DEEPSTREAM SDK 6.1 FOR NVIDIA DGPU/X86 AND JETSON

RN-09353-003 | May 19, 2022
Advance Information | Subject to Change

6.1 Release Notes

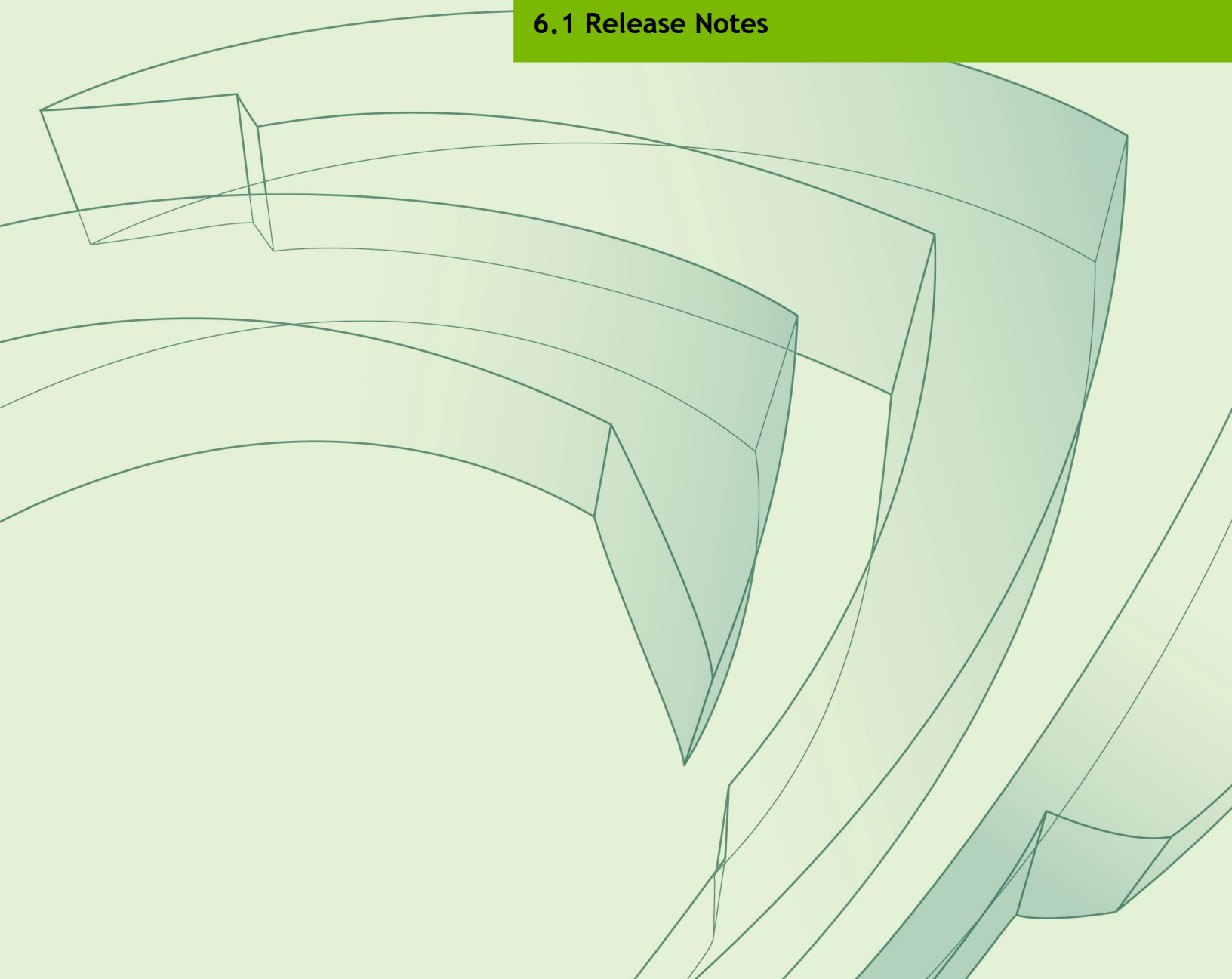

TABLE OF CONTENTS

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, and NVIDIA® Jetson AGX Orin™.

1.1 WHAT'S NEW

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

1.1.1 DS 6.1

- Supports Ubuntu 20.04 and GStreamer 1.16 version both on dGPU/x86 and Jetson.

