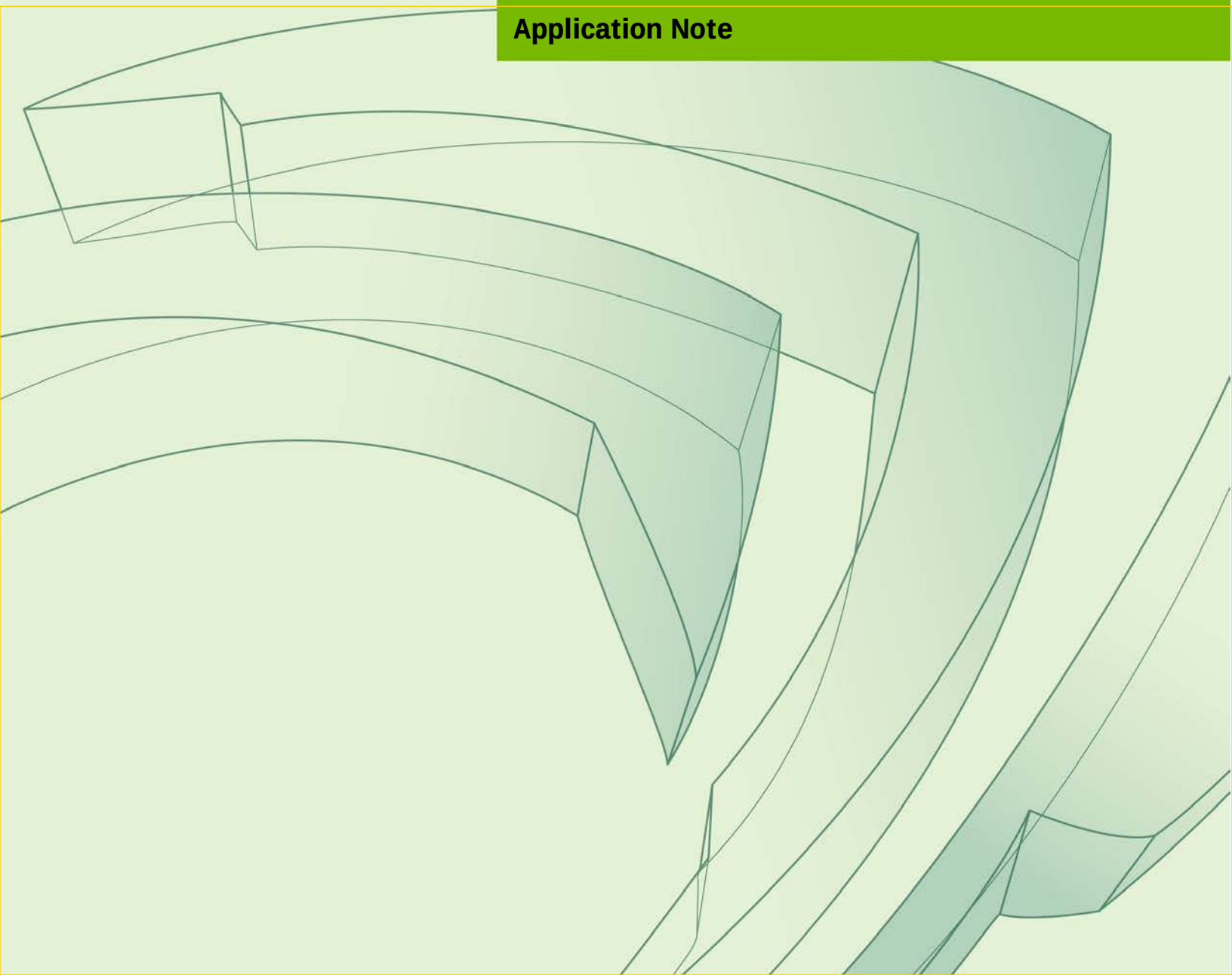
DEEPSTREAM 4.0 PLUGIN MANUAL

SWE-SWDOCDPSTR-002-PGRF | August 6, 2019
Advance Information | Subject to Change

Application Note



DOCUMENT CHANGE HISTORY

Date	Author	Revision History
Nov. 19, 2018	Bhushan Rupde, Jonathan Sachs	Release 3.0 (Initial release)
Aug. 6, 2019	Bhushan Rupde, Jonathan Sachs	Release 4.0 (Unified release)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 INTRODUCTION

DeepStream SDK is based on the GStreamer framework. This manual describes the DeepStream GStreamer plugins and the DeepStream input, outputs, and control parameters.

