



DEEPSTREAM SDK 7.1 FOR NVIDIA DGPU/X86 AND JETSON

RN-09353-003 | October 17, 2024
Advance Information | Subject to Change

7.1 Release Notes



TABLE OF CONTENTS

- 1.0 ABOUT THIS RELEASE 3**
 - 1.1 What’s New 3
 - 1.1.1 DS 7.1 3
 - 1.1.2 DS 7.0 (Previous Release) 4
 - 1.1.3 Graph Composer 4.1.0 5
 - 1.2 Contents of this Release 7
 - 1.3 Documentation in this Release 7
 - 1.4 Differences With Deepstream 6.1 and Above 7
 - 1.5 Breaking Changes 8
- 2.0 LIMITATIONS 10**
- 3.0 NOTES 13**
 - 3.1 Applications May Be Deployed in a Docker Container 13
 - 3.2 Sample Applications Malfunction if Docker Environment Cannot Support Display 16
 - 3.3 Installing Deepstream On Jetson 17
 - 3.4 Triton Inference Server In Deepstream 17

1.0 ABOUT THIS RELEASE

These release notes are for the NVIDIA® DeepStream SDK for NVIDIA® Tesla®, NVIDIA® Ampere®, NVIDIA® Hopper®, NVIDIA® Ada Lovelace®, NVIDIA® Jetson AGX Orin™, NVIDIA® Jetson Orin™ NX, and NVIDIA® Jetson Orin™ Nano.

1.1 WHAT'S NEW

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

1.1.1 DS 7.1

- ▶ Supports Triton 24.08 and Rivermax v1.40/v1.50.
- ▶ Jetson package based on JP 6.1 (r36.4 BSP).
- ▶ New Service maker framework in Python (Alpha): New application layer that removes the need to understand GStreamer application programming paradigm and enable to use Python.
- ▶ Support Gray 16 LE type.
- ▶ Postprocessing plugin to support output tensor meta from custom preprocessing
- ▶ Support Access to tensor metadata with Service Maker C++ APIs
- ▶ nvvideoconvert to support UYVY (8-bit YCbCr-4:2:2) on x86/dGPU
- ▶ Enhanced Single-View 3D Tracking.
- ▶ Improved ReID Accuracy in Tracker.
- ▶ NVIDIA TAO toolkit (previously called NVIDIA Transfer Learning Toolkit) models from https://github.com/NVIDIA-AI-IOT/deepstream_tao_apps (branch: release/tao_ds7.1ga) integrated into SDK.
- ▶ Improved stability.
- ▶ Source code release
 - Nvuriscbin

- Message broker: protocol adapter code and nvmsgbroker API code release (amqp, azure, kafka, mqtt, redis, nvmsgbroker)
- ▶ Python bindings and samples updates:
 - Build system update: new build system using PyPA to support pip 24.2
 - Pybind11 version updated to v2.13.0
 - New bindings:
 - NvDsObjEncOutParams, NvDsObjEncUsrArgs
 - nvds_obj_enc_create_context(), nvds_obj_enc_process(), nvds_obj_enc_finish(), nvds_obj_enc_destroy_context()
 - NvDsAnalyticsObjInfo.objStatus
 - NvDsObjReid

1.1.2 DS 7.0 (Previous Release)

- ▶ Supports Triton 23.10 for x86/dGPU, Triton 24.03 for Jetson and Rivermax v1.40.
- ▶ Jetson package based on JP 6.0 (r36.3 BSP).
- ▶ New DeepStream libraries in Python (Alpha):
- ▶ All Goodness of DeepStream, CV-CUDA, Video and Image Codecs packaged as Python APIs to easily integrate into custom frameworks.
- ▶ New Service maker framework (Alpha): New application layer that removes the need to understand GStreamer application programming paradigm.
- ▶ Sensor Fusion with BEVFusion – Support BEVFusion with DeepStream existing 3D Framework.
- ▶ WSL2 support (Alpha): Remove dependency on a dedicated Linux box for development.
- ▶ PipeTuner 1.0 (Alpha): Automate DeepStream pipeline optimization using annotated data and user videos.
- ▶ Support for ARM SBSA platform (Alpha).
- ▶ Support for IGX platform (Alpha).
- ▶ Enhanced REST API support to control DeepStream pipeline on-the-fly.
- ▶ Enhanced Single-View 3D Tracking.
- ▶ Enhanced NvDCF tracker support with PVA backend on Jetson.
- ▶ Improved ReID Accuracy in Tracker.
- ▶ Enhanced fault-tolerance of Sub-batching for Tracker.
- ▶ Enhancements in new Gst-nvstreammux plugin. 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 7.0 Release.
- ▶ Performance optimizations.
- ▶ NVIDIA TAO toolkit (previously called NVIDIA Transfer Learning Toolkit) models from https://github.com/NVIDIA-AI-IOT/deepstream_tao_apps (branch: release/tao5.3_ds7.0ga) integrated into SDK.

