormatDesc()` 9. `ICudaEngine::getBindingIndex()` 10. `ICudaEngine::getBindingName()` 11. `ICudaEngine::getBindingVectorizedDim()` 12. `ICudaEngine::getLocation()` 13. `ICudaEngine::getMaxBatchSize()` 14. `ICudaEngine::getNbBindings()` 15. `ICudaEngine::getProfileDimensions()` 16. `ICudaEngine::getProfileShapeValues()` 17. `ICudaEngine::hasImplicitBatchDimension()` 18. `ICudaEngine::isExecutionBinding()` 19. `ICudaEngine::isShapeBinding()`	1. `ICudaEngine::getTensorIOMode()` 2. `delete ObjectName` 3. `ICudaEngine::getTensorBytesPerComponent()` 4. `ICudaEngine::getTensorComponentsPerElement()` 5. `ICudaEngine::getTensorDataType()` 6. `ICudaEngine::getTensorShape()` 7. `ICudaEngine::getTensorFormat()` 8. `ICudaEngine::getTensorFormatDesc()` 9. Name-based methods 10. Name-based methods 11. `ICudaEngine::getTensorVectorizedDim()` 12. `ITensor::getLocation()` 13. Implicit batch is no longer supported 14. `ICudaEngine::getNbIOTensors()` 15. `ICudaEngine::getProfileShape()` 16. `ICudaEngine::getShapeValues()` 17. Implicit batch is no longer supported 18. No name-based equivalent replacement 19. `ICudaEngine::isShapeInferenceIO()`

C++ API	Superseded API
1. `IDeconvolutionLayer::getKernelSize()` 2. `IDeconvolutionLayer::getPadding()` 3. `IDeconvolutionLayer::getStride()` 4. `IDeconvolutionLayer::setKernelSize()` 5. `IDeconvolutionLayer::setPadding()` 6. `IDeconvolutionLayer::setStride()`	1. `IDeconvolutionLayer::getKernelSizeNd()` 2. `IDeconvolutionLayer::getPaddingNd()` 3. `IDeconvolutionLayer::getStrideNd()` 4. `IDeconvolutionLayer::setKernelSizeNd()` 5. `IDeconvolutionLayer::setPaddingNd()` 6. `IDeconvolutionLayer::setStrideNd()`
1. `IExecutionContext::destroy()` 2. `IExecutionContext::enqueue()` 3. `IExecutionContext::enqueueV2()` 4. `IExecutionContext::execute()` 5. `IExecutionContext::getBindingDimensions()` 6. `IExecutionContext::getShapeBinding()` 7. `IExecutionContext::getStrides()` 8. `IExecutionContext::setBindingDimensions()` 9. `IExecutionContext::setInputShapeBinding()` 10. `IExecutionContext::setOptimizationProfile()`	1. `delete ObjectName` 2. `IExecutionContext::enqueueV3()` 3. `IExecutionContext::enqueueV3()` 4. `IExecutionContext::executeV2()` 5. `IExecutionContext::getTensorShape()` 6. `IExecutionContext::getTensorAddress()` or `getOutputTensorAddress()` 7. `IExecutionContext::getTensorStrides()` 8. `IExecutionContext::setInputShape()` 9. `IExecutionContext::setInputTensorAddress()` or `setTensorAddress()` 10. `IExecutionContext::setOptimizationProfileAsync()`
`IFullyConnectedLayer`	`IMatrixMultiplyLayer`
`IGpuAllocator::free()`	`IGpuAllocator::deallocate()`
`IHostMemory::destroy()`	`delete ObjectName`
1. `INetworkDefinition::addConvolution()` 2. `INetworkDefinition::addDeconvolution()` 3. `INetworkDefinition::addFullyConnected()` 4. `INetworkDefinition::addPadding()` 5. `INetworkDefinition::addPooling()` 6. `INetworkDefinition::addRNNv2()` 7. `INetworkDefinition::destroy()` 8. `INetworkDefinition::hasExplicitPrecision()` 9. `INetworkDefinition::hasImplicitBatchDimension()`	1. `INetworkDefinition::addConvolutionNd()` 2. `INetworkDefinition::addDeconvolutionNd()` 3. `INetworkDefinition::addMatrixMultiply()` 4. `INetworkDefinition::addPaddingNd()` 5. `INetworkDefinition::addPoolingNd()` 6. `INetworkDefinition::addLoop()` 7. `delete ObjectName` 8. Explicit precision support is removed in 10.0 9. Implicit batch support is removed
`IOnnxConfig::destroy()`	`delete ObjectName`
1. `IPaddingLayer::getPostPadding()`	1. `IPaddingLayer::getPostPaddingNd()`

