DEEPSTREAM SDK 5.1 FOR NVIDIA DGPU AND JETSON

RN-09353-003 | February 24, 2021
Advance Information | Subject to Change

5.1 Release Notes

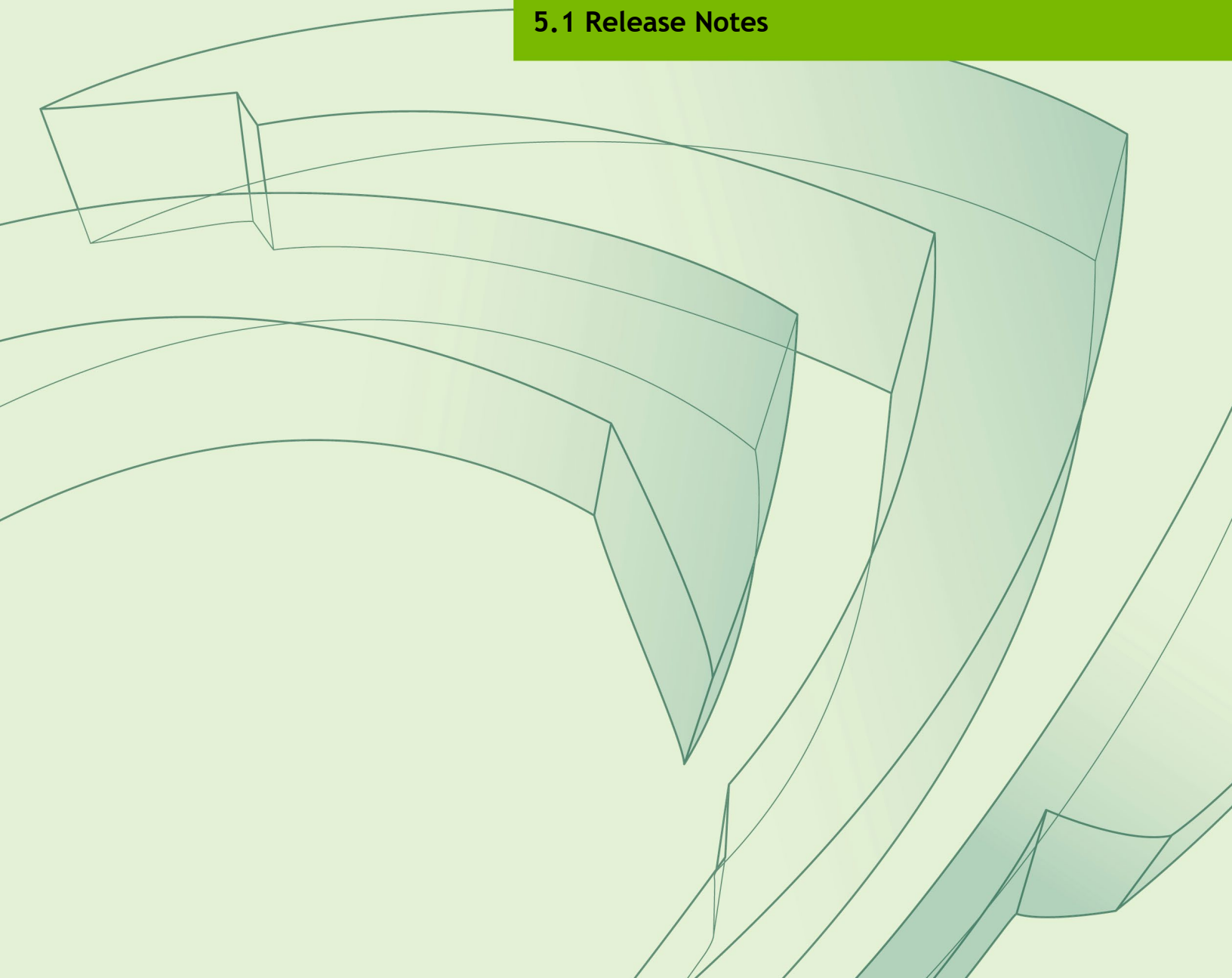

TABLE OF CONTENTS

1.0 ABOUT THIS RELEASE

These release notes are for the NVIDIA® DeepStream SDK for NVIDIA® Tesla®, NVIDIA® Ampere® NVIDIA® Jetson AGX Xavier™, NVIDIA® Jetson Xavier™ NX, NVIDIA® Jetson Nano™, and NVIDIA® Jetson™ TX2 platforms.