DeepStream SDK is supported on systems that contain an NVIDIA® Jetson™ module or an NVIDIA dGPU adapter.[1]

The manual is intended for engineers who want to develop DeepStream applications or additional plugins using the DeepStream SDK. It also contains information about metadata used in the SDK. Developers can add custom metadata as well.

The manual describes the methods defined in the SDK for implementing custom inferencing layers using the `IPlugin` interface of TensorRT™.

You can refer the sample examples shipped with the SDK as you use this manual to familiarize yourself with DeepStream application and plugin development.

[1] This manual uses the term *dGPU* ("discrete GPU") to refer to NVIDIA GPU expansion card products such as NVIDIA® Tesla® T4 and P4, NVIDIA® GeForce® GTX 1080, and NVIDIA® GeForce® RTX 2080. This version of DeepStream SDK runs on specific dGPU products on x86_64 platforms supported by NVIDIA driver 418+ and NVIDIA® TensorRT™ 5.1 and later versions.

2.0 GSTREAMER PLUGIN DETAILS

2.1 GST-NVINFER

The `Gst-nvinfer` plugin does inferencing on input data using NVIDIA® TensorRT™.

The plugin accepts batched NV12/RGBA buffers from upstream. The `NvDsBatchMeta` structure must already be attached to the Gst Buffers.

The low-level library (`libnvds_infer`) operates on any of INT8 RGB, BGR, or GRAY data with dimension of Network Height and Network Width.

The `Gst-nvinfer` plugin performs transforms (format conversion and scaling), on the input frame based on network requirements, and passes the transformed data to the low-level library.

The low-level library preprocesses the transformed frames (performs normalization and mean subtraction) and produces final float RGB/BGR/GRAY planar data which is passed to the TensorRT engine for inferencing. The output type generated by the low-level library depends on the network type.

The pre-processing function is:

$$y = net\text{-}scale\text{-}factor * (x - mean)$$

Where:

- x is the input pixel value. It is an *int8* with range [0,255].
- *mean* is the corresponding mean value, read either from the mean file or as `offsets[c]`, where `c` is the channel to which the input pixel belongs, and `offsets` is the array specified in the configuration file. It is a *float*.
- *net-scale-factor* is the pixel scaling factor specified in the configuration file. It is a *float*.
- y is the corresponding output pixel value. It is a *float*.

`nvinfer` currently works on the following type of networks:

- Multi-class object detection
- Multi-label classification
- Segmentation

The `Gst-nvinfer` plugin can work in two modes:

- **Primary mode**: Operates on full frames
- **Secondary mode**: Operates on objects added in the meta by upstream components

When the plugin is operating as a secondary classifier along with the tracker, it tries to improve performance by avoiding re-inferencing on the same objects in every frame. It does this by caching the classification output in a map with the object's unique ID as the key. The object is inferred upon only when it is first seen in a frame (based on its object ID) or when the size (bounding box area) of the object increases by 20% or more. This optimization is possible only when the tracker is added as an upstream element.

Detailed documentation of the TensorRT interface is available at:

```
https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html
```

The plugin supports the `IPlugin` interface for custom layers. Refer to section IPlugin Interface for details.

The plugin also supports the interface for custom functions for parsing outputs of object detectors and initialization of non-image input layers in cases where there are more than one input layer.

Refer to `sources/includes/nvdsinfer_custom_impl.h` for the custom method implementations for custom models.