- ▶ 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/tao5.3_ds7.0ga).
- ▶ Improved stability.
- ▶ New Samples:
 - deepstream-ipc-test: Demonstrates decoder buffer sharing IPC use-case on Jetson platform for live streams to optimize NVDEC HW utilization. This example uses IPC sink and IPC source element to interconnect GStreamer pipelines that run in different processes.
 - deepstream-3d-lidar-sensor-fusion application to support `bevfusion` and `v2xfusion` for multi sensors fusion.
- ▶ Python bindings and samples updates:
 - ARB SBSA support.
 - Deepstream_test3 app: pipeline component latency measurement through `nvds_measure_buffer_latency()` binding.
 - Check for integrated vs. discrete GPU.
 - Support RGB color format for `nvbufsurface` GPU buffer access.

DeepStream 7.0 Applications can be migrated to DeepStream 7.1 Refer to the “Application Migration to DeepStream 7.1 from DeepStream 7.0” section in the *NVIDIA DeepStream SDK Developer Guide 7.1 Release*.

1.1.3 Graph Composer 4.1.0

Graph Composer 4.1.0 has multiple new features added to the sdk along with the updated compute stack.

Graph Execution Engine

- ▶ Supported on Ubuntu 22.04 x86_64 and NVIDIA Jetson.
- ▶ Version updated to 4.1.0.
 - Graph Composer
- ▶ Version updated to 4.1.0.
- ▶ x86 only - Ubuntu 22.04 and Windows 10.

Container Builder

- ▶ No change in version.
 - Registry
- ▶ No change in version.
 - Extensions update
- ▶ Minor version of all the extensions is updated.

Following deprecated API's have been removed

1. `gxf_result_t GxfLoadExtension(gxf_context_t context, const char* filename);`
2. `gxf_result_t GxfLoadExtensionManifest(gxf_context_t context, const char* manifest_filename);`
3. `gxf_result_t GxfEntityCreate(gxf_context_t context, gxf_uid_t* eid);`

Following extension and examples are deprecated:

Extensions

- ▶ `NvDsBodyPose2D`
- ▶ `NvDsEmotionExt`
- ▶ `NvDsFacialLandmarks`
- ▶ `NvDsGazeExt`
- ▶ `NvDsGesture`
- ▶ `NvDsHeartRateExt`

Examples

- ▶ Audio model and examples

Cuda & RMM Extension updates

4. A new memory allocator interface, named `CudaAllocator`, is added in `CudaExtension` to support async memory allocation / deallocation operations. A new memory allocator, named `StreamOrderedAllocator`, which implements the `CudaAllocator` interface is based on cuda runtime's stream ordered memory allocator.
5. A new GXF extension, named `RMMExtension`, based on nvidia Rapids Memory Manager framework has been added. RMM Allocator also implements the new aysnc memory allocation interface.
6. A new buffer type, named `CudaBuffer`, was added to support async device workloads.
7. New scheduling terms `CudaStreamSchedulingTerm`, `CudaEventSchedulingTerm`, `CudaBufferAvailableSchedulingTerm` have been added to manage device workloads effectively.

Performance Improvements

8. Support for removing a component from an entity has been added. This improves reuse of message entities and thereby increases performance as well.
9. The value of `KMaxComponents` constant reduced to 1024 from 10240 which reduces usage stack memory primarily used by fixed vectors. Support for `HeapBased FixedVectors` added to further reduce stack memory usage for large vectors.
10. Multiple perf improvements in the c++ api's for creating and querying components in a message entity. No code changes are needed to realize these improvements.