1.1 WHAT'S NEW

The following new features are supported in this DeepStream SDK release:

1.1.1 DS 5.1

- New plugins:
  - Gst-audio/video template plugins for implementing custom algorithms (non Gstreamer based)
  - Gst-inferaudio (alpha quality) plugin for supporting audio classifier.
  - New nvstreammux (alpha quality) can be enabled by exporting `USE_NEW_NVSTREAMMUX=yes`. For more information, see the "Gst-nvstreammux New Alpha" section in the *NVIDIA DeepStream SDK Developer Guide 5.1 Release*.

Note: **The old nvstreammux functionality will be deprecated in future.**

- New sample applications:
  - Smart Record example: Demonstrates event based smart record functionality.
  - deepstream-audio: Audio App to show Audio classifier usage.
- Enhancements/Bug fixes:
  - Fixes for Memory leaks and RTSP for improved stability
  - NvDCF tracker enhancements

- Redis message broker support
- Python binding miscellaneous fixes
- Misc. fixes in nvinfer, muxer, osd and video convert plugins

- Python binding enhancements:
  - Python Bindings for `NvDsPastFrameObj` and casting
  - Python Bindings for optical flow plugin
  - Python Bindings for segmentation plugin
  - Allow write access to frame images (some restriction apply)
- New sample applications in Python:
  - Optical Flow example: expose flow vectors as NumPy array.
  - Segmentation example: expose segmentation map as NumPy array.
  - Analytics example: demonstrate analytics plugin and metadata usage.

1.1.2 DS 5.0.1

Bug fixes in:

- Demuxer and Dewarper components
- Python Bindings

1.1.3 DS 5.0 (GA)

- Support for Triton Inference Server
- On the fly Model updates
- Event based Smart Record
- Cloud to device messaging
- Improved NVDCF tracker
- Facility to attach encoded detected Image objects as meta data.
- Sample application which showcases use of opencv in `dsexample` plugin
- Native Red Hat Enterprise Linux support
- Python Bindings incorporated into SDK
- Transfer learning toolkit Models from https://github.com/NVIDIA-AI-IOT/deepstream_tao_apps integrated into SDK
- Python binding enhancements
  - Access to frame image data as NumPy array
  - Access to inference output tensor data
  - Additional sample applications
    - Probe for image data, then use OpenCV to annotate and save frames to file
    - Probe for inference output tensors to parse in Python
    - USB camera input
    - RTSP stream output

- Better time-stamp handling for live RTSP cameras
- DRC stream support for Jetson
- 10 Bit H264 and H265 stream support
- Misc. bug fixes and Improved stability
- Support for flowing Metadata attached before `Gst-nvv4l2` decoder
- Gst-nvinfer plugin:
  - Support for TensorRT 7.0+:
  - Explicit Full Dimension Network Support
  - Non-maximum Suppression (NMS) for bounding box Clustering
  - On-the-fly model update (Engine/Plan file only)
  - Support for `yolov3-spp` detector
  - Support for Mask-RCNN Instance segmentation
- New plugins:
  - Gst-nvdsanalytics plugin for ROI detection, line crossing and direction detection.
  - Gst-nvinferserver plugin for supporting Triton inference server using C++ client APIs (https://docs.nvidia.com/deeplearning/sdk/triton-inference-server-master-branch-guide/docs/index.html).
- New sample applications:
  - Analytics example: Demonstrates batched analytics like ROI filtering, Line crossing, direction detection and overcrowding.
  - OpenCV example: Demonstrates the use of OpenCV in dsexample plugin.
  - Image as Metadata example: Demonstrates how to attach encoded object image as meta data and save the images in jpeg format.
  - Appsrc and Appsink example: Demonstrates AppSrc and AppSink usage for consuming and giving data from non-DS code.
  - Transfer learning example: Demonstrates a mechanism to save the images for objects which have lesser confidence. This can further be used for Model training.
  - Mask-RCNN example: Demonstrates Instance segmentation using Mask-RCNN model along with sending mask information to cloud.
- Source code release for the below plugin:
  - Gst-nvdsosd: GStreamer Gst-nvdsosd plugin for overlaying bounding boxes, lines, text, object masks.