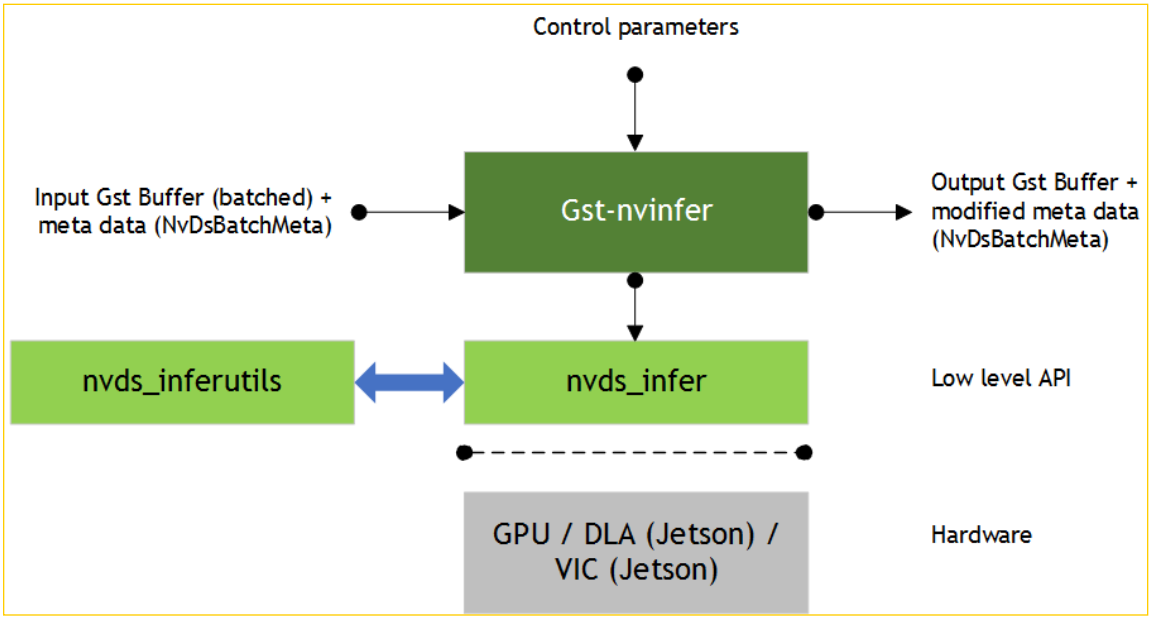


Figure 1. Gst-nvinfer inputs and outputs

Downstream components receive a Gst Buffer with unmodified contents plus the metadata created from the inference output of the `Gst-nvinfer` plugin.

The plugin can be used for cascaded inferencing. That is, it can perform **primary inferencing** directly on input data, then perform **secondary inferencing** on the results of primary inferencing, and so on. See the sample application `deepstream-test2` for more details.

2.1.1 Inputs and Outputs

This section summarizes the inputs, outputs, and communication facilities of the Gst-nvinfer plugin.

- Inputs
  - Gst Buffer
  - `NvDsBatchMeta` (attaching `NvDsFrameMeta`)
  - Caffe Model and Caffe Prototxt
  - ONNX
  - UFF file
  - TLT Encoded Model and Key
  - Offline: Supports engine files generated by Transfer Learning Toolkit SDK Model converters

Layers: Supports all layers supported by TensorRT, see:

```
https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html
```

- Control parameters: `Gst-nvinfer` gets control parameters from a configuration file. You can specify this by setting the property `config-file-path`. For details, see Gst-nvinfer File Configuration Specifications. Other control parameters that can be set through `GObject` properties are:
  - Batch size
  - Inference interval
  - Attach inference tensor outputs as buffer metadata
- The parameters set through the GObject properties override the parameters in the Gst-nvinfer configuration file.
- Outputs
  - Gst Buffer
  - Depending on network type and configured parameters, one or more of:
    - NvDsObjectMeta
    - NvDsClassifierMeta
    - NvDsInferSegmentationMeta
    - NvDsInferTensorMeta

2.1.2 Features

Table 1 summarizes the features of the plugin.

Table 1. Features of the Gst-nvinfer plugin

Feature	Description	Release
Transfer-Learning-Toolkit encoded model support	—	DS 4.0
Gray input model support	Support for models with single channel gray input	DS 4.0
Tensor output as meta	Raw tensor output is attached as meta data to Gst Buffers and flowed through the pipeline	DS 4.0
Segmentation model	Supports segmentation model	DS 4.0
Maintain input aspect ratio	Configurable support for maintaining aspect ratio when scaling input frame to network resolution	DS 4.0