C++ API	Superseded API
2. `IPaddingLayer::getPrePadding()` 3. `IPaddingLayer::setPostPadding()` 4. `IPaddingLayer::setPrePadding()`	2. `IPaddingLayer::getPrePaddingNd()` 3. `IPaddingLayer::setPostPaddingNd()` 4. `IPaddingLayer::setPrePaddingNd()`
1. `IPoolingLayer::getPadding()` 2. `IPoolingLayer::getStride()` 3. `IPoolingLayer::getWindowSize()` 4. `IPoolingLayer::setPadding()` 5. `IPoolingLayer::setStride()` 6. `IPoolingLayer::setWindowSize()`	1. `IPoolingLayer::getPaddingNd()` 2. `IPoolingLayer::getStrideNd()` 3. `IPoolingLayer::getWindowSizeNd()` 4. `IPoolingLayer::setPaddingNd()` 5. `IPoolingLayer::setStrideNd()` 6. `IPoolingLayer::setWindowSizeNd()`
`IRefitter::destroy()`	`delete ObjectName`
1. `IResizeLayer::getAlignCorners()` 2. `IResizeLayer::setAlignCorners()`	1. `IResizeLayer::getAlignCornersNd()` 2. `IResizeLayer::setAlignCornersNd()`
1. `IRuntime::deserializeCudaEngine(void const* blob, std::size_t size, IPluginFactory* pluginFactory)` 2. `IRuntime::destroy()`	1. Use `deserializeCudaEngine` with two parameters 2. `delete ObjectName`
`IRNNv2Layer`	`ILoop`
`kNV_TENSORRT_VERSION_IMPL`	`#define NV_TENSORRT_VERSION_INT(major, minor, patch) ((major) *10000L + (minor) *100L + (patch) *1L)` Note: TensorRT version encoding was changed to accommodate a two digit minor version.
1. `NetworkDefinitionCreationFlag::kEXPLICIT_BATCH` 2. `NetworkDefinitionCreationFlag::kEXPLICIT_PRECISION`	Support is removed in 10.0
1. `NV_TENSORRT_SONAME_MAJOR` 2. `NV_TENSORRT_SONAME_MINOR` 3. `NV_TENSORRT_SONAME_PATCH`	1. `NV_TENSORRT_MAJOR` 2. `NV_TENSORRT_MINOR` 3. `NV_TENSORRT_PATCH`
1. `PaddingMode::kCAFFE_ROUND_DOWN` 2. `PaddingMode::kCAFFE_ROUND_UP`	Caffe is not supported since 9.0

C++ API	Superseded API
1. `PreviewFeature::kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805` 2. `PreviewFeature::kFASTER_DYNAMIC_SHAPES_0805`	1. External tactics are always disabled for core code 2. This flag is on by default
1. `ProfilingVerbosity::kDEFAULT` 2. `ProfilingVerbosity::kVERBOSE`	1. `ProfilingVerbosity::kLAYER_NAMES_ONLY` 2. `ProfilingVerbosity::kDETAILED`
`ResizeMode`	Use `InterpolationMode`, alias is removed
1. `RNNDirection` 2. `RNNGateType` 3. `RNNInputMode` 4. `RNNOperation`	RNN related data structures are removed
`SampleMode::kDEFAULT`	`SampleMode::kSTRICT_BOUNDS`
`SliceMode`	Use `SampleMode`, alias is removed

2.5. Removed C++ Plugins

Table 7. Removed C++ Plugins and their Suggested Superseded Plugin

C++ Plugin	Superseded Plugin
1. `createAnchorGeneratorPlugin()` 2. `createBatchedNMSPlugin()` 3. `createInstanceNormalizationPlugin()` 4. `createNMSPlugin()` 5. `createNormalizePlugin()` 6. `createPriorBoxPlugin()` 7. `createRegionPlugin()` 8. `createReorgPlugin()` 9. `createRPNROIPlugin()` 10. `createSplitPlugin()`	1. `GridAnchorPluginCreator::createPlugin()` 2. `BatchedNMSPluginCreator::createPlugin()` 3. `InstanceNormalizationPluginCreator::createPlugin()` 4. `NMSPluginCreator::createPlugin()` 5. `NormalizePluginCreator::createPlugin()` 6. `PriorBoxPluginCreator::createPlugin()` 7. `RegionPluginCreator::createPlugin()` 8. `ReorgPluginCreator::createPlugin()` 9. `RPROIPluginCreator::createPlugin()` 10. `INetworkDefinition::addSlice()`
`struct Quadruple`	Related plugins are removed