Note: **DS 6.1 for Jetson is based on JP 5.0.1 DP which is for developer preview only.**

- Supports Triton 22.03.
- Stereo depth camera support.
- NMOS (Networked Media Open Specifications) support.
- Mellanox NIC support for transmitting Compressed/Uncompressed streams.
- UCX/RDMA support for efficient data transmission across multiple DeepStream pipelines running on different nodes.
- Post processing plugin to support inference post processing.
- Continued Support for 2D body pose estimation, facial landmark estimation, Emotion recognition, Gaze, Heart Rate, and Gesture. (https://github.com/NVIDIA-AI-IOT/deepstream_tao_apps branch: release/tao3.0_ds6.1ga).
- Enhancements in new Gst-nvstreammux plugin to support video conferencing use cases.

New nvstreammux can be enabled by exporting `USE_NEW_NVSTREAMMUX=yes`. For more information, see the "Gst-nvstreammux" section in the *NVIDIA DeepStream SDK Developer Guide 6.1 Release*.

- Enhancements in Gst-audio/video template plugins.
- Performance optimizations.
- Improved NVDCF tracker.
- NVIDIA TAO toolkit (previously called NVIDIA Transfer Learning Toolkit) Models from https://github.com/NVIDIA-AI-IOT/deepstream_tao_apps (branch: release/tao3.0_ds6.1ga) integrated into SDK.
- New sample applications in Python:
  - deepstream-preprocess-test - multi-stream pipeline using nvdspreprocess plugin with custom ROIs
  - deepstream-test3 – updated to support Triton inferencing in addition to TRT
- New plugins:
  - Gst-nvdsucx plugin to send and receive data over RDMA.
  - Gst-nvds3dfilter plugin for stereo depth camera.
  - Gst-nvdspostprocess plugin for having separate postprocessing on inference output.
  - Gst-nvdsmetautils contains two plugins - nvdsmetainsert and nvdsmetaextract, which can be used for NvDs or custom metadata serialization and de-serialization.

    Use case example: serialized custom metadata can be embedded into H264 bitstream, transmitted, received over remote host, decoded and de-serialized.
  - Gst-nvdsudpsink plugin for supporting Mellanox NIC for transmission. For more details Refer to the "Gst-nvdsudpsink" section in the *NVIDIA DeepStream SDK Developer Guide 6.1 Release*.
- New sample applications:
  - DeepStream NMOS Application: This application demonstrates how to create a DeepStream app as an NMOS Node.
  - DeepStream UCX test Applications:
    - DeepStream UCX test 1: Demonstrates how to use the communication plugin Gst-nvdsucx to send and receive video data over RDMA without any special metadata.
    - DeepStream UCX test 2: Demonstrates how to use the communication plugin Gst-nvdsucx to send and receive video/metadata data over RDMA along with the custom serialization and deserialization through the `libnvds_video_metadata_serialization.so` library.
    - DeepStream UCX test 3: Demonstrates how to use the communication plugin Gst-nvdsucx to send and receive audio/metadata data over RDMA using the

custom audio serialization and deserialization through the `libnvds_audio_metadata_serialization.so` library.

- DeepStream 3D Depth Camera Reference App: Demonstrates how to setup depth capture, depth render, 3D-point-cloud processing and 3D-points render pipelines over DS3D interfaces and custom-libs of `ds3d::dataloader`, `ds3d::datafilter` and `ds3d::datarender`.

DeepStream 6.0 Applications can be migrated to DeepStream 6.1. Refer to the "Application Migration to DeepStream 6.1 from DeepStream 6.0" section in the *NVIDIA DeepStream SDK Developer Guide 6.1 Release*.

1.1.2 Graph Composer 2.0.0

- Graph Execution Engine
  - Graph runtime to execute graphs implemented based on Graph Specification
  - Supported on Ubuntu 20.04 x86_64 and NVIDIA Jetson
- Graph composer tools
  - Composer with new UI
    - x86 only (Ubuntu 20.04)
    - User friendly editor view
    - Registry list in view
    - Graph edit with various options
    - Graph open/save
    - Property editor
    - Graph Launcher
    - Container Builder Launcher
    - Registry options
    - Extension Generator
    - Subgraph support
  - Registry CLI
    - Local and NVIDIA Cloud repository
    - Version management based on Semantic versioning
    - Command Line Interface tool
    - Graph install for graph deploy
  - Container Builder CLI
- New sample graphs
  - DeepStream 3D depth camera - Demonstrates how to setup depth capture, depth render, 3D-point-cloud processing and 3D-points render pipelines over DS3D

interfaces and custom-libs of `ds3d::dataloader`, `ds3d::datafilter` and `ds3d::datarender`.