Feature	Description	Release
Custom cuda engine creation interface	Interface for generating CUDA engines from TensorRT `INetworkDefinition` and `IBuilder` APIs instead of model files	DS 4.0
Caffe Model support	—	DS 2.0
UFF Model support	—	DS 3.0
ONNX Model support	—	DS 3.0
Multiple modes of operation	Support for cascaded inferencing	DS 2.0
Asynchronous mode of operation for secondary inferencing	Infer asynchronously for secondary classifiers	DS 2.0
Grouping using CV::Group rectangles	For detector bounding box clustering	DS 2.0
Configurable batch-size processing	User can configure batch size for processing	DS 2.0
No Restriction on number of output blobs	Supports any number of output blobs	DS 3.0
Configurable number of detected classes (detectors)	Supports configurable number of detected classes	DS 3.0
Support for Classes: configurable (> 32)	Support any number of classes	DS 3.0
Application access to raw inference output	Application can access inference output buffers for user specified layer	DS 3.0
Support for single shot detector (SSD)	—	DS 3.0
Secondary GPU Inference Engines (GIEs) operate as detector on primary bounding box	Support secondary inferencing as detector	DS 2.0
Multiclass secondary support	Support multiple classifier network outputs	DS 2.0
Grouping using DBSCAN	For detector bounding box clustering	DS 3.0
Loading an external lib containing IPlugin implementation for custom layers (IPluginCreator & IPluginFactory)	Supports loading (`dlopen()`) a library containing IPlugin implementation for custom layers	DS 3.0
Multi GPU	Select GPU on which we want to run inference	DS 2.0
Detection width height configuration	Filter out detected objects based on min/max object size threshold	DS 2.0
Allow user to register custom parser	Supports final output layer bounding box parsing for custom detector network	DS 2.0
Bounding box filtering based on configurable object size	Supports inferencing in secondary mode objects meeting min/max size threshold	DS 2.0
Configurable operation interval	Interval for inferencing (number of batched buffers skipped)	DS 2.0
Select Top and bottom regions of interest (RoIs)	Removes detected objects in top and bottom areas	DS 2.0
Operate on Specific object type (Secondary mode)	Process only objects of define classes for secondary inferencing	DS 2.0

Feature	Description	Release
Configurable blob names for parsing bounding box (detector)	Support configurable names for output blobs for detectors	DS 2.0
Allow configuration file input	Support configuration file as input (mandatory in DS 3.0)	DS 2.0
Allow selection of class id for operation	Supports secondary inferencing based on class ID	DS 2.0
Support for Full Frame Inference: Primary as a classifier	Can work as classifier as well in primary mode	DS 2.0
Multiclass secondary support	Support multiple classifier network outputs	DS 2.0
Secondary GIEs operate as detector on primary bounding box Support secondary inferencing as detector	—	DS 2.0
Supports FP16, FP32 and INT8 models FP16 and INT8 are platform dependent	—	DS 2.0
Supports TensorRT Engine file as input	—	DS 2.0
Inference input layer initialization Initializing non-video input layers in case of more than one input layers	—	DS 3.0
Support for FasterRCNN	—	DS 3.0
Support for Yolo detector (YoloV3/V3-tiny/V2/V2-tiny)	—	DS 4.0

2.1.3 Gst-nvinfer File Configuration Specifications

The `Gst-nvinfer` configuration file uses a "Key File" format described in:

```
https://specifications.freedesktop.org/desktop-entry-spec/latest
```

The `[property]` group configures the general behavior of the plugin. It is the only mandatory group.

The `[class-attrs-all]` group configures detection parameters for all classes.

The `[class-attrs-<class-id>]` group configures detection parameters for a class specified by `<class-id>`. For example, the `[class-attrs-23]` group configures detection parameters for class ID 23. This type of group has the same keys as `[class-attrs-all]`.

Table 2 and Table 3, respectively describe the keys supported for `[property]` groups and `[class-attrs-…]` groups.

Table 2. Gst-nvinfer plugin, [property] group, supported keys

Property	Meaning	Type and Range	Example *Notes*	Network Types / *Applicable to GIEs (Primary/Seconday)*
num-detected-classes	Number of classes detected by the network	Integer, >0	num-detected-classes=91	Detector *Both*
net-scale-factor	Pixel normalization factor	Float, >0.0	net-scale-factor=0.031	All *Both*
model-file	Pathname of the caffemodel file. Not required if `model-engine-file` is used	String	model-file=/home/ubuntu/model.caffemodel	All *Both*
proto-file	Pathname of the prototxt file. Not required if `model-engine-file` is used	String	proto-file=/home/**ubuntu**/model.prototxt	All *Both*
int8-calib-file	Pathname of the INT8 calibration file for dynamic range adjustment with an FP32 model	String	int8-calib-file=/home/ubuntu/int8_calib	All *Both*
batch-size	Number of frames or objects to be inferred together in a batch	Integer, >0	batch-size=30	All *Both*
model-engine-file	Pathname of the serialized model engine file	String	model-engine-file=/home/ubuntu/model.engine	All *Both*
uff-file	Pathname of the UFF model file	String	uff-file=/home/ubuntu/model.uff	All *Both*
onnx-file	Pathname of the ONNX model file	String	onnx-file=/home/ubuntu/model.onnx	All *Both*
enable-dbscan	Indicates whether to use DBSCAN or the OpenCV `groupRectangles()` function for grouping detected objects	Boolean	enable-dbscan=1	Detector *Both*
labelfile-path	Pathname of a text file containing the labels for the model	String	labelfile-path=/home/ubuntu/model_labels.txt	Detector & classifier *Both*
mean-file	Pathname of mean data file (PPM format)	String	mean-file=/home/ubuntu/model_meanfile.ppm	All *Both*