Note: **DeepStream pre 5.1 and starting from DeepStream 4.0 Applications can be migrated to DeepStream 5.1. Refer to the "Application Migration to DeepStream 5.1" section in the *NVIDIA DeepStream SDK Developer Guide 5.1 Release*.**

1.2 CONTENTS OF THIS RELEASE

This release includes the following:

- The DeepStream SDK. Refer to *NVIDIA DeepStream SDK Developer Guide 5.1 Release* for a detailed description of the contents of the DeepStream release package. The *Developer Guide* also contains other information to help you get started with DeepStream, including information about system software and hardware requirements and external software dependencies that you must install before you use the SDK.

  - For detailed information about GStreamer plugins, metadata usage, see the "DeepStream Plugin Guide" section in the *NVIDIA DeepStream SDK Developer Guide 5.1 Release*.
  - For detailed troubleshooting information and frequently asked questions, see the "DeepStream Troubleshooting and FAQ Guide" section in the *NVIDIA DeepStream SDK Developer Guide 5.1 Release*.

- DeepStream SDK for dGPU and Jetson Software License Agreement (SLA).
- `LICENSE.txt` contains the license terms of third-party libraries used.

1.3 DOCUMENTATION IN THIS RELEASE

This release contains the following documentation.

- *NVIDIA DeepStream SDK Developer Guide 5.1 Release*
- *NVIDIA DeepStream SDK API Reference*
- *NVIDIA DeepStream Python API Reference*

1.4 DIFFERENCES WITH DEEPSTREAM 4.0

These are the major differences from DeepStream 4.0:

- Bounding box coordinates are now in float data type.

1.5 BREAKING CHANGES WITH DEEPSTREAM 5.1

These are major differences of the 5.1 release from DeepStream 5.0 and 5.0.1:

- The definition of `attributeLabel` (`nvdsinfer.h`) has changed from `const char *` to `char *`.

  Implementation of `NvDsInferClassiferParseCustomFunc` must now allocate strings on heap using `strdup` or equivalent and assign the address to `attributeLabel`.

2.0 LIMITATIONS

This section provides details about issues discovered during development and QA but not resolved in this release.

- With V4L2 codecs only MAX 1024 (decode + encode) instances are provided. The maximum number of instances can be increased by doing changes in open-source code.
- `detected-min-w` and `detected-min-h` must be set to values larger than 32 in the primary inference configuration file (`config_infer_primary.txt`) for `gst-dsexample` on Jetson.
- The Kafka protocol adapter sometimes does not automatically reconnect when the Kafka Broker to which it is connected goes down and comes back up, thereby requiring application restart.
- If the nvds log file, `ds.log`, has been deleted, then to restart logging you must delete the file `/run/rsyslogd.pid` within the container before reenabling logging by running the `setup_nvds_logger.sh` script as described in the "nvds_logger: Logging Framework" sub-section in the "Gst-nvmsgbroker" section in the *NVIDIA DeepStream Developer Guide*.
- On Jetson, running a DeepStream application over SSH (via putty) with X11 forwarding does not work.
- DeepStream currently expects model network width to be a multiple of 4 and network height to be a multiple of 2.
- Triton Inference Server implementation in DeepStream currently supports a single GPU. The models need to be configured to use a single GPU.
- For some models sometime output in DeepStream is not exactly same as observed in TAO Toolkit. This is due to input scaling algorithm differences.
- Dynamic resolution change support is Alpha quality.
- On the fly Model update only supports same type of Model with same Network parameters.
- On Jetson, BT709 input streams are not supported for sample application deepstream-transfer-learning-app and deepstream-image-meta-test.

- GStreamer sink elements sending upstream QOS events lead to memory leaks in the DeepStream pipeline. Applications should make sure to set the property "qos" to FALSE explicitly. For reference applications, this can be done by setting qos=0 in all sink groups of the configuration file.
- Multiple networks are simultaneously not supported on a single dla.
- The sample `objectDetector_FasterRCNN` works only with the default TensorRT plugins that comes as part of TensorRT installation. The samples do not work with opensource plugins library from https://github.com/NVIDIA/TensorRT.
- The YOLOV3, FasterRCNN, SSD, DSSD, RetinaNet and MaskRCNN models from TAO toolkit will require a TensorRT open-source build. To build TensorRT open-source plugins, refer to https://github.com/NVIDIA/TensorRT.
- deepstream-audio app stops processing audio buffers showing 0.0 FPS PERF log and may log error: `Generic low level error for audio test and zero fps`. Restart the pipeline when this happens.
- YoloV3 TAO toolkit model does not work with TensorRT 7.2.2 in this release.
- DeepStream cannot be installed on the current 16 GB Xavier NX production modules since Jetpack software takes the entire 16 GB emmc memory space. We recommend using Xavier NX developer kits with 32 GB SD card.