- DeepStream UCX test 1: Demonstrates how to use the communication plugin Gst-nvdsucx to send and receive video data over RDMA without any special metadata.
- DeepStream UCX test 2: Demonstrates how to use the communication plugin Gst-nvdsucx to send and receive video/metadata data over RDMA, along with the custom serialization and deserialization through the `libnvds_video_metadata_serialization.so` library.

1.2 CONTENTS OF THIS RELEASE

This release includes the following:

- The DeepStream SDK. Refer to *NVIDIA DeepStream SDK Developer Guide 6.1 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 6.1 Release*.
  - For detailed troubleshooting information and frequently asked questions, see the "DeepStream Troubleshooting and FAQ Guide" section in the *NVIDIA DeepStream SDK Developer Guide 6.1 Release*.
- Graph Composer 2.0.0 and DeepStream reference graphs for dGPU and Jetson.
- 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 6.1 Release*
- *NVIDIA DeepStream SDK API Reference*
- *NVIDIA DeepStream Python API Reference*

1.4 DIFFERENCES WITH DEEPSTREAM 6.0

Gst-nvoverlaysink on Jetson is removed in the DS 6.1 release.

Note:	**OpenCV is deprecated by default. But you can enable OpenCV in plugins such as `nvinfer` (`nvdsinfer`) and `dsexample` (`gst-dsexample`) by setting `WITH_OPENCV=1` in the `Makefile` of these components. Refer to the component README for more instructions.**

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. This requires the application to restart.
- If the `nvds` log file `ds.log` has been deleted, 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. This is described in the "nvds_logger: Logging Framework" sub-section in the "Gst-nvmsgbroker" section in the *NVIDIA DeepStream Developer Guide 6.1 Release*.
- 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.

- DLA numbers on Jetson NX and Jetson AGX are lower compared to DeepStream-5.1 release. This is because, a few layers now run on DLA instead of GPU and are little slow on the DLA. This is done to free up the GPU.
- 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.
- Rivermax SDK is not part of DeepStream. So, the following warning is observed (gst-plugin-scanner:33257):

  ```
  GStreamer-WARNING **: 11:38:46.882: Failed to load plugin
  '/usr/lib/x86_64-linux-gnu/gstreamer-
  1.0/deepstream/libnvdsgst_udp.so': librivermax.so.0: cannot open
  shared object file: No such file or directory
  ```

  You can ignore this warning safely.

- Sample graphs containing `NvDsMultiSrcInput` component result in segmentation fault when an error occurs during graph/pipeline initialization.
- When using Composer WebSocket streaming, sometimes error like "`Error while sending buffer: invalid state`" is seen, or the window becomes unresponsive. Refreshing the browser page might fix it.
- Composer WebRTC Streaming is supported only on RTX GPUs.
- Int8 mode is not supported for DLA in this release.
- When running in unicast DNS-SD mode, there are three test cases in the AMWA NMOS Testing Tool “IS-04 Node API” test suite that the deepstream-nmos app cannot pass (test_15, test_16 and test_21). There is no workaround for this issue yet. But, the implementation works correctly when only one or two Registry instances can be discovered via unicast DNS-SD, and with any number of Registries in multicast DNS-SD mode.
- On jetson, when the screen is idle, fps is lowered for DeepStream applications. this behavior is by design to save power. However, if user does not want screen idle then refer to the FAQ for WAR.
- On Jetson AGX for large number of streams (>65 H265 1080P streams) sometimes message like “`NVDEC_COMMON: Get sync point failed`” is seen. In such case try to increase Ulimit (`$ulimit -S -n 8192`) and run the application.
- RDMA functionality only supported on x86 and that too in x86 devel docker for now.
- You cannot build the DeepStream out of the box on jetson dockers.

  However, DeepStream triton docker image for Jetsons could be used to build DeepStream source with the workarounds mentioned in the section: "Building DeepStream reference application source inside L4T triton container" in the *NVIDIA DeepStream Developer Guide 6.1 Release.*

- Optical flow plugin not supported on NVIDIA® Jetson AGX Orin™.
- There can be performance drop from TensorRT to Triton for some Models (5 to 15%).