2.6. Removed Safety C++ APIs

Table 8. Removed Safety C++ APIs and their Suggested Superseded Safety API

Safety C++ API	Superseded Safety API
1. `safe::ICudaEngine::bindingIsInput()`	1. `safe::ICudaEngine::tensorIOMode()`
2. `safe::ICudaEngine::getBindingBytesPerComponent()`	2. `safe::ICudaEngine::getTensorBytesPerComponent()`
3. `safe::ICudaEngine::getBindingComponentsPerElement()`	3. `safe::ICudaEngine::getTensorComponentsPerElement()`
4. `safe::ICudaEngine::getBindingDataType()`	4. `safe::ICudaEngine::getTensorDataType()`
5. `safe::ICudaEngine::getBindingDimensions()`	5. `safe::ICudaEngine::getTensorShape()`
6. `safe::ICudaEngine::getBindingIndex()`	6. `safe::name-based methods`
7. `safe::ICudaEngine::getBindingName()`	7. `safe::name-based methods`
8. `safe::ICudaEngine::getBindingVectorizedDim()`	8. `safe::ICudaEngine::getTensorVectorizedDim()`
9. `safe::ICudaEngine::getNbBindings()`	9. `safe::ICudaEngine::getNbIOTensors()`
10. `safe::ICudaEngine::getTensorFormat()`	10. `safe::ICudaEngine::getBindingFormat()`
1. `safe::IExecutionContext::enqueueV2()`	1. `safe::IExecutionContext::enqueueV3()`
2. `safe::IExecutionContext::getStrides()`	2. `safe::IExecutionContext::getTensorStrides()`

Chapter 3. trtexec

3.1. trtexec Flag Changes

Table 9. Changes to flag `workspace` and `minTiming`

TensorRT 8.x	TensorRT 10.0

`trtexec \`
`  --onnx=/path/to/model.onnx \`
`  --saveEngine=/path/to/engine.trt \`
`  --optShapes=input:$INPUT_SHAPE \`
`  --avgTiming=1 \`
`  --workspace=1024 \`
`  --minTiming=1`
	`trtexec \`
	`  --onnx=/path/to/model.onnx \`
	`  --saveEngine=/path/to/engine.trt \`
	`  --optShapes=input:$INPUT_SHAPE \`
	`  --avgTiming=1 \`
	`  --memPoolSize=workspace:1024`

3.2. Removed trtexec Flags

Table 10. Removed trtexec Flags and their Suggested Superseded Flag

trtexec Flag	Superseded Flag
`--minTiming`	`avgTiming`
`--preview=features` options: ▶ `disableExternalTacticSourcesForCore0805` ▶ `fasterDynamicShapes0805`	N/A
`--workspace=N`	`--memPoolSize=poolspec`

3.3. Deprecated trtexec Flags

Table 11. Deprecated trtexec Flags

trtexec Flag
`--buildOnly`
`--explicitPrecision`
`--heuristic`
`--nvtxMode`

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Arm

Arm, AMBA and Arm Powered are registered trademarks of Arm Limited. Cortex, MPCore and Mali are trademarks of Arm Limited. "Arm" is used to represent Arm Holdings plc; its operating company Arm Limited; and the regional subsidiaries Arm Inc.; Arm KK; Arm Korea Limited.; Arm Taiwan Limited; Arm France SAS; Arm Consulting (Shanghai) Co. Ltd.; Arm Germany GmbH; Arm Embedded Technologies Pvt. Ltd.; Arm Norway, AS and Arm Sweden AB.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

Blackberry/QNX

Copyright © 2020 BlackBerry Limited. All rights reserved.

Trademarks, including but not limited to BLACKBERRY, EMBLEM Design, QNX, AVIAGE, MOMENTICS, NEUTRINO and QNX CAR are the trademarks or registered trademarks of BlackBerry Limited, used under license, and the exclusive rights to such trademarks are expressly reserved.

Google

Android, Android TV, Google Play and the Google Play logo are trademarks of Google, Inc.

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com

Trademarks

NVIDIA, the NVIDIA logo, and BlueField, CUDA, DALI, DRIVE, Hopper, JetPack, Jetson AGX Xavier, Jetson Nano, Maxwell, NGC, Nsight, Orin, Pascal, Quadro, Tegra, TensorRT, Triton, Turing and Volta are trademarks and/or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com