Tooling Improvements

11. GXF Server is now wrapped in a systemd service, improving ease of use for Graph Composer tools suite.

All files and configs used by GXF registry is now moved to `/var/tmp/gxf`

1.2 CONTENTS OF THIS RELEASE

This release includes the following:

- ▶ The DeepStream SDK. Refer to *NVIDIA DeepStream SDK Developer Guide 7.1 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 and metadata usage, see the “DeepStream Plugin Guide” section in the *NVIDIA DeepStream SDK Developer Guide 7.1 Release*.
 - For detailed troubleshooting information and frequently asked questions, see the “DeepStream Troubleshooting and FAQ Guide” section in the *NVIDIA DeepStream SDK Developer Guide 7.1 Release*.
- ▶ Graph Composer 4.1.0 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 7.1 Release*
- ▶ *NVIDIA DeepStream SDK API Reference*
- ▶ *NVIDIA DeepStream Python API Reference*

1.4 DIFFERENCES WITH DEEPSTREAM 6.1 AND ABOVE

`gststreamer1.0-libav`, `libav`, OSS encoder, decoder plugins (`x264/x265`) and `audioparsers` packages are removed in DeepStream dockers from DeepStream 6.1. You may install these packages based on your requirement (`gststreamer1.0-plugins-good/` `gststreamer1.0-plugins-bad/` `gststreamer1.0-plugins-ugly`). While running DeepStream applications inside dockers, you may see the following warnings:

```
WARNING from src_elem: No decoder available for type 'audio/mpeg,
mpegversion=(int)4, framed=(boolean)true, stream-format=(string)raw,
level=(string)2, base-profile=(string)lc, profile=(string)lc,
codec_data=(buffer)119056e500, rate=(int)48000, channels=(int)2'.
```

```
Debug info: gsturidecodebin.c(920): unknown_type_cb ():
```

To avoid such warnings, install `gststreamer1.0-libav` and `gststreamer1.0-plugins-good` inside docker.

Specifically, for `deepstream-nmos`, `deepstream-avsync-app` and python based `deepstream-imagedata-multistream` app you would need to install `gststreamer1.0-libav` and `gststreamer1.0-plugins-good`.

`Gst-nveglglessink` plugin is deprecated. Use `Gst-nv3dsink` plugin for Jetson instead.

1.5 BREAKING CHANGES

- ▶ From DeepStream 7.1, the following models are deprecated.
 - Facetetect, FacedetectIR, PeopleSegnet, bodyposenet, gesturennet,emotionnet, hearratenet, gazenet, facial landmark estimation models and its corresponding applications.
 - Yolo OSS, SSD, DSSD, Yolov3, Yolov4, Yolov4-tiny, Fasterrcnn, Densenet models and it's corresponding applications.
 - `deepstream-mrcnn-test`: Instead use use github app. README contains instructions to use github app.
 - `deepstream-segmentation-test` application from SDK- Use github app. README contains instructions to use github app.
 - `deepstream-segmentation` Python sample application.
- ▶ DeepStream Audio support, ASR, TTS plugin deprecated. User to use Nvidia RIVA speech sdk.
- ▶ Deprecated support for uff and caffe models.
- ▶ All the models packaged in SDK are now onnx models.
- ▶ `tao-converter` is removed because TAO toolkit does not support tlt, uff models anymore.
- ▶ For Protocol adapters - Password through config file will be deprecated from the next release.
- ▶ Some of the open-source libraries related to Codecs have been removed so user might see some warnings as below, which can be ignored safely:


```
(gst-plugin-scanner:1433): GStreamer-WARNING **: 18:05:56.454: Failed to
load plugin '/usr/lib/aarch64-linux-gnu/gstreamer-1.0/libgstfaad.so':
libfaad.so.2: cannot open shared object file: No such file or directory
/bin/bash: line 1: lsmod: command not found
/bin/bash: line 1: modprobe: command not found
```

2.0 LIMITATIONS

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

- ▶ DeepStream on Jetson is based on L4T BSP version r36.4. Refer to “Known Issues” section in [Jetson release notes](#).
- ▶ With V4L2 codecs only MAX 1024 (decode + encode) instances are provided. The maximum number of instances can be increased by making 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 7.1 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 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 the same type of Model with same Network parameters.

- ▶ 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.