3.0 NOTES

- Optical flow is supported only on dGPUs having Turing architecture (onwards) and on NVIDIA® Jetson AGX Xavier™ , NVIDIA® Jetson Xavier™ NX, and NVIDIA® Jetson AGX Orin™.
- NVIDIA® DeepStream SDK 6.1 supports TAO 3.0 models (https://developer.nvidia.com/tao-toolkit). For more details, see https://github.com/NVIDIA-AI-IOT/deepstream_tao_apps.
- Jetson AGX Orin would be useful compared to Jetson AGX Xavier for the cases where DeepStream performance on Jetson AGX Xavier is GPU bound.

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 6.1. 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-6.1` package:

```
<deepstream-
6.1_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:6.1-devel

     docker pull nvcr.io/nvidia/deepstream:6.1-samples

     docker pull nvcr.io/nvidia/deepstream:6.1-iot

     docker pull nvcr.io/nvidia/deepstream:6.1-base

     docker pull nvcr.io/nvidia/deepstream:6.1-triton
     ```

   - For Jetson dockers:

     ```
     docker pull nvcr.io/nvidia/deepstream-l4t:6.1-samples
     docker pull nvcr.io/nvidia/deepstream-l4t:6.1-iot
     ```

```
docker pull nvcr.io/nvidia/deepstream-l4t:6.1-samples

docker pull nvcr.io/nvidia/deepstream-l4t:6.1-triton
```

Container Create options:

- For Jetson:

```
{
    "HostConfig": {
            "Runtime": "nvidia"
    },
    "WorkingDir": "
/opt/nvidia/deepstream/deepstream-
6.1/sources/apps/sample_apps/deepstream-test4",
    "ENTRYPOINT": [
        "/opt/nvidia/deepstream/deepstream-6.1/bin/deepstream-test4-
app",
        "-i", "/opt/nvidia/deepstream/deepstream-6.1/
samples/streams/sample_720p.h264",
        "-p",
"/opt/nvidia/deepstream/deepstream-
6.1/lib/libnvds_azure_edge_proto.so",
        "--no-display",
        "-s",
        "1",
        "--topic",
        "mytopic"
    ]
}
```

- For X86:

```
{
    "HostConfig": {
        "Runtime": "nvidia"
    },
    "WorkingDir": "/opt/nvidia/deepstream/deepstream-
6.1/sources/apps/sample_apps/deepstream-test4",
    "ENTRYPOINT": [
        "/opt/nvidia/deepstream/deepstream-6.1/bin/deepstream-test4-
app",
        "-i", "/opt/nvidia/deepstream/deepstream-
6.1/samples/streams/sample_720p.h264",
        "-p",
        "/opt/nvidia/deepstream/deepstream-
6.1/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 `ds`:

```
    {
        "routes": {
            "route": "FROM /messages/modules/ds/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 5.0.1 DP.
2. Select all the JetPack 5.0.1 components except DeepStreamSDK from the "Additional SDKs" section.

   Refer to the "Quick Start Guide" section in *NVIDIA DeepStream SDK Guide 6.1 Release* for updating additional BSP library. Continue with the DeepStream installation instructions after the BSP update.

Note: **NVIDIA Container Runtime package shall be installed using JetPack 5.0.1 and is a pre-requisite for all DeepStream L4T docker containers.**

3.4 TRITON INFERENCE SERVER IN DEEPSTREAM

Triton inference server on dGPU is supported only via docker `deepstream-l4t:6.1-triton` for x86. On Jetson we support that with or without docker.

Refer to the *NVIDIA DeepStream Development Guide 6.1 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 Supports both Tensorflow 1.x and Tensorflow 2.x. Triton defaults to use Tensorflow 1.x. If users need to run Tensorlfow 2.x models, need to update plugin config with: `infer_config{ backend { trt_is { model_repo{ backend_configs { backend: "tensorflow" setting: "version" value: "2" } } } }`
ONNX	Yes	Yes	Supports TensorRT optimization
PyTorch	Yes	No	PyTorch model must be traced with an example input and saved as a TorchScript Module (.pt)

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

3.5 DEPRECATION PLAN IN THE UPCOMING RELEASES

- `Gst-nveglglessink` plugin will be deprecated from next release.
- HW mode for `Gst-nvdsosd` is deprecated and will not be supported from next release.

Notice

Trademarks