3.0 NOTES

- Optical flow is supported only on dGPUs having Turing architecture and on NVIDIA® Jetson AGX Xavier™ and NVIDIA® Jetson Xavier™ NX.
- NVIDIA® Jetson Nano™ and NVIDIA® Jetson™ TX2 support only FP16 and FP32 network precisions with NVIDIA® TensorRT™.
- Jetson AGX Xavier supports INT8, FP16 and FP32 network precisions with TensorRT.
- NVIDIA® DeepStream SDK 5.1 supports TAO 3.0 models (https://developer.nvidia.com/tao-toolkit). For more details, see https://github.com/NVIDIA-AI-IOT/deepstream_tao_apps.

3.1 APPLICATIONS MAY BE DEPLOYED IN A DOCKER CONTAINER

Applications built with DeepStream can be deployed using a Docker container, available on NGC (https://ngc.nvidia.com/). Sign up for an NVIDIA GPU Cloud account and look for `DeepStream` containers to get started.

After you sign into your NGC account, navigate to `Dashboard → Setup → Get API` key to get your `nvcr.io` authentication details.

As an example, you can use the DeepStream 5.1 docker containers on NGC and run the `deepstream-test4-app` sample application as an Azure edge runtime module on your edge device.

The following procedure deploys `deepstream-test4-app`:

- Using a sample video stream (sample_720p.h264)
- Sending messages with minimal schema
- Running with display disabled

- Using message topic `mytopic` (message topic may not be empty)

Set up and install Azure IoT Edge on your system with the instructions provided in the Azure module client README file in the `deepstream-5.1` package:

```
<deepstream-
5.1_package>/sources/libs/azure_protocol_adaptor/module_client/README
```

Note: **For the Jetson platform, omit installation of the Moby packages. Moby is currently incompatible with NVIDIA Container Runtime.**

See the Azure documentation for information about prerequisites for creating an Azure edge device on the Azure portal:

https://docs.microsoft.com/en-us/azure/iot-edge/how-to-deploy-modules-portal#prerequisites

To deploy deepstream-test4-app as an Azure IoT edge runtime module

1. On the Azure portal, click the IoT edge device you have created and click Set Modules.
2. Enter these settings:

Container Registry Settings:

Name: `NGC`
Address: `nvcr.io`
User name: `$oauthtoken`
Password: *use the password or API key from your NGC account*

Deployment modules:

Add a new module with the name `ds`

Image URI:

- For x86 dockers:

```
docker pull nvcr.io/nvidia/deepstream:5.1-21.02-devel
docker pull nvcr.io/nvidia/deepstream:5.1-21.02-samples
docker pull nvcr.io/nvidia/deepstream:5.1-21.02-iot
docker pull nvcr.io/nvidia/deepstream:5.1-21.02-base
docker pull nvcr.io/nvidia/deepstream:5.1-21.02-triton
```

- For Jetson dockers:

```
docker pull nvcr.io/nvidia/deepstream-l4t:5.1-21.02-samples
docker pull nvcr.io/nvidia/deepstream-l4t:5.1-21.02-iot
docker pull nvcr.io/nvidia/deepstream-l4t:5.1-21.02-base
```

Container Create options:

- For Jetson:

```
{
    "HostConfig": {
            "Runtime": "nvidia"
    },
    "WorkingDir": "
/opt/nvidia/deepstream/deepstream-
5.1/sources/apps/sample_apps/deepstream-test4",
    "ENTRYPOINT": [
        "/opt/nvidia/deepstream/deepstream-5.1/bin/deepstream-test4-
app",
        "-i", "/opt/nvidia/deepstream/deepstream-5.1/
samples/streams/sample_720p.h264",
        "-p",
"/opt/nvidia/deepstream/deepstream-
5.1/lib/libnvds_azure_edge_proto.so",
        "--no-display",
        "-s",
        "1",
        "--topic",
        "mytopic"
    ]
}
```

- For X86:

```
{
    "HostConfig": {
        "Runtime": "nvidia"
    },
    "WorkingDir": "/opt/nvidia/deepstream/deepstream-
5.1/sources/apps/sample_apps/deepstream-test4",
    "ENTRYPOINT": [
        "/opt/nvidia/deepstream/deepstream-5.1/bin/deepstream-test4-
app",
        "-i", "/opt/nvidia/deepstream/deepstream-
5.1/samples/streams/sample_720p.h264",
        "-p",
        "/opt/nvidia/deepstream/deepstream-
5.1/lib/libnvds_azure_edge_proto.so",
        "--no-display",
        "-s",
        "1",
        "--topic",
        "mytopic"
    ]}
```