- ▶ When using Composer WebSocket streaming, sometimes errors 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.
- ▶ 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.
- ▶ RDMA functionality is only supported on x86 and only in x86 Triton docker for now.
- ▶ You cannot build the DeepStream out of the box on Jetson dockers except its Triton variant.
- ▶ There can be performance drop from TensorRT to Triton for some models (5 to 15%). In such cases, user may want to use `nvinfer` plugin instead `nvinferserver` plugin.
- ▶ NVRM: XID errors seen sometimes when running 200+ streams on Ampere, Hopper and ADA.
- ▶ NVRM: XID errors seen on some setups with `gst-dsexample` and transfer learning sample apps.
- ▶ Sometimes during `deepstream-testsr` app execution, assertion "GStreamer-CRITICAL **: 12:55:35.006: gst_pad_link_full: assertion 'GST_IS_PAD sinkpad)' failed" is seen which can be safely ignored.
- ▶ For some of the models during engine file generation, error "[TRT]: 3: [builder.cpp::~Builder::307] Error Code 3: API Usage Error" observed from TensorRT, but has no impact on functionality and can be safely ignored.
- ▶ `deepstream-server` app is not supported with new `nvstreammux` plugin.
- ▶ TAO point-pillar model works only in FP32 mode.
- ▶ REST API support for few components (`decoder`, `preprocessor`, `nvinfer` along with stream addition deletion support) with limited configuration options. However, you can extend the functionality with the steps mentioned in SDK documentation.
- ▶ With Basler camera, on Jetson, only images with width of multiple of 4 supported.
- ▶ Sometimes, error "GLib (gthread-posix.c): Unexpected error from C library during 'pthread_setspecific': Invalid argument" is seen while running DeepStream applications.

The issue is caused because of a bug in `glib 2.0-2.72` version which comes with `ubuntu22.04` by default. The issue is addressed in `glib2.76` and its installation is required to fix the issue (<https://github.com/GNOME/glib/tree/2.76.6>).

- ▶ `deepstream-lidar-inference-app` sample app fails to run on Jetson. The issue is caused because of a bug in `glib 2.0-2.72` version which comes with `ubuntu22.04` by

default. The issue is addressed in `glib2.76` and its installation is required to fix the issue (<https://github.com/GNOME/glib/tree/2.76.6>).

- ▶ In some cases, performance with Python sample apps may be lower than C version.
- ▶ while running `deepstream-opencv-test` app, warning `"gst_caps_features_set_parent_refcount: assertion 'refcount == NULL' failed"` observed. No impact on functionality & can be safely ignored.
- ▶ Observing below errors for Jetson dockers (but no impact on functionality)
 - a) While decoding: `/bin/dash: 1: lsmod: not found` and `/bin/dash: 1: modprobe: not found`.
 - b) At start pipeline: Failed to detect NVIDIA driver version.
- ▶ Performance drop for some Models seen with `nvInferServer` in `gRPC` mode when run on ARM SBSA w.r.t to x86.
- ▶ Minor perf drop observed w.r.t to DS 6.4 release with `NvDCF` perf configuration. In this case Setting environment variable `NVDS_DISABLE_CUDADEV_BLOCKINGSYNC=1` helps improve performance.
- ▶ DeepStream Applications fail to run with `nv3dsink` on ARM SBSA platforms like Grace Hopper, OOB. User should use other sink types like `fakesink` or `RTSP` out. Or refer to https://docs.nvidia.com/metropolis/deepstream/dev-guide/text/DS_Installation.html/#ds-installation-sbsa-known-limitation to enable display.
- ▶ For Azure, messages sent not matching with messages received on server side.
- ▶ Performance in WSL is not at par with Ubuntu system.
- ▶ Image encode not supported in WSL.
- ▶ For IGX when used in `dGPU` mode, `Xid` errors can be seen for 50+ streams.
- ▶ IPC sample application `deepstream-ipc-test-app` does not reliably work on Orin Nano & NX variants. It throws `"UnixSocketConnect: Waiting timed out"` error.
- ▶ On Jetson, engine file generation fails occasionally inside container. This is due to a bug in `TensorRT-10.3` version. The issue is fixed in `TensorRT-10.5` version. So, to overcome this issue, you should either install `TensorRT-10.5` inside the docker or generate the engine file outside docker, on baremetal, and copy it inside the docker.
- ▶ The error "Failed to query video capabilities: Invalid argument" is observed while running DeepStream applications. You can safely ignore it.
- ▶ `Numpy 2.x` is not supported by `PyDS`.

3.0 NOTES

- ▶ Optical flow is supported only on dGPUs having Turing architecture (onwards) and NVIDIA® Jetson Orin™ family.
- ▶ REST API commands only work after the video shows up on the host screen.
- ▶ NVIDIA® DeepStream SDK 7.1 supports TAO model-based applications (<https://developer.nvidia.com/tao-toolkit>). For more details, see https://github.com/NVIDIA-AI-IOT/deepstream_tao_apps (branch: `release/tao_ds7.1ga`).
- ▶ On vGPU, only CUDA device memory `NVBUF_MEM_CUDA_DEVICE` supported.