Network Types / *Applicable to GIEs (Primary/Seconday)*				
Property	**Meaning**	**Type and Range**	**Example** *Notes*	
gie-unique-id	Unique ID to be assigned to the GIE to enable the application and other elements to identify detected bounding boxes and labels	Integer, >0	gie-unique-id=2	All *Both*
operate-on-gie-id	Unique ID of the GIE on whose metadata (bounding boxes) this GIE is to operate on	Integer, >0	operate-on-gie-id=1	All *Both*
operate-on-class-ids	Class IDs of the parent GIE on which this GIE is to operate on	Semicolon delimited integer array	operate-on-class-ids=1;2 *Operates on objects with class IDs 1, 2 generated by parent GIE*	All *Both*
interval	Specifies the number of consecutive batches to be skipped for inference	Integer, >0	interval=1	All *Primary*
input-object-min-width	Secondary GIE infers only on objects with this minimum width	Integer, ≥0	input-object-min-width=40	All *Secondary*
input-object-min-height	Secondary GIE infers only on objects with this minimum height	Integer, ≥0	input-object-min-height=40	All *Secondary*
input-object-max-width	Secondary GIE infers only on objects with this maximum width	Integer, ≥0	input-object-max-width=256 *0 disables the threshold*	All *Secondary*
input-object-max-height	Secondary GIE infers only on objects with this maximum height	Integer, ≥0	input-object-max-height=256 *0 disables the threshold*	All *Secondary*
uff-input-dims	Dimensions of the UFF model	channel; height; width; input-order All integers, ≥0	input-dims=3;224;224;0 *Possible values for input-order are: 0: NCHW 1: NHWC*	All *Both*
network-mode	Data format to be used by inference	Integer 0: FP32 1: INT8 2: FP16	network-mode=0	All *Both*

Network Types / *Applicable to GIEs (Primary/Seconday)*				
Property	**Meaning**	**Type and Range**	**Example** *Notes*	
offsets	Array of mean values of color components to be subtracted from each pixel. Array length must equal the number of color components in the frame. The plugin multiplies mean values by `net-scale-factor`.	Semicolon delimited float array, all values ≥0	offsets=77.5;21.2;11.8	All *Both*
output-blob-names	Array of output layer names	Semicolon delimited string array	**For detector**: output-blob-names=coverage;bbox **For multi-label classifiers**: output-blob-names=coverage_attrib1;coverage_attrib2	All *Both*
parse-bbox-func-name	Name of the custom bounding box parsing function. If not specified, Gst-nvinfer uses the internal function for the resnet model provided by the SDK.	String	parse-bbox-func-name= parse_bbox_resnet	Detector *Both*
custom-lib-path	Absolute pathname of a library containing custom method implementations for custom models	String	custom-lib-path= /home/ubuntu/ libresnet_custom_impl.so	All *Both*
model-color-format	Color format required by the model.	Integer 0: RGB 1: BGR	model-color-format=0	All *Both*

Network Types / *Applicable to GIEs (Primary/Seconday)*				
Property	**Meaning**	**Type and Range**	**Example** *Notes*	
classifier-async-mode	Enables inference on detected objects and asynchronous metadata attachments. Works only when tracker-ids are attached. Pushes buffer downstream without waiting for inference results. Attaches metadata after the inference results are available to next Gst Buffer in its internal queue.	Boolean	classifier-async-mode=1	Classifier *Secondary*
process-mode	Mode (primary or secondary) in which the element is to operate on	Integer 1=Primary 2=Secondary	gie-mode=1	All *Both*
classifier-threshold	Minimum threshold label probability. The GIE outputs the label having the highest probability if it is greater than this threshold	Float, ≥0	classifier-threshold=0.4	Classifier *Both*
uff-input-blob-name	Name of the input blob in the UFF file	String	uff-input-blob-name=Input_1	All *Both*
secondary-reinfer-interval	Reinference interval for objects, in frames	Integer, ≥0	secondary-reinfer-interval=15	Classifier *Secondary*
output-tensor-meta	Gst-nvinfer attaches raw tensor output as Gst Buffer metadata.	Boolean	output-tensor-meta=1	All *Both*
enable-dla	Indicates whether to use the DLA engine for inferencing. **Note**: DLA is supported only on Jetson AGX Xavier™. Currently work in progress.	Boolean	enable-dla=1	All *Both*
use-dla-core	DLA core to be used. **Note**: Supported only on Jetson AGX Xavier™. Currently work in progress.	Integer, ≥0	use-dla-core=0	All *Both*

Network Types / *Applicable to GIEs (Primary/Seconday)*				
Property	**Meaning**	**Type and Range**	**Example** ***Notes***	
network-type	Type of network	Integer 0: Detector 1: Classifier 2: Segmentation	network-type=1	All *Both*
maintain-aspect-ratio	Indicates whether to maintain aspect ratio while scaling input.	Boolean	maintain-aspect-ratio=1	All *Both*
parse-classifier-func-name	Name of the custom classifier output parsing function. If not specified, Gst-nvinfer uses the internal parsing function for softmax layers.	String	parse-classifier-func-name= parse_bbox_softmax	Classifier *Both*
custom-network-config	Pathname of the configuration file for custom networks available in the custom interface for creating CUDA engines.	String	custom-network-config=/home/ubuntu/ network.config	All *Both*
tlt-encoded-model	Pathname of the Transfer Learning Toolkit (TLT) encoded model.	String	tlt-encoded-model= /home/ubuntu/ model.etlt	All *Both*
tlt-model-key	Key for the TLT encoded model.	String	tlt-model-key=abc	All *Both*
segmentation-threshold	Confidence threshold for the segmentation model to output a valid class for a pixel. If confidence is less than this threshold, class output for that pixel is -1.	Float, ≥0.0	segmentation-threshold=0.3	Segmentation *Both*

Table 3. Gst-nvinfer plugin, [class-attrs-...] groups, supported keys

Detector or Classifier / *Applicable to GIEs (Primary/Seconday)*				
Name	**Description**	**Type and Range**	**Example** ***Notes***	
threshold	Detection threshold	Float, >=0	threshold=0.5	Object detector *Both*

Detector or Classifier / *Applicable to GIEs (Primary/Seconday)*				
Name	**Description**	**Type and Range**	**Example** *Notes*	
eps	Epsilon values for OpenCV `grouprectangles()` function and DBSCAN algorithm	Float, >=0	eps=0.2	Object detector *Both*
group-threshold	Threshold value for rectangle merging for OpenCV `grouprectangles()` function	Integer, >=0	group-threshold=1 *0 disables the clustering functionality*	Object detector *Both*
minBoxes	Minimum number of points required to form a dense region for DBSCAN algorithm	Integer, ≥0	minBoxes=1 *0 disables the clustering functionality*	Object detector *Both*
roi-top-offset	Offset of the RoI from the top of the frame. Only objects within the RoI are output.	Integer, ≥0	roi-top-offset=200	Object detector *Both*
roi-bottom-offset	Offset of the RoI from the bottom of the frame. Only objects within the RoI are output.	Integer, ≥0	roi-bottom-offset=200	Object detector *Both*
detected-min-w	Minimum width in pixels of detected objects to be output by the GIE	Integer, ≥0	detected-min-w=64	Object detector *Both*
detected-min-h	Minimum height in pixels of detected objects to be output by the GIE	Integer, ≥0	detected-min-h=64	Object detector *Both*
detected-max-w	Maximum width in pixels of detected objects to be output by the GIE	Integer, ≥0	detected-max-w=200 *0 disables the property*	Object detector *Both*
detected-max-h	Maximum height in pixels of detected objects to be output by the GIE	Integer, ≥0	detected-max-h=200 *0 disables the property*	Object detector *Both*

2.1.4 Gst Properties

The values set through Gst properties override the values of properties in the configuration file. The application does this for certain properties that it needs to set programmatically.