3. Specify route options for the module:
   - Option 1: Use a default route where every message from every module is sent upstream.

```
    {
      "routes": {
        "route": "FROM /messages/* INTO $upstream"
      }
```

```
    }
```

- Option 2: Specific routes where messages sent upstream can be filtered based on topic name. For example, in the sample test programs, topic name `mytopic` is used for the module name `ds5`:

```
    {
        "routes": {
            "route": "FROM /messages/modules/ds5/outputs/mytopic INTO
$upstream"
        }
    }
```

3.2 SAMPLE APPLICATIONS MALFUNCTION IF DOCKER ENVIRONMENT CANNOT SUPPORT DISPLAY

If the Docker environment cannot support display, the sample applications `deepstream-test1`, `deepstream-test2`, `deepstream-test3`, and `deepstream-test4` do not work as expected.

Workaround:

To correct this problem, you must recompile the test applications after replacing `nveglglessink` with `fakesink`. With `deepstream-test4`, you also have the option to specify `fakesink` by adding the `--no-display` command line switch.

3.3 INSTALLING DEEPSTREAM ON JETSON

1. Download the NVIDIA SDK Manager to install JetPack 4.5.1 GA and DeepStream SDK.
2. Select all the JetPack 4.5.1 components and DeepStreamSDK from the "Additional SDKs" section.

Note:

- **NVIDIA Container Runtime" package shall be installed using JetPack 4.5.1 GA and is a pre-requisite for all DeepStream L4T docker containers.**
- **It is recommended to use SD card images for Jetson Nano instead of flashing through SDK Manager. The minimum recommended size for SD cards is 32 GB**

3.4 VIDEO CONTENT SPECIFIC PERFORMANCE DEGRADATION ON AMPERE GPU

Streams having complex encoding content, when used with DeepStream application might cause performance degradation when decoding large number of 1080p `H265` instances (typically more than >100 instances on GA 100).

In such case it appears that DeepStream app is stalled, but it takes time to initialize the decoders. Even after all decoders are up, sometimes poor utilization of decode engine is observed.

If there is enough diversity in the contents for the stream, there will not be any issue.

Workaround:

To circumvent the issue, you can sequentially launch individual decoder source bin with some delay in between and connect the corresponding bin to the sink pad of `nvstreammux` in the running state. Initially only one source should be connected in the pipeline, and the pipeline should be set into running state. After this, you can dynamically add the required number of sources/bins. This requires modification of application source code. Reference for adding/deleting source bin in pipeline is provided in runtime source add delete application here: https://github.com/NVIDIA-AI-IOT/deepstream_reference_apps/tree/master/runtime_source_add_delete.

3.5 TRITON INFERENCE SERVER IN DEEPSTREAM

Triton inference server on dGPU is supported only via docker container `deepstream:5.1-21.02-triton` for x86.

Refer to the *NVIDIA DeepStream Development Guide 5.1 Release* for more details about Triton inference server.

Triton inference server Supports following frameworks:

Framework	Tesla	Jetson	Notes / Limitations
TensorRT	Yes	Yes	Supports TensorRT plan or engine file (.plan)
TensorFlow	Yes	Yes	Supports TensorRT optimization Supported model formats: *GraphDef* or *SavedModel* Other TF formats such as checkpoint variables or estimators not directly supported
ONNX	Yes	No	Supports TensorRT optimization

PyTorch	Yes	No	PyTorch model must be traced with an example input and saved as a TorchScript Module (.pt)
Caffe2	No	No	Caffe2 Netdef models not supported with DeepStream - Triton

For more information refer to the following links:

- Triton inference server model repository: https://docs.nvidia.com/deeplearning/sdk/triton-inference-server-guide/docs/model_repository.html

  Also contains more information on the supported frameworks.

- TensorRT optimization in Triton inference server for ONNX and TensorFlow: https://docs.nvidia.com/deeplearning/sdk/triton-inference-server-guide/docs/optimization.html#framework-specific-optimization
- TensorFlow with TensorRT: https://docs.nvidia.com/deeplearning/frameworks/tf-trt-user-guide/index.html
- TensorFlow saved model: https://www.tensorflow.org/guide/saved_model#the_savedmodel_format_on_disk

Notice

Trademarks