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.**
- **When using docker make sure `libopencv-dev` package is installed inside docker if the Application requires it.**

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 `nvcv.io` authentication details.

As an example, you can use DeepStream 7.1 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-7.1` package:

```
<deepstream-7.1_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

12. On the Azure portal, click the IoT edge device you have created and click `Set Modules`.
13. 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:7.1-gc-triton-devel
```

Multi-Arch dockers for x86 and Jetson:

- For x86:

```
docker pull --platform linux/amd64 nvcr.io/nvidia/deepstream:7.1-triton-multiarch

docker pull --platform linux/amd64 nvcr.io/nvidia/deepstream:7.1-samples-multiarch
```

- For Jetson:

```
docker pull --platform linux/arm64 nvcr.io/nvidia/deepstream:7.1-triton-multiarch
docker pull --platform linux/arm64 nvcr.io/nvidia/deepstream:7.1-samples-multiarch
```

Container Create options:

- For Jetson:

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

- For X86:

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

```

```

        "-p",
"/opt/nvidia/deepstream/deepstream/lib/libnvds_azure_edge_proto.so",
        "--no-display",
        "-s",
        "1",
        "--topic",
        "mytopic"
    ]}

```

14. 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` on x86 and `nv3dsink` on Jetson with `fakesink`. With `deepstream-test4`, you also have the option to specify `fakesink` by adding the `--no-display` command line switch.

Alternatively virtual display can be used. For more information refer to “How to visualize the output if the display is not attached to the system” section in “Quick Start Guide” section of *NVIDIA DeepStream Developer Guide 7.1 Release*.

3.3 INSTALLING DEEPSTREAM ON JETSON

1. Download the NVIDIA SDK Manager to install JetPack 6.1.
2. Select all the JetPack 6.1 components except DeepStreamSDK from the “Additional SDKs” section.

Refer to the “Quick Start Guide” section in *NVIDIA DeepStream Developer Guide 7.1 Release* to update additional BSP libraries if available. Continue with the DeepStream installation instructions after the BSP update.

Note: NVIDIA Container Runtime package shall be installed using JetPack 6.1 and is a pre-requisite for all DeepStream L4T docker containers.

3.4 TRITON INFERENCE SERVER IN DEEPSTREAM

Triton inference server (version 24.08) on dGPU is supported only via docker `deepstream:7.1-triton-multiarch` for x86. On Jetson, version 24.08 is supported with or without docker.

Refer to the *NVIDIA DeepStream Development Guide 7.1 Release* for more details about Triton inference server.

Triton inference server Supports following frameworks:

Framework	Test a	Jetson	Notes / Limitations
TensorRT	Yes	Yes	Supports TensorRT plan or engine file (*.plan, *.engine) - Triton model config.pbtxt for TensorRT engine file format <pre>platform: "tensorrt_plan" default_model_filename: "model.engine" input [...] output [...]</pre> Triton-TensorRT backend documentation: https://github.com/triton-inference-server/tensorrt_backend
TensorFlow	Yes	Yes	- Supports Tensorflow 2.x (Tensorflow 1.x is deprecated) - Supports TF-TensorRT optimization

			<ul style="list-style-type: none"> - Supported model formats: <i>GraphDef</i> or <i>SavedModel</i> - Other TF formats such as checkpoint variables or estimators are not directly supported - Triton model config.pbtxt for Graphdef format platform: "tensorflow_graphdef" default_model_filename: "model.graphdef" - Triton model config.pbtxt for Graphdef format platform: " tensorflow_savedmodel" " default_model_filename: "model.savedmodel" <p>Triton Tensorflow backend documentation: https://github.com/triton-inference-server/tensorflow_backend</p>
ONNX	Yes	Yes	<ul style="list-style-type: none"> - Supports ONNX model - Supports ONNX TensorRT optimization <p>Triton model config.pbtxt for ONNX</p> <pre>platform: "onnxruntime_onnx" default_model_filename: "model.onnx" # [optional: TensorRT optimization, disabled by default] optimization { execution_accelerators { gpu_execution_accelerator : [{ name : "tensorrt" parameters { key: "precision_mode" value: "FP16" } parameters { key: "max_workspace_size_bytes" value: "1073741824" }}] }}</pre> <p>Triton ONNXRuntime backend documentation: https://github.com/triton-inference-server/onnxruntime_backend</p>
PyTorch(TorchScript)	Yes	Yes	<ul style="list-style-type: none"> - Support TorchScript models(file format *.pt), PyTorch model must be traced and saved as a TorchScript Model (.pt) <p>Triton model config.pbtxt for TorchScript format</p> <pre>backend: "pytorch"</pre>

			<pre>platform: "pytorch_libtorch" default_model_filename: "model.pt" input [{ name: "INPUT0" }] output [{ name: "OUTPUT0" } { name: "OUTPUT1" }]</pre> <p>Triton Pytorch backend documentation: https://github.com/triton-inference-server/pytorch_backend</p>
Python Backend	Yes	Yes	<ul style="list-style-type: none"> - Support Custom Triton-Python backend - Support Python Custom conda Execution Environment <p>Triton model config.pbtxt for Python file format</p> <pre>backend: "python" default_model_filename: "model.py" # [optional: custom conda env, disabled by default] parameters: { key: "EXECUTION_ENV_PATH", value: {string_value: "\$\$TRITON_MODEL_DIRECTORY/python3.6.tar.gz"} }</pre> <p>Triton Python backend documentation: https://github.com/triton-inference-server/python_backend</p>
Ensemble Models	Yes	Yes	<ul style="list-style-type: none"> - Support Triton Ensemble Models to connect multiple model inference graph <p>Triton model config.pbtxt for ensemble model</p> <pre>platform: "ensemble" input[...] output[...] ensemble_scheduling { step [{model_name: "model_A"}, {model_name: "model_B"}, {model_name: "model_C"},] }</pre> <p>Triton Ensemble Model documentation: https://github.com/triton-inference-server/server/blob/main/docs/user_guide/architecture.md#ensemble-models</p>

For more information refer to the following links:

- ▶ Triton inference server documentation entry: <https://github.com/triton-inference-server/server>
- ▶ Triton inference server model repository: https://github.com/triton-inference-server/server/blob/main/docs/user_guide/model_repository.md
- ▶ Triton inference server supported backends and QA: <https://github.com/triton-inference-server/backend>
- ▶ Triton Model TensorRT optimization for ONNX and TensorFlow: https://github.com/triton-inference-server/server/blob/main/docs/user_guide/optimization.md#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

THE INFORMATION IN THIS DOCUMENT AND ALL OTHER INFORMATION CONTAINED IN NVIDIA DOCUMENTATION REFERENCED IN THIS DOCUMENT IS PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE INFORMATION FOR THE PRODUCT, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the product described in this document shall be limited in accordance with the NVIDIA terms and conditions of sale for the product. THE NVIDIA PRODUCT DESCRIBED IN THIS DOCUMENT IS NOT FAULT TOLERANT AND IS NOT DESIGNED, MANUFACTURED OR INTENDED FOR USE IN CONNECTION WITH THE DESIGN, CONSTRUCTION, MAINTENANCE, AND/OR OPERATION OF ANY SYSTEM WHERE THE USE OR A FAILURE OF SUCH SYSTEM COULD RESULT IN A SITUATION THAT THREATENS THE SAFETY OF HUMAN LIFE OR SEVERE PHYSICAL HARM OR PROPERTY DAMAGE (INCLUDING, FOR EXAMPLE, USE IN CONNECTION WITH ANY NUCLEAR, AVIONICS, LIFE SUPPORT OR OTHER LIFE CRITICAL APPLICATION). NVIDIA EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR SUCH HIGH RISK USES. NVIDIA SHALL NOT BE LIABLE TO CUSTOMER OR ANY THIRD PARTY, IN WHOLE OR IN PART, FOR ANY CLAIMS OR DAMAGES ARISING FROM SUCH HIGH RISK USES.

NVIDIA makes no representation or warranty that the product described in this document will be suitable for any specified use without further testing or modification. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to ensure the product is suitable and fit for the application planned by customer and to do 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 does not accept any 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.

Other than the right for customer to use the information in this document with the product, no other license, either expressed or implied, is hereby granted by NVIDIA under this document. Reproduction of information in this document is permissible only if reproduction is approved by NVIDIA in writing, is reproduced without alteration, and is accompanied by all associated conditions, limitations, and notices.

Trademarks

NVIDIA, the NVIDIA logo, TensorRT, NVIDIA Ampere, NVIDIA Hopper and NVIDIA Tesla 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 © 2024 NVIDIA CORPORATION & AFFILIATES. All rights